

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

Fair and Efficient Transmission Over Gbps Dual Ring Networks

by
Abdelnaser Mohammad Adas

The advances in fiber optics technology provide large bandwidth and enable the support of a wide variety of services. New network architectures have been proposed, such as Metaring and Distributed Queue Dual Bus (DQDB), that try to take advantage of the new capabilities. Because of the very small packet transmission time relative to the feedback time a challenging issue in high speed networks is the efficient and fair share of the channel bandwidth among the competing users. In this thesis we first investigate and compare the performance of the Global and Local Fairness Mechanisms (GFM and LFM, respectively). They have been proposed recently for fair bandwidth allocation in high speed dual ring networks employing destination release. (a slot that has been read by its destination is immediately released and can be used again by other nodes). We show the sensitivity of both mechanisms to various system parameters, such as channel bandwidth and ring latency. We introduce the Dynamic Medium Access Control Mechanism (DMAC) which does not suffer from the limitations of GFM and LFM, introduces fairness in a very effective and efficient way, and is insensitive to the network parameters.

FAIR AND EFFICIENT TRANSMISSION
OVER GBPS DUAL RING
NETWORKS

by

Abdelnaser M. Adas

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering


Department of Electrical and Computer Engineering
May 1993

APPROVAL PAGE

Fair and Efficient Transmission
Over Gbps Dual Ring
Networks

Abdelnaser M. Adas

~~Dr. Dennis Karvelas~~
Assistant Professor of Computer Science, NJIT

Dr. Anthony Robbi 
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Sotirios Ziavras
Assistant Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Abdelnaser Mohammad Adas

Degree: Master of Science in Electrical Engineering

Date: May 1993

Undergraduate and Graduate Education

- Master of Science in Electrical Engineering,
New Jersey Institute of Technology, Newark, NJ, 1993
- Bachelor of Science in Electrical Engineering,
University of Jordan, Amman, Jordan, 1988

Major: Electrical Engineering

This thesis is dedicated to
My Father and Mother

ACKNOWLEDGMENT

The author wishes to express his sincere gratitude to his supervisor, Assistant Professor Dennis Karvelas, for his guidance, friendship, and moral support throughout this research.

Special thanks go to Associate Professor Anthony Robbi and Assistant Professor Sotirios Ziavras for serving as members of the committee.

The author would like to express his gratitude to his dear friend Magd Donia for his help with thesis formatting and his moral support.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION.....	1
2 METARING NETWORK.....	7
3 METARING FAIRNESS MECHANISMS	12
3.1 Global Fairness Mechanism (GFM)	12
3.2 Local Fairness Mechanism	16
3.2.1 LFM Properties	20
3.2.2 Throughput Analysis.....	23
4 A NEW DYNAMIC MAC MECHANISM.....	25
4.1 Dual Bus Mechanisms for MANs	25
4.2 The Dynamic MAC Mechanism for Dual Ring MANs.....	29
4.2.1 DMAC Implementation.....	30
4.2.2 The DMAC Operation	35
4.2.3 Advantages of DMAC Mechanism.....	40
5 PERFORMANCE ANALYSIS.....	42
6 CONCLUSIONS	49
APPENDICES	51
A- GFM Simulation Program	52
B- LFM Simulation Program.....	58
C- DMAC Simulation Program	70
REFERENCES	80

LIST OF FIGURES

Figure	Page
1 Dual Ring Architecture.....	7
2 Node Starvation.....	11
3 Three Independent Subsets of Users That Communicate Only Among Themselves.....	15
4 The Basic LFM Operation For Slotted Mode.....	17
5 Heavily And Lightly Loaded Nodes In The Same REQUEST PATH.....	20
6 Distances Between The Interfering Nodes Are Not Similar	22
7 DQDB Dual Bus Architecture.....	26
8 DMAC Node Components That Control The Transmission On Ring A.....	32
9 A Pseudo Code For DMAC Algorithm.....	40
10 Asymmetric Location Of The Nodes And Asymmetric Load.....	43
11 Traffic Of Lightly And Heavily Loaded Nodes Interfere.....	44
12 Localized Pattern Scenario.....	46

LIST OF TABLES

Table	Page
1 Throughput Performance For Traffic Scenario 2	43
2 Throughput Performance For Traffic Scenario 3	45
3 Throughput Performance For Traffic Scenario 4	47
4 Throughput Performance For Traffic Scenario 5	47

CHAPTER 1

INTRODUCTION

The increasing need to share computing resources and information has led to the significant growth of the Local Area Network (LAN) industry. LANs allow personal computers (PCs), hosts, and peripheral devices to communicate with each other, in a relatively small geographical area operating at various transmission speeds up to 20 mega bits per second (Mbps).

In the past, dumb terminals and low speed desktop systems were the main digital communication devices. Networks exclusively carried digital data. Therefore, using a 16 Mbps LAN to connect these devices was considered to be sufficient. Today, due to the advancements in information technology, many types of information are carried in a digital form including human voice, images, video, music, and facsimile. These types of information require large bandwidths that current LANs cannot support. Another very important development is the advancement in computer technology that enables desktop systems to achieve speeds of 50 million instructions per second (MIPS). In contrast, in the mid 80's, only main frames could achieve speeds in excess of 10 MIPS. At the same time the software industry has introduced multitasking and multi-user operating systems, like UNIX and OS/2, that can support high speed high quality graphics, as well as windowing and graphical user interfaces.

Some may argue that the increasing power of workstations and PCs may decrease the need for computer networks. However, the opposite seems to be true. Ubiquitous and distributed computing power of this magnitude

increases the demand of remote access to shared on-line information and expensive computing resources such as high quality printing and plotting. Furthermore, it enables the implementation of multimedia systems that can support, in an integral manner, voice, data, image, and video applications. The support of all these types of services over large distances is important, since they have the potential of eliminating the tyranny of distances and improving drastically efficiency and productivity. Therefore, the need of Metropolitan Area Networks (MANs) that can provide the large bandwidths required by the above applications and can connect the powerful PCs, workstations and main frames over large distances becomes evident.

The feasibility of MANs in our days is mainly due to three reasons. The first, is the advancement in computer technology. High speed processors can provide very sophisticated communications functions. Similarly, powerful software systems can support distributed processing, communications, and offer software tools to help in the design and management of such networks. The second, is the advancement in fiber optic technology that has increased drastically the channel bandwidth, and has made fiber an economically viable medium in almost any communication environment. The third, is that users and vendors are beginning to understand and expect integrated communication services.

LANs can not be extended directly to high speeds and large distances to become MANs. The reason is inherent inefficiencies in their Medium Access Control (MAC) schemes. For example, the performance of the Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is related to the ratio between packet transmission time and propagation time. The larger the ratio, the better the performance. It is clear that for MANs this ratio is small. On the

other hand, the performance of token ring networks is high when the ratio between the packet transmission time and token rotation time is large. Obviously the ratio is small in the case of a MAN, and the system efficiency is low specially if we consider the case where just only one station transmits.

It has been shown [1] that the theoretical maximum utilization (U) of a medium using an IEEE 802 MAC scheme satisfies the following inequality :

$$U \leq \frac{1}{1+b}$$

$$\text{Where: } b = \frac{\text{propagation time}}{\text{transmission time}} = \frac{R * D}{L * V}$$

Where D = length of the medium, L = packet size, R = data rate, and V = propagation speed. As an example, consider a LAN with the following values for the system parameters: $D = 1$ km, $L = 1024$ bits, $R = 10$ Mbps and $V = 230,000$ km/sec (this is the propagation delay for coaxial cable = 0.77 the speed of light). These values would yield an $b = 0.0849$ and a maximum utilization of 92.2 percent. Consider now a MAN with $D = 20$ km, $L = 1024$ bits, $R = 1$ Gbps, and $V = 200,000$ km/sec (propagation speed for optical fiber). These values would yield a value of 97.6 and a maximum utilization of only 1.0 percent.

The failure of LAN MAC schemes in high bandwidth networks motivated the introduction of new control algorithms for fast networks: like Express-Net, Fasnet, FDDI and C-net. The operation of these network however, still follows a cyclic order in which the various nodes are allowed to transmit one after the other. In order to initialize the cycle, an inter cycle gap must be introduced. This gap becomes larger as the network size and the

bandwidth increase. As a result, the throughput of the network significantly deteriorates. The low throughput of these algorithms and the fact that destination release and concurrent transmission, over distinct segments of the network, can significantly increase the effective throughput, have inspired the introduction of other networks such as Metaring and Distributed Queue Dual Bus (DQDB).

A challenging problem in high speed networks is the efficient and fair channel bandwidth allocation among the competing nodes; which is due to the small packet transmission time relative to the feedback time. Recently, the Global and Local Fairness Mechanisms (GFM and LFM, respectively) have been introduced for fair bandwidth allocation in high speed dual ring networks with destination release. GFM regulates the access to the network by considering it as a single communication resource. Therefore, it cannot fully utilize the throughput advantages offered by destination release, especially under non-uniform traffic conditions and when the number of active nodes on the networks is relatively small.

LFM was introduced to solve the low throughput utilization of the GFM. It considers the network as a distributed collection of resources and not as a single resource. According to LFM, if there are m independent subsets of nodes that communicate only among themselves, then the network will be divided into m distinct segments. In each segment a fairness algorithm similar to GFM will be used to regulate the transmission of the interfering nodes.

We will show later that LFM has the following drawbacks: a) it wastes bandwidth when heavily and lightly loaded nodes are competing for the

same network link(s), b) it exhibits unfair behavior, i.e. the location of a node on the ring has a strong effect on the bandwidth they can acquire, c) its operation is sensitive to network parameters such as size and bandwidth. We also mention that in some cases we have derived analytic equations that can provide the throughput of each node as well as the aggregate total throughput of the network.

The limitations of GFM and LFM have motivated us to introduce the Dynamic Medium Access Control (DMAC) mechanism for dual ring networks with destination release. The operation of DMAC borrows ideas from the operation of recently proposed MAC mechanisms for dual bus architectures. The motivation is that in many cases a dual ring can be considered as a collection of dual bus networks with each dual bus containing a set of interfering nodes for the channel. Therefore, bandwidth allocation techniques similar to the ones introduced for dual bus architectures can be employed for dual ring networks. The proposed DMAC introduces fairness in a very efficient way and provides high throughput. Its operation does not involve any feedback signal and therefore it is insensitive to the various network parameters.

The organization of the rest of the thesis is as follows. In Chapter 2 we describe the Metaring architecture and we investigate its performance. In Chapter 3 we briefly describe the GFM and LFM fairness mechanisms and discuss their advantages and disadvantages. In Chapter 4 we provide a brief description of the various bandwidth allocation mechanisms which have been recently proposed for dual bus networks and we introduce the DMAC algorithm. In Chapter 5 we investigate the performance of DMAC and we

compare it with the corresponding performance of GFM, LFM. Finally in Chapter 6 we present the conclusions.

CHAPTER 2

METARING NETWORK

The introduction of large network size and high bandwidth, decreases drastically the ratio of the packet transmission time to the end-to-end propagation delay. This small ratio allows the network to accommodate multiple packets simultaneously and makes the destination release and concurrent transmission more attractive. The Metaring architecture was introduced to increase the throughput of a ring-based Local Area Network beyond its single link capacity, by destination release and concurrent transmission over distinct segments of the ring.

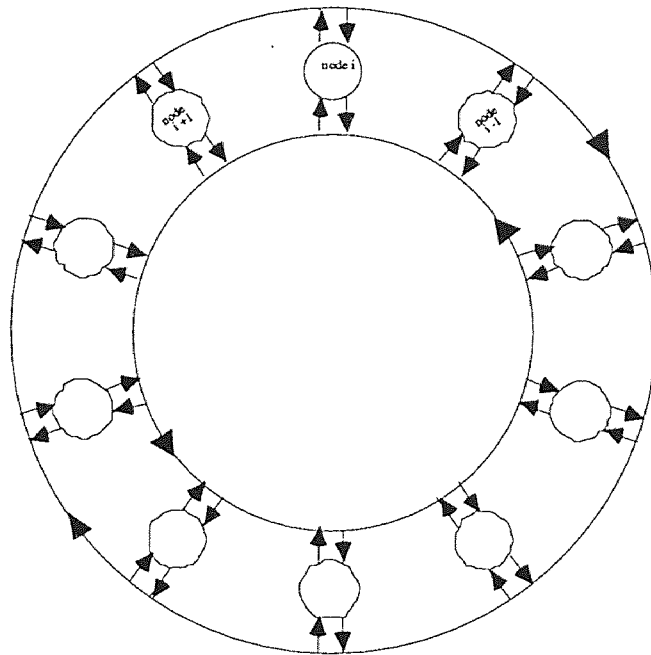


Figure 1: Dual ring architecture.

Figure 1 shows the Metaring architecture. It consists of two unidirectional links in which information travels in opposite directions. The

proposed network has two medium access modes: slotted and buffer insertion. In both cases packets can be transmitted in either direction according to the shortest path routing rule which always selects the ring on which the destination is closer. The amount of information transmitted on each bus is controlled by a special signal traveling on the opposite bus. The opposite direction for data and control is necessary because starvation is caused by the upstream traffic. Therefore, control signals should be sent upstream to the source of starvation. Since information is transmitted on both buses, each node will also execute independently two fairness algorithms, one for each direction. The packets are removed by their destinations and the addresses are arranged in an increasing order (e.g., the n nodes on the ring are numbered from 1 to n). Furthermore, control messages are exchanged between neighboring nodes enabling them to perform specific functions.

The Metaring network can operate under two basic access control modes: buffer insertion, for variable size packets, or slotted for fixed size packets. In both modes the packets are removed by their destination. In order for the nodes to be able to determine very quickly whether to remove a packet from the network, a short 8 bits identity is used.

Buffer Insertion Mode

In this mode each node uses an insertion buffer (IB) on the receiving side of each ring that can store one maximum size packet. A node can transmit a packet at any time as long as its insertion buffer is empty. If traffic arrives when the node is in the middle of a packet transmission, then the ring traffic will be stored in the insertion buffer. Then, after the node has

transmitted the packet, it will pause transmission (even if it has more packets waiting in its queue) until the insertion buffer becomes idle again. In this case non preemptive priority is given to the ring traffic.

Slotted Mode

The slotted mode is used in order to reduce the delay caused by the insertion buffer. This is done at the cost of making the packet size constant. The same hardware interface of the insertion buffer mode is used for the slotted mode. A control bit, the busy bit (BB), in the header of each slot is used to describe the status of a slot. When a node writes on a slot it also makes the busy bit equal to 1; indicating in this way to the other nodes that the slot is full and no one else can write on it. When the destination node receives a slot, it removes it from the network; resetting its busy bit to 0. This will indicate to the other nodes that the slot is now empty and that they can write on it. A node can transmit whenever it receives an empty slot and has a packet waiting in its queue.

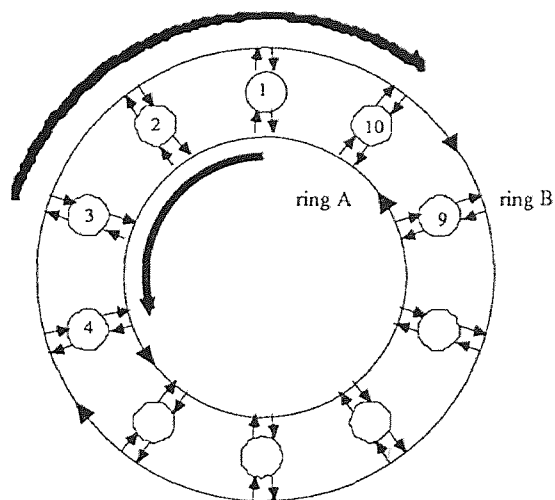
Throughput Performance

Let us now consider a dual network consisted of n overloaded nodes (i.e., they always have packets to send). It is evident that the maximum distance a packet can travel over the ring is $n/2$; since each time a packet is to be transmitted, the node will select the ring for which the destination is closer. If we now assume that we have uniform traffic, then the average distance a packet must travel is $n/4$. Therefore, by using destination release and allowing concurrent transmission we will have, on the average, four stations transmitting simultaneously. Consequently, the overall bandwidth

provided by each link of the dual slotted or buffer insertion ring will be 4 times more than the one of a single token ring.

The previous discussion has shown that destination release and concurrent transmission (combined with dual ring architecture) can drastically improve the performance of the system. However, unrestricted transmission by the nodes and destination release may cause several problems which we discuss below:

- **Starvation:** unrestricted transmission may cause starvation, because upstream nodes have a non preemptive priority over down stream nodes. Starvation can happen if some nodes are in the path of the transmission of an upstream heavily loaded node which will prevent them from accessing the channel for a quite long period of time. Figure 2, Provides an example of starvation. Node 1 transmits continuously to node 4 on ring A and node 3 transmits continuously to node 10 on ring B. As a result, 2 will not be able to transmit any packet on either ring; it is a starved node.



Node 1 transmits continuously to node 4 using the inner ring.
 Node 3 transmits continuously to node 10 using the outer ring.
 Node 2 can not transmit on both rings, and it is starved.

Figure 2: Node Starvation.

- **Priority:** the distributed nature of the access and the destination release, complicate significantly the implementation of a priority access scheme and the integration of synchronous and asynchronous traffic.
- **Fairness:** It becomes extremely difficult to distribute the bandwidth in a fair way between the nodes of each class. This is due to non-preemptive priority inherent to upstream nodes.

The aforementioned problems clearly indicate that new MAC mechanisms must be introduced. Such mechanisms must be able to take advantage of the destination release and concurrent transmission, in order to achieve high throughput. At the same time must prevent starvation and provide fair bandwidth distribution among the nodes serving the same priority class.

CHAPTER 3

METARING FAIRNESS MECHANISMS

There are mainly two Medium Access Mechanisms that have been proposed for the Metaring Architecture: the Global and Local Fairness Mechanisms (GFM and LFM, respectively). The main goal of these mechanisms is to achieve high throughput and provide fairness among competing nodes. In this Chapter we describe the GFM and LFM mechanisms and present their advantages. Then we show where they fail to achieve their objectives and provide reasons for their limitations. We finally derive analytic equations which can estimate the throughput of each node under certain conditions.

3.1 Global Fairness Mechanism (GFM)

GFM views the whole ring as a single resource and gives all nodes equal transmission opportunities. The access on each direction of the ring is regulated by a single control message, the SAT, which circulates in the opposite direction (of the data traffic whose transmission is regulating). Each node can transmit a maximum of K packets per transmission cycle; transmission cycle is the time interval between two successive arrivals of the SAT at the same node. A node is SATisfied if its queue is empty or it has transmitted at least L packets since the last time it observed the SAT signal. Otherwise, the node is said to be starved or unSATisfied and will hold the SAT until it becomes satisfied.

The basic ring access mechanism for one direction is described below. Note that GFM mechanism is the same for the slotted and buffer insertion access modes. For more details the interesting reader is referred to [2, 3].

The GFM consists mainly of two algorithms: Forward SAT, and Send packet. Forward SAT is used to determine what actions the node has to take when it receives the SAT. Send Packet is used to determine what actions a node should take when it sees a slot on the forward channel. We provide below a description of both algorithms.

Forward SAT Algorithm: As we have already mentioned before when a satisfied node sees the SAT signal it will forward it immediately to the upstream nodes. Otherwise, i.e. the node is unsatisfied, it will hold the SAT until it becomes satisfied. only then it will forward the SAT upstream. After a node forwards a SAT, it can send K more packets, where $K \geq L$ (a simple case $K=L= 1$).

Send Packet Algorithm: When a node sees a slot it will check the busy bit of that slot to find if an upstream node has written on it. If the slot is busy and the node is the destination for that slot, it will release it. If the slot is empty the node will transmit a packet if and only if the number of packets that it has transmitted since the last SAT arrival is less than K .

GFM has the following two advantages:

It is Fair: It guarantees, given K and L , that after each rotation of the SAT signal the subset of nodes with L packets in their output buffer will have transmitted at least L packets and at most K packets. All other nodes with less than L packets will have transmitted all of them. The GFM can also be

implemented in an asymmetric manner, that is, the various nodes can use different values for K and L , i.e. K_i and L_i for node "i". In this way nodes with higher traffic requirements (e.g., file servers, bridges) can acquire more bandwidth.

It is deadlock free: When a node is unsatisfied it will hold the SAT until it becomes satisfied. The upstream nodes will transmit their K packets and then will stop until they see the SAT signal again. Therefore the upstream nodes of node i will eventually become idle, and node i will transmit L packets and forward the SAT. Therefore, the SAT signal can not be held indefinitely by a node and GFM is deadlock free.

The GFM has two basic drawbacks. First, it is global. That is, it views the whole network as a single resource and regulates the access to the network according to that. In other words, every node sees the same transmission constraints even if it does not interfere with any other node. In Figure 3 for example, there are three independent subsets of users that communicate only among themselves. The global fairness algorithm will force all groups to maintain fairness among themselves even if they do not interfere at all.

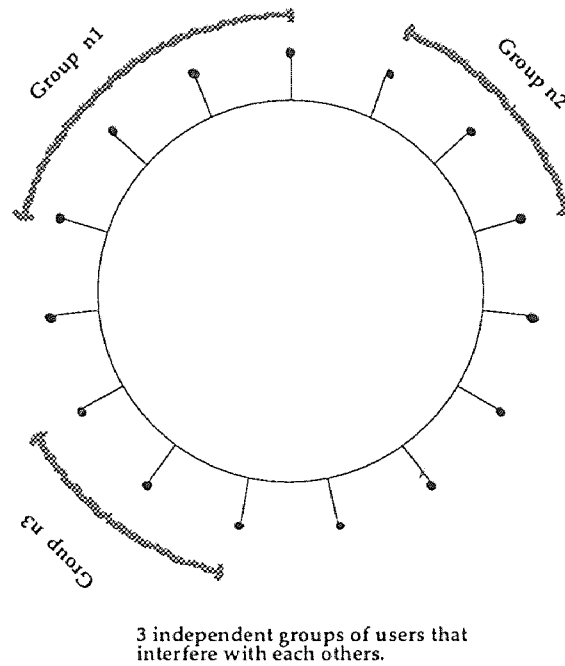


Figure 3: Three independent subsets of users that communicate only among themselves.

Second, the mechanism is continuous. That is, it operates even when no node starves which may result in unnecessary performance degradation. For example, consider the case where there is only one active node. The node will transmit a packet (if $K = L = 1$) or K packets (if $K > 1$) and will stop until it receives the SAT. It is clear that as the network size and bandwidth increase, the ratio between the packet transmission time and the SAT rotation time will decrease and the throughput performance of the system will be unacceptable. The above argument is also true for any case where the number of active nodes in the network is relatively small. In the previous example the similarity between GFM and the token ring operation is obvious. It is therefore evident why GFM is very sensitive to network parameters and inappropriate for high speed MANs

3.2 Local Fairness Mechanism

The disadvantages of GFM motivated the introduction of the Local Fairness Mechanism (LFM) in [4, 5] which is initiated only when starvation is detected. In addition, it involves only the segments of interfering nodes.

LFM divides the network into a collection of communication resources according to the load distribution. Each resource is a subset of links. A fairness algorithm, similar to GFM, is triggered in those subsets where starvation occurs and regulates the medium access between the interfering nodes. At the same time other nodes in the non-congested parts of the network can have free medium access. According to this mechanism, a fairness algorithm can be triggered locally only when the potential of starvation exists. Since each resource in the network can be considered as a dual bus, LFM can be implemented on both dual bus and dual ring topologies with only small modifications. A full algorithm for dual bus networks is given in [5]. A description of LFM for dual ring is given below.

The LFM algorithm that is executed in each node alternate between two modes of operations: non-restricted mode and restricted mode. In the non-restricted mode, a node can transmit any time it sees an empty slot (slotted mode) or its insertion buffer is idle (insertion buffer mode). In the restricted mode, a node can transmit only a predefined quota of packets before it returns back to the non-restricted mode.

Normally, each node is operating in the non-restricted mode. When a node become starved, it activates a control mechanism that switches the operation of the node as well as its upstream nodes, that cause the starvation, into the restricted mode. When all the nodes involved in an access conflict

are satisfied, the restricted mode of operation is terminated. According to LFM each node uses two types of control signals:

1. Request (REQ): this signal initiates the period of restricted mode of operation and is forwarded upstream over the congested segment of the ring.
2. Grant (GNT): this signal is used when the node is satisfied; in order to terminate the local fairness cycle.

The basic operation of the Local Fairness algorithm for slotted mode is demonstrated in Figure 4.

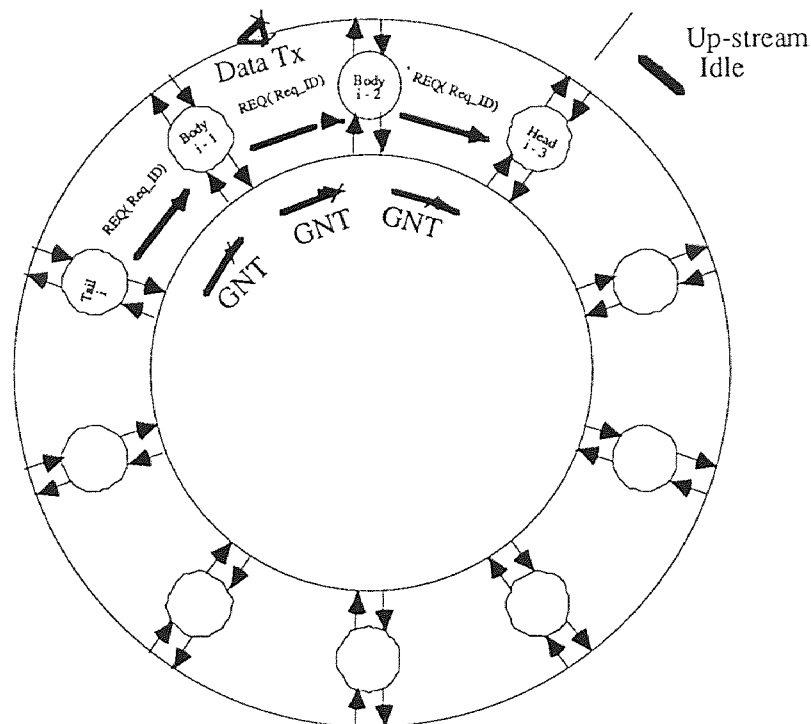


Figure 4: The basic LFM operation for slotted mode

In this figure, we assume that node "i" initiates the algorithm and that there is at least one node upstream to "i" which does not see upstream traffic (

node "i-3"). Node "i" triggers the fairness operation by sending a REQ signal upstream. It changes its state to tail node (T state) and switches to the restricted mode of operation. When the upstream nodes receive the REQ signal they will forward it upstream whenever they see a busy slot on the forward channel; after forwarding the REQ signal a node will also switch to the restricted mode of operation and change its state to body node (B state). When the REQ signal reaches node "i-3" which does not see upstream traffic, this node will switch to the restricted mode of operation, change its state to a head node (H state) and block the request. In this way a REQUEST PATH with a unique and distinct head and tail nodes is created.

When the tail node "i" becomes satisfied (i.e., it has transmitted its predefined quota), it will send a GNT signal upstream, switch back to non-restricted mode of operation, and change its state to that of Free Access (F.A. state). The upstream nodes which will receive the GNT signal will follow a similar procedure. That is, if they are in B state, they will switch to T state and forward GNT upon satisfaction. When the node in the H state (node "i-3") receives the GNT, it will switch to the F.A. state (i.e., non restricted mode) and the local cycle will be terminated.

The symmetry of the ring will cause a deadlock problem when the network traffic is high; since all nodes will see busy upstream traffic and the REQUEST PATH will cover the entire ring. In this case, all the nodes will be in the B state. Therefore, the nodes will eventually transmit their quotas and no node will be allowed to transmit any packets. The reason is that in order for the nodes to transmit more packets they have to renew their quotas. This will not happen unless they switch to the non-restricted mode. Since only the node in the T state can send a GNT signal upstream no such node exists,

because all nodes are in the B state, and the ring will be in a deadlock because there is no tail node present.

In order to solve the deadlock problem, the request control signal must have a REQ_ID as a parameter and the request must be send in the following format: REQ(REQ_ID). Each node must maintain a REQ_ID variable. This variable will identify the original tail node of the REQUEST PATH. By using the REQ_ID, two REQUEST PATHs can be merged when they overlap as follows:

- a. when a node receives REQ(j) and its REQ_ID is $i \neq j$, it will merge the two REQUEST PATHs and switch to B state; if it is not already in the B state. Then if $j > i$, it will forward upstream the REQ(j) and change its REQ_ID variable to j. Otherwise (i.e. $j < i$), it will delete the request from the ring.
- b. if the REQ_ID of the node is the same with the one arrived on the REQ signal, then the same REQUEST PATH covers the entire ring and no head is currently present. In this case the node switches to the combined head-tail (HT) state.

According to LFM mechanism each node can be in one of 5 states , Free Access (F.A.), Tail (T), Body (B), Head (H) and the combined Head-Tail (HT). A simple election algorithm is used to ensure that each REQUEST PATH has a single tail and a single head; except when the REQUEST PATH covers the entire ring. In this case we will have one node in the head-tail state (HT) and all the other nodes will be in the body (B) state.

3.2.1 LFM Properties

In this section we provide a brief discussion of the properties of LFM in terms of throughput, fairness and possibility of deadlock.

Throughput: The introduction of fairness usually affects the maximum system throughput. In LFM the fairness algorithm is triggered only when starvation is detected and it is enforced locally (i.e., only among the segment of the network, where the conflict occurs); allowing the other nodes in the non congested segments to transmit freely without any constrain. Hence LFM is expected to have higher throughput than GFM where the fairness algorithm is applies to all nodes continuously.

However, this algorithm wastes bandwidth in the case where the traffic of heavily and lightly loaded nodes interfere. This bandwidth loss depends on both the distances between the nodes and the offered load. The larger the distances between the nodes, the larger the band width wastage. This case is presented in Figure 5.

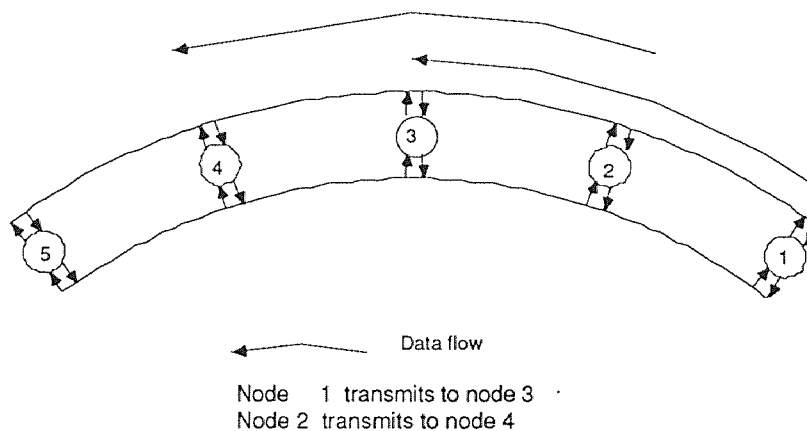


Figure 5: Heavily and lightly loaded nodes in the same REQUEST PATH

In the above figure, there is no traffic upstream to node 1. Node 1 is heavily loaded and transmits continuously to node 3 and node 2 is lightly loaded and transmits to node 4. Since node 1 is upstream to node 2 and is continuously transmitting, node 2 will starve. Therefore it will send a REQ to node 1. At the same time it will switch to T state and enter the restricted mode of operation. When node 1 receives the REQ signal, it will change to the H state and switch to the restricted mode. A REQUEST PATH is now complete. Let us assume that the quota for both nodes are the same and equal to q and the number of slots between node 1 and node 2 is n . Then the node's transmission during each local fairness cycle will be as follows:

Node 1 will transmit q frames and after that it will allow empty slots to pass by. When the empty slots arrive at node 2, it will transmit q frames (or less depending on how many packets are waiting in its queue), switch to non-restricted mode, and send a GNT signal upstream. Since node 2 is lightly loaded, it may not have any other frames to transmit. Nevertheless, it will still see $2*n$ empty slots that node 1 has allowed to pass which will be wasted. The reason for these empty slots is that when node 2 sends the GNT upstream, all the n slots on the forward channel between node 1 and 2 will be empty. It will also take another n slots for the GNT to arrive at node 1 and enable node 1 to transmit again. It is clear from the above example that as the distance between nodes 1 and 2 becomes large, the number of slot between them will increase and more bandwidth will be wasted.

Fairness: In the case where the network size is relatively small and the distances between neighbor active nodes in the REQUEST PATH are similar, local fairness is achieved by limiting the transmission of each node in the restricted mode to a predefined quota of frames or bytes. However, if the

network size is large and the distances between the neighbor active nodes are not similar, then same quota for all nodes does not necessarily lead to similar throughputs. The reason is that during the transition from the restricted mode to non-restricted mode some of the nodes may have an advantage over others and transmit more frames; even though they have the same quota. An example of such a case is shown in Figure 6. In this figure we have three active nodes 1, 2, and 3. The nodes are heavily loaded and their traffic interferes. The distances between them are not similar. Node 1 will have an advantage over nodes 2 and 3 and will acquire more bandwidth than any of them.

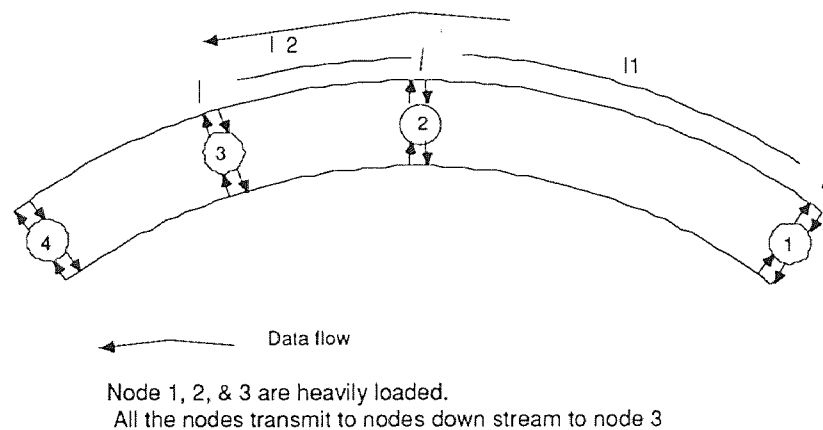


Figure 6: Distances between the interfering nodes are not similar

Deadlock free: The presence of one head and one tail for each REQUEST PATH (except for the case where we have a single head-tail node), ensures that one GNT signal will be sent and the network will never into a deadlock.

We also mention that when the network is fully loaded (i.e., nodes have always something to transmit), a single REQUEST PATH will cover the entire

ring. In this case all the nodes will transmit their quotas. When a node switches to the non-restricted mode and renew its quota, it will almost immediately switch back to the restricted mode; this is because it will be always covered by heavily loaded up-stream nodes. In this case each node will be in the restricted mode most of the time and the LFM will operate similar to GFM.

3.2.2 Throughput Analysis

In the case of slot reuse the derivation of analytic estimates for the throughputs of the various nodes under every load condition is a very difficult problem. However, our simulation results have shown that for certain cases of loading such derivation is possible. Consider for instance Figure 6 where nodes 1,2 and 3 are heavily loaded and all of them transmit to nodes below node 3. The number of slots between 1 and 2 is l_1 , and between 2 and 3 is l_2 (where $l_1 > l_2$). The quotas for 1,2 and 3 are q_1 , q_2 , and q_3 , respectively. We have found that if the condition $2l_2 + q_2 + q_3 > 2l_1$ is satisfied, then the number of times the three nodes switch to restricted mode is the same. That is any time a REQUEST PATH is established, it will include the 3 nodes which will move to the non-restricted mode before a new REQUEST PATH is established. If the above condition is satisfied, then the throughput of each node is given by the following equation:

$$T_i = \frac{n_i}{\sum_{i=1}^3 n_i}$$

Where T_i is the throughput of node i , and n_i is the number of slots transmitted by node i between any successive REQUEST PATH

establishments. In this case n_i will be equal to q_i , plus the number of slots between node i and the nearest neighbor active node which is part of the same REQUEST PATH. Therefore we can write:

$$n_1 = q_1 + 2h$$

$$n_2 = q_1 + 2L$$

$$n_3 = q_1 + 2L$$

It is clear from the above example that the throughput of each node will depend on the distances between the nodes. This dependency will make the LFM exhibit an unfair behavior and enable some nodes to have higher throughputs. In the above example if the nodes have the same quota, node 1 will have higher throughput than node 2 or node 3.

CHAPTER 4

A NEW DYNAMIC MAC MECHANISM

The performance analysis of GFA and LFA shows clearly their sensitivity to network parameters, especially under non-uniform load distribution. In this chapter we introduce a new access mechanism, the Dynamic MAC mechanism (DMAC), that tries to solve GFA and LFA problems. The operation of DMAC borrows ideas from a variety of access mechanisms that have been recently proposed for high speed MANs. For this reason, in the sequel, we first provide a brief description of the most prominent MAC mechanisms that have been proposed for dual bus architectures. Then we introduce DMAC and discuss its properties and advantages.

4.1 Dual Bus Mechanisms for MANs

Recently the Distributed Queue Dual Bus (DQDB) was introduced [6, 7] for dual bus high speed MANs; DQDB has been adopted by the IEEE as the 802.6 standards for MANs. However, the basic DQDB MAC mechanism exhibits unfair behavior. That is, the location of a node in the network drastically affects the amount of channel bandwidth it can receive. A modification of this algorithm, called the Bandwidth Balancing (BWB) mechanism was proposed in [8] to deal with the above unfairness problem; we will refer to this mechanism as BWB_DQDB. BWB-DQDB can provide the lightly loaded nodes with the requested bandwidth, and evenly distribute the remaining bandwidth among the heavily loaded nodes. However its operation requires some bandwidth wastage and its converges to the steady state, where fair bandwidth allocation is achieved, is rather slow.

For these reasons the No Slot Waste Bandwidth Balancing (NSW_BWB) mechanism was introduced [9, 10, 11]. NSW_BWB divides the channel bandwidth in a fair and efficient way among the competing nodes but without wasting any channel slots. Furthermore, it can converge faster to the steady state (than BWB_DQDB) and is insensitive to the network parameters. The performance of NSW_BWB mechanism under the presence of erasure nodes has been also investigated in [12]. The erasure nodes have been introduced to allow slot reuse. That is, when a slot that has been read by its destination passes in front of an erasure node, it is released and another node can write it. As a result the aggregate system throughput increases. In the sequel, we provide a brief description of these mechanisms. Such discussion will facilitate understanding the operation of DMAC.

DQDB MAC Mechanism

In DQDB the network nodes are connected to two unidirectional buses A and B; as it shown in Figure 7.

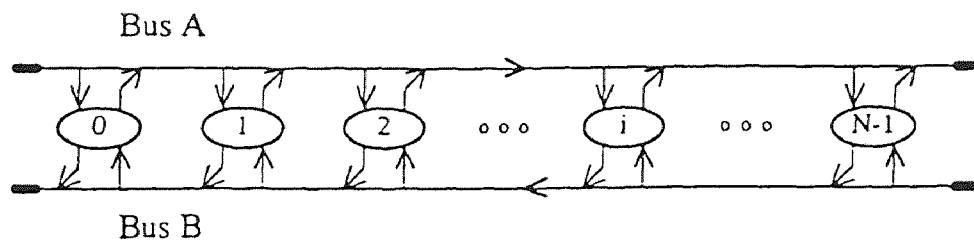


Figure 7: DQDB dual bus architecture.

The information on the buses travels in opposite directions. Node 0 on bus A and node N-1 on bus B are responsible for generating the slots. When a node wants to transmit to other nodes located to its right, it will use bus A. Otherwise it will use bus B. The operation of both buses is the same.

Therefore, we will describe only the operation on bus A. We will use the term forward bus for bus A, and reverse bus for bus B. In DQDB the header of each slot has a Busy Bit (BB), which indicates whether the slot on the forward bus is full (i.e., an upstream node has used the slot) and a Request Bit (RB) which indicates whether a downstream node on the reverse bus has requested a reservation for a slot.

Each node has a Request Counter (RQ_CTR) and a Count Down Counter (CD_CTR). Their operation is as follows. When the node is idle, it increments RQ_CTR by one for every RB = 1 seen on the reverse bus, and decrements RQ_CTR by one for every empty slot seen on the forward bus. In this way RQ_CTR keeps track of the number of downstream nodes that have made slot's reservation. When a packet arrives at a node, the node sends a request upstream (on the reverse bus), transfers the content of RQ_CTR to CD_CTR, and resets RQ_CTR to 0. After this instant, CD_CTR is decremented by one for every empty slot seen on the forward bus, and RQ_CTR is incremented by one for every RB = 1 seen on the reverse bus. When CD_CTR becomes zero, the node transmits its packet in the first empty slot on the forward bus. We see that CD_CTR determines the number of empty slots that a node must allow to pass, due to reservations by downstream nodes, before it is allowed to transmit its packet.

BWB DQDB Mechanism

According to this mechanism every time a node transmits M packets increases the value of RQ_CTR by one; allowing in this way an empty slot to go downstream and be written by the first active downstream node with CD_CTR = 0. It is evident that if all the downstream nodes are idle, this free

slot will be wasted. The amount of channel bandwidth loss will depend on the value of M . The smaller the value of M , the higher the bandwidth wastage. However, as the value of M increases, the system takes more time to reach the steady state where the fair bandwidth allocation is achieved. Therefore, there is a tradeoff between channel utilization and convergence speed.

NSW_BWB Mechanism

NSW_BWB mechanism informs a node in advance about the future of the free slot that may allow to pass; i.e. whether another node will use it. Therefore, the node can let a free slot to pass only if a downstream node is going to use it. In this way no slot is wasted. Consequently, the nodes can use a small value of M and decrease significantly the required time to reach steady state.

The NSW_BWB mechanism uses an additional control bit in the slot header called the Transmit Additional Request (TAR) bit. Whenever a node i transmits its M th packet, it makes the TAR bit = 1 in the written slot. The first active downstream node j that has an available packet, for which a request has not yet been sent, will make the TAR bit equal to zero and send an additional request upstream. This additional request will be seen by node i , which will increment its RQ_CTR by one; allowing a free slot to go downstream. It is clear that an extra request will be sent upstream only if a downstream node has an available packet. This will insure that the idle slot that node i will allow to pass will be written and no slots will be wasted.

We point out that the extra request that node j will send, will be also seen by all the nodes upstream to node i , which will increment their

RQ_CTRs by one. These nodes will be compensated by not allowing node i to send a request for the next packet waiting in its queue.

Bandwidth Balancing Mechanism Under the Presence of Erasure Nodes

In [12] the performance of NSW_BWB and BWB_DQDB is investigated in the presence of erasure nodes. The erasure nodes are special nodes that will release any slot that has already been read by its destination. The released slots can then be used by other nodes, and this will significantly increase the throughput of the system. We will refer to NSW_BWB under the presence of erasure nodes as NSW_BWB_EN. According to this mechanism each erasure node has an Erased Slot Counter (ES_CTR) and a Request Counter (RQ_CTR). The operation of the RQ_CTR is similar to that of regular nodes. ES_CTR on the other hand, increases by one whenever a slot is erased and RQ_CTR is greater than 0; which is an indicator that there are active nodes downstream and one of them may use the erased slot. When the erasure node sees a request on the reverse bus and its ES_CTR is greater than 0, it will reset the request and decrement its ES_CTR by one.

4.2 The Dynamic MAC Mechanism for Dual Ring MANs

The objective of the new mechanism is to achieve fairness and high throughput regardless of system parameters such as number of nodes, network size, channel capacity, and load distribution. In order to achieve high throughput, the nodes must transmit continuously unless there is an active downstream node that may be affected by this transmission. In this case the downstream node must inform the upstream nodes about its bandwidth requirements by sending reservations. Since the operation of both rings is the

same, in the next section we will give a full description of the operation on ring A. We will use the term forward channel for ring A and reverse channel for ring B.

4.2.1 DMAC Implementation

According to DMAC mechanism, the header of each slot will have a Request Bit (RB), Busy Bit (BB) and Transmit Additional Request (TAR) bit. The RB will indicate whether a slot on the reverse channel is carrying a reservation from a downstream node. The BB will indicate whether the slot on the forward channel is empty. Finally, the Transmit Additional Request (TAR) bit will allow a downstream node to send an extra request when it has packets waiting in its queue. Each node sends two types of requests to the upstream nodes: a) Regular requests, that is, requests that a node sends when a packet becomes first in queue. b) Extra requests, that is, requests that a node sends when it erases a TAR bit.

The problem with ring networks is that there is no physical termination of the channel and for this reason an inserted request may circulate forever and increase continuously the values of RQ_CTR at the various nodes. It is evident that a mechanism is needed for the removal of requests. A solution to this problem is to use a REQ_ID instead of a Request Bit in the header of each slot. When a node wants to send a request on the reverse channel, it actually inserts its own ID in the REQ_ID field. Therefore, the source nodes will be able to remove their own requests from the ring. In the sequel we describe the various components inside each node that control the transmission on ring A. As shown in Figure 8, each node has the following:

Upstream Request Counter (URQ_CTR): Counts the number of requests (regular and extra) that this node has to send to upstream nodes.

Request Counter (RQ_CTR): Keeps track of the number of free slots this node must allow to pass, due to the reservations made by downstream nodes, before it can transmit a packet .

Queue: Contains the packets that have arrived at the node for transmission on channel A.

Transmit Register: Holds the first packet in the Queue. When this first packet starts its transmission, the bits of the next packet in the Queue start entering the Transmit Register and the next available packet becomes first in the Queue. Therefore, it can be transmitted on the immediate next slot.

Bandwidth Balancing Counter (BWB_CTR): It is used to determine when the node will transmit a TAR=1 bit on the forward channel. Its operation is as follows. BWB_CTR is incremented by one for every packet transmitted by the particular node. Whenever $BWB_CTR = M$ the node sends a TAR bit downstream, resets BWB_CTR to 0, and sets a flag, the TAR_FLAG, to 1. When a packet becomes first in queue and the TAR_FLAG = 1, the node will not send a request upstream, compensating in this way the upstream nodes for the additional request they saw which was inserted by a downstream node will send when it erased a TAR bit.

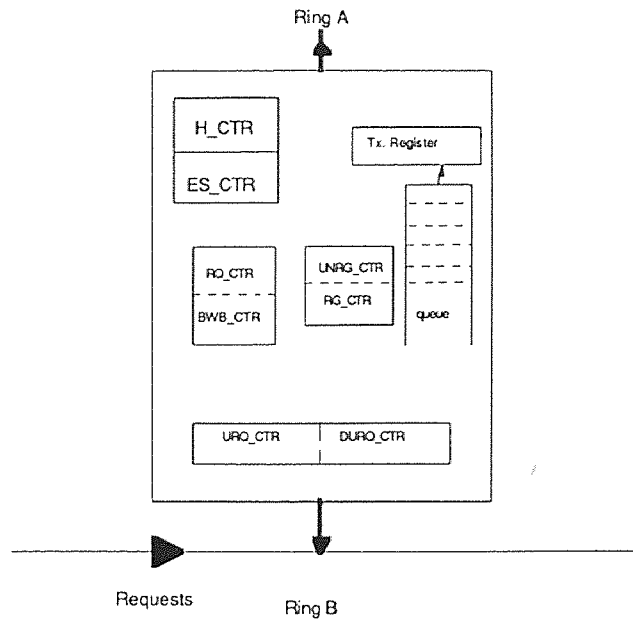


Figure 8: DMAC node components that control the transmission on Ring A .

Unregister Counter (UNRG_CTR): Counts the number of unregistered packets in the node's Queue. Unregistered are packets for which requests have not yet been sent upstream and whose presence may enable a node to erase a TAR=1 bit and send an extra request. In order for a node to do that its UNRG_CTR must be equal to greater than M. For more information on UNRG_CTR we refer to [9].

Register Counter (RG_CTR): counts the number of registered packets in the node's Queue. Registered are packets for which a request has been sent or packets whose presence has been already used for erasing a TAR= 1 bit. Packets can become registered in the following two ways: a) when a packet becomes first in Queue and a regular request for it is sent upstream, b) when UNRG_CTR is greater than or equal to M and a TAR = 1 bit is seen on the

forward channel. In this case the node will reset the TAR bit to 0, increase RG_CTR by M, and decrease UNRG_CTR by M; we see that in this case a set of M packets becomes registered. For more details on the operation of RG_CTR we refer to [9]. The objective of RG_CTR and UNRG_CTR is to ensure that the node will transmit the same number of TAR = 1 bits that erases. It is shown in [9] that this condition will guarantee fair bandwidth allocation among the various nodes.

Delay Upstream Request Counter (DURQ_CTR): counts the number of requests that a node has removed from the reverse channel. These requests will be forwarded to the upstream nodes whenever the node sees a busy slot on the forward channel, or erased (if the node upstream traffic is idle). The objective of DURQ_CTR is to increase the system throughput by reducing the number of requests on the reverse channel.

Erased Slot Counter (ES_CTR): This counter determines the number of requests that can be erased by the node. In the sequel we describe its operation on the forward and reverse channels.

Operation on Forward Channel: When a node releases a slot it increments its ES_CTR by one only in two cases: a) if there is an active downstream node that will use the slot, i.e. RQ_CTR > 0, b) if there is no active downstream node present, but the node itself can use the slot to transmit a packet for which it has not sent a request, i.e. RQ_CTR = 0 and URQ_CTR > 0.

Operation on Reverse Channel: When a node sees a RB = 1 on the reverse channel and its ES_CTR > 0, it resets RB to 0 and decrements ES_CTR by one. On the other hand if the RB = 0 and both ES_CTR and URQ_CTR are greater than zero (this is the case where the node uses the slot that released to

transmit a packet for which a reservation has not been made), the node will decrement both URQ_CTR and ES_CTR by one.

We point out that the motivation for erasing requests, in addition to erasing slots, is higher system throughput. Otherwise upstream nodes seeing the same number of requests (as with no erasure nodes) will allow free slots to go downstream which may be wasted, since downstream nodes may have already sent their packets in the released slots, and the improvement in the throughput will be minor. We also mention that it is not a good idea to simply erase a request for every slot released. Consider for instant the case where there is only one active node, say node i . Node i is heavily loaded and transmits to node j . If node j increments its ES_CTR for every slot it releases its ES_CTR will increase to a large value. If later a node downstream to node j becomes active and sends a request bit upstream, this request will be erased by node j (since its ES_CTR > 0) and the downstream node may not receive any bandwidth.

Head Counter (H_CTR): This counter is used to assist the node to decide on whether it will be a head node. Its operation is as follows. The node initializes H_CTR with H (the value of H used depends on the network size). When it sees an empty slot on the forward channel it decrements H_CTR by one. Otherwise (i.e., the slot is busy), it sets H_CTR to H . The node is considered a head when $H_CTR = 0$. The head node does not see upstream traffic. Therefore it will not send any request upstream for its own packets and will erase all requests seen on the reverse channel. The motivation behind the H_CTR is to increase the system throughput by allowing the head node to block its own requests and any request it sees on the reverse bus. Since the upstream to head nodes have no effect on its transmission, or to the other

downstream nodes, any request passing upstream to head will only prevent the upstream nodes from transmitting their packets.

The main idea of DMAC is to split the network into a number of independent ring packets. Each ring packet contains the nodes that interfere with each other. A dynamic fairness algorithm can then be applied to each packet. This fairness algorithm will allow each node to transmit continuously unless it starts affecting a downstream node. In this case the downstream node will inform the upstream nodes about its traffic requirements by sending requests on the reverse channel and forcing them to allow free slots to pass by. The DMAC mechanism will provide the lightly loaded nodes with the requested bandwidth and evenly divide the remaining bandwidth among the heavily loaded nodes. It will achieve high throughput by allowing destination release and will be insensitive to the network parameters; since the interaction between the nodes does not require any feedback control messages.

4.2.2 The DMAC Operation

In this section we describe the operation of DMAC by looking at the various events that may occur, and describing the corresponding behavior of the node. We assume that the nodes will first look at the reverse channel and then at the forward bus.

- a- **Segment arrival(s):** UNRG_CTR will increase by one; if a long message arrives UNRG_CTR will increase by the number of packets in the message.
- b- **Segment becomes first in queue:** if TAR flag is equal to 0 and RG_CTR is greater than 0 (packet is registered), the node will send a regular request on

the reverse bus. If $TAR_FLAG = 0$ and $RG_CTR = 0$ (unregistered packet) the node will send a regular request, increment RG_CTR by one and decrement $UNRG_CTR$ by one. If the node is a head node it will not send any request upstream. Finally, if the TAR flag = 1 no action will be taken by the node.

c- A slot arrives at the reverse bus: There are two cases: $RB=0$ and $RB=1$.

When $RB = 0$: if both ES_CTR and URQ_CTR are greater than 0 (the node has used the erased slot to transmit a packet for which it has not sent a request), the node will decrement both of them by 1. If at the same time $DURQ_CTR$ is greater than 0, the node will make $RB = 1$, and decrease $DURQ_CTR$ by 1. However, if $DURQ_CTR$ is equal to 0 and URQ_CTR is still greater than 0 (after it was decreased by one), the node will send a request upstream and will decrement URQ_CTR by 1.

When $RB = 1$: The node will increment RQ_CTR by 1. If ES_CTR is greater than 0 the node will make $RB = 0$, and decrement ES_CTR and RQ_CTR by 1. Then the node will check $DURQ_CTR$. If it is greater than 0 it will decrement $DURQ_CTR$ by 1 and make $RB = 1$. Otherwise (i.e. $DURQ_CTR$ is 0), the node will check the URQ_CTR . If $URQ_CTR > 0$ then it will make $RB = 1$ and will decrement URQ_CTR by 1. If $ES_CTR = 0$ and $DURQ_CTR = 0$ then $RB = 0$ and $DURQ_CTR = 1$. If $ES_CTR = 0$ and $DURQ_CTR > 0$ the node will not take any action.

d- A slot is seen on forward bus: This event can be divided into the following two events:

Slot is erased: The node will increment ES_CTR by 1 if: a) $RQ_CTR > 0$ (an active downstream node will use the released slot). b) $RQ_CTR = 0$ and

$URQ_CTR > 0$ (the node can use the released slot for a packet for which a request has not been sent yet). If TAR bit is equal to 1 the node will erase it.

Slot is empty: If H_CTR is greater than 0 the node will decrement it by 1. If RQ_CTR is greater than 0, the node will allow the slot to pass by. Otherwise (i.e., $RQ_CTR = 0$), the node will transmit a packet; if one is available (i.e., $RG_CTR + UNRG_CTR > 0$).

e- Node Transmits a packet : If TAR_FLAG is equal to 1 and RG_CTR is equal to 0, $UNRG_CTR$ will decrease by 1. Otherwise, the RG_CTR will be decrease by 1. The node will increment the BWB_CTR by 1 and it will reset the TAR_FLAG to 0. If by increasing BWB_CTR its value becomes equal to M , the TAR bit will be set to 1 in the written slot, BWB will be reset to 0, and TAR_FLAG will be set to 1; to indicate, that a request should not be sent upstream for the next packet becomes first in queue. In the following Figure 9 we provide a pseudo code description of the various actions that a node must take.

a- Packet arrival(s):

$UNRG_CTR = UNRG_CTR + \text{number of packets in the message.}$

b- Packet becomes first in queue:

```

If  $TAR\_FLAG = 0$  AND  $RG\_CTR > 0$  then
  If  $H\_CTR > 0$  then
     $URQ\_CTR = URQ\_CTR + 1;$ 
  End if
Else If  $TAR\_FLAG = 0$  AND  $RG\_CTR = 0$  then
   $RG\_CTR = RG\_CTR + 1;$ 
   $UNRG\_CTR = UNRG\_CTR - 1;$ 
  If  $H\_CTR > 0$ 
     $URQ\_CTR = URQ\_CTR + 1;$ 
  End if
End if

```


c- Slots arrived at the reverse bus:

When the RB = 0

```

If ( ES_CTR > 0 ) and ( URQ_CTR > 0 ) Then
    ES_CTR = ES_CTR - 1;
    URQ_CTR = URQ_CTR - 1;
End If
If ( DURQ_CTR > 0 ) Then
    DURQ_CTR = DURQ_CTR - 1;
    RB = 1;
    SL_REQ_ID = DURQ_CTR_ID
Else If ( URQ_CTR > 0 ) Then
    URQ_CTR = URQ_CTR - 1;
    RB = 1;
    SL_REQ_ID = i;
End if
End if

```

When RB = 1

```

RQ_CTR = RQ_CTR + 1;
If ( ES_CTR > 0 ) Then
    If ( URQ_CTR > 0 ) Then
        ES_CTR = ES_CTR - 1;
        RB = 1;
        SL_REQ_ID = i;
    Else If ( DURQ_CTR > 0 ) Then
        RB = 1;
        ES_CTR = ES_CTR - 1;
        DURQ_CTR = DURQ_CTR - 1;
        SL_REQ_ID = DURQ_CTR_ID;
    Else RB = 0;
        ES_CTR = ES_CTR -1;
    End if
End if
Else If ( DURQ_CTR = 0 ) Then
    RB = 1;
    DURQ_CTR = DURQ_CTR + 1;
    DURQ_CTR_ID = SL_REQ_ID
End if
End if

```

d - Slot is seen on forward bus:

If slot is erased:

```

If ( RQ_CTR > 0 ) OR { (RQ_CTR = 0) AND (RG_CTR > 0) } Then
    ES_CTR = ES_CTR + 1;
End if
If TAR bit = 1 Then
    TAR bit = 0;
End if

```

Slot is empty:

```

If ( H_CTR > 0 ) Then
    H_CTR = H_CTR - 1;
End if
If ( RQ_CTR > 0 ) Then
    RQ_CTR = RQ_CTR - 1;
    If (RQ_CTR = 0) AND (DURQ_CTR > 0) AND (H_CTR = 0) Then
        DURQ_CTR = DURQ_CTR - 1;
    End if
Else If ( RG_CTR + UNRG_CTR > 0 ) Then
    call transmit packet event;
End if
End if

```

Slot is busy

```

H_CTR = H;
If ( TAR bit = 1 ) AND (UNRG_CTR >= M) Then
    TAR bit = 0;
    RG_CTR = RG_CTR + M;
    UNRG_CTR = UNRG_CTR - M;
    If (H_CTR > 0) Then
        URQ_CTR = URQ_CTR + 1;
    End if
End if

```

Transmit packet:

```

If (TAR flag = 1) AND (RG_CTR = 0) Then
    UNRG_CTR = UNRG_CTR - 1;
Else
    UNRG_CTR = UNRG_CTR - 1;
End if
If ( TAR flag = 1 ) AND (ES_CTR > 0) Then
    ES_CTR = ES_CTR - 1;
End if
TAR flag = 0;

```

```

BWB_CTR = BWB_CTR + 1;
If (BWB_CTR = M ) Then
    BWB_CTR = 0;
    TAR flag = 1;
    TAR bit = 1;
End if

```

Figure 9: A pseudo code for DMAC algorithm

4.2.3 Advantages of DMAC Mechanism

Fairness: By ensuring that the number of TAR bits that a node erases is equal to the number of TAR bits that it inserts, the lightly loaded nodes will get the requested bandwidth and heavily loaded nodes will evenly share the remaining bandwidth.

High Throughput: In DMAC the network is considered to consist of multiple resources rather than a single resource. The network is divided into segments that contain the competing nodes. A local fairness mechanism is used in each segment independently from the other segments and allows the nodes to transmit continuously, unless a downstream node makes a reservation. Therefore, the DMAC mechanism will have a high throughput. For example, if we have 4 independent groups of nodes, the aggregate throughput of the system will be 4 times the throughput of a single link. Furthermore, the slot reuse inside each segment may increase the throughput even more. Finally, even if the network is heavily loaded and no head is present, slot reuse will allow the network to carry traffic greater than twice the single link capacity.

Deadlock Free: By using the REQ_ID in each slot, we guarantee that if a request is not deleted due to slot release, the request will be removed from the network by the node that inserted it and the values of RQ_CTR will not increase indefinitely.

Insensitivity of DMAC to Network Parameters: The fairness and throughput are not affected by system parameters such as network size, channel capacity, number of nodes, or load distribution. This is because the mechanism is dynamic and it does not involve any feedback signal.

CHAPTER 5

PERFORMANCE ANALYSIS

In this chapter we study the throughput performance of GFM, LFM and DMAC mechanisms. The simulator for each mechanism has been implemented in the C programming language. The number of nodes, the destination distribution, the ring size and the distance between the nodes are all variables. This enables us to study and simulate a wide range of system configurations.

In the cases presented here, we simulate the operation of one ring (the operation and performance of the other ring is similar). The packets generated have a fixed size of 1000 bits, this is the typical packet size used in the Metaring architecture. The slot size is equal to 1 packet size, the ring capacity is 1 Gbps, and the propagation delay is 5 μ sec. In order to analyze and compare the performance of the three MAC mechanisms, different traffic scenarios have been constructed. The simulation results and the corresponding discussion are given below.

Scenario 1: Only one node is active and it is heavily loaded. The network size is 20 km. The number of nodes is 10, symmetrically located around the ring. Then with the previous values for the system parameters, the number of slots between two neighboring nodes will be equal to 10. In this case the simulation provides for both DMAC and LFM a throughput of 1.0. In contrast, the throughput of GFM is only 0.2 (the value used for K is 20). These results are expected, since LFM and DMAC allow nodes to transmit continuously unless their transmission affects downstream nodes. In scenario 1, where there is one only active node, there is no downstream node affected

and for this reason the throughput of the active node will be 1.0. However, in the case of GFM, the node will transmit K packets and then stop and wait for the SAT control signal. Since the SAT rotation time is $100 \mu\text{sec}$ and the time for transmitting 20 packets is $20 \mu\text{sec}$, the active node throughput will be 0.2.

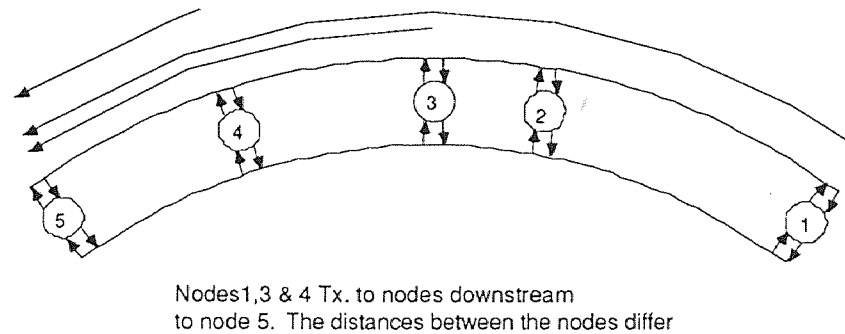


Figure 10: Asymmetric location of the nodes and asymmetric load.

Scenario 2: As shown in Figure 10, nodes 1, 3, and 4 are the only active nodes, and they are heavily loaded. Node 1 transmits to 5, 3 and 4 transmit to 6. The distance between nodes 1 and 3 is 20 slots and between nodes 3 and 4 is 4 slots. The quota in LFM is 24 and the value of k in GFM is also 24. The throughputs of the three active nodes are presented in table 1.

Node ID	1	3	4	total
GFM	0.20	0.20	0.20	0.60
LFM	0.52	0.24	0.24	1.0
DMAC	0.333	0.333	0.333	1.0

Table 1: Throughput performance for traffic scenario 2.

The above results show that in the case of GFM the three active nodes acquire the same bandwidth and the aggregate throughput is only 0.6. LFM has a total throughput of 1.0, but node 1 acquires more than twice as much bandwidth as nodes 3 or 4. The simulation results show that only DMAC has a throughput of 1.0, and at the same time evenly distributes the channel bandwidth among the three nodes. The reason for the unfair behavior of LFM is the different distances between the active nodes. Therefore, the throughput of each active node depends on both the number of slots between these nodes and their quota. Our analysis which was presented in Chapter 3 can also be used here to provide estimates of the nodes' throughputs. Our equations provide the following values of throughput for the three active nodes: 0.518 for node 1, 0.241 for node 2 and 0.241 for node 3. We see that the analytic estimates are in excellent agreement with the simulation results.

Scenario 3: In this scenario we investigate the case where we have both heavily and lightly loaded nodes. As it is also shown in Figure 11, we have two active nodes, 1 and 4. Node 1 is heavily loaded. Node 4 is lightly loaded (offered load = 0.1).

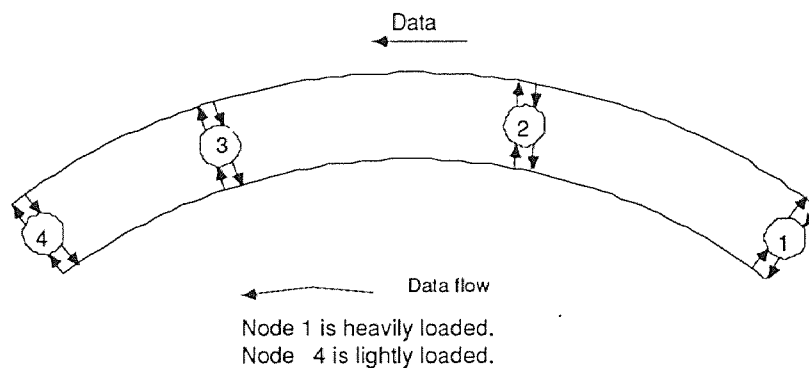


Figure 11 : Traffic of Lightly and heavily loaded nodes interfere.

Table 2 shows that all three mechanisms provide the lightly loaded node with the requested bandwidth but DMAC is the only one that does not waste any bandwidth. This result verifies the conclusions of our previous discussion on the disadvantages of LFM. That is, the heavily loaded node allows empty slots for the lightly loaded node which can not use them and therefore are wasted.

Node ID	1	4	Total
GFM	0.24	0.1	0.34
LFM	0.76	0.1	0.86
DMAC	0.9	0.1	1.0

Table 2: Throughput performance for traffic scenario 3

Scenario 4: In this case we consider a localized pattern. The nodes in this traffic scenario can be divided into 4 groups (see Figure 12). Each group has nodes that communicate only among themselves and do not transmit to nodes in other groups. As it shown in this figure, group 1 contains node 1, group 2 contains nodes 2 and 3, group 3 contains nodes 4,5 and 6, and group 4 contains nodes 7,8,9 and 10.

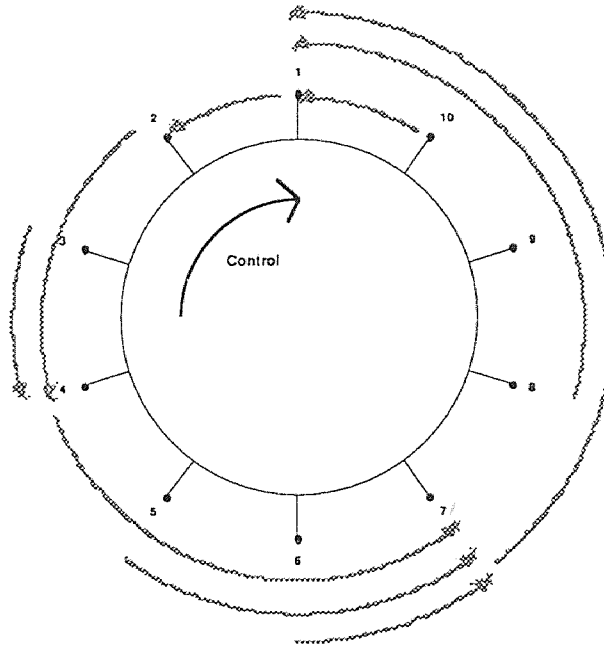


Figure 12: Localized pattern scenario.

Table 3 shows the throughputs of the various nodes and the aggregate throughput of the system for all three mechanisms. In this scenario both DMAC and LFM achieve a very high throughput of 4.0 and at the same time they introduce fairness among the interfering nodes. DMAC considers the ring as 4 independent resources and therefore divides the dual ring to 4 independent segments. Each segment runs a localized fairness algorithm which, as we have explained before, distributes the channel bandwidth in a fair and efficient way. LFM treats these groups independently. A local fairness algorithm is also used here to regulate the transmit simultaneously. The difference between DMAC and LFM is that in the case of lightly loaded nodes inside the groups, LFM may waste bandwidth and its aggregate throughput may be less than 4.0; whereas in the case of DMAC the lightly loaded node will have no effect and the aggregate throughput will remain the same (4.0).

NodeID	1	2	3	4	5	6	7	8	9	10	Total
GFM	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	2.5
LFM	1.0	0.5	0.5	0.333	0.333	0.333	0.25	0.25	0.25	0.25	4.0
DMAC	1.0	0.5	0.5	0.333	0.333	0.333	0.25	0.25	0.25	0.25	4.0

Table 3: Throughput performance for traffic scenario 4

Scenario 5: We investigate the case where all nodes are heavily loaded and we have uniform traffic. The results are shown in Table 4.

NodeID	1	2	3	4	5	6	7	8	9	10	Total
GFM	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35	0.35	3.5
LFM	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	0.32	3.2
DMAC	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	0.22	2.2

Table 4: Throughput performance for traffic scenario 5.

The results above show that all mechanisms provide nodes with the same throughput. We see that GFM has the highest throughput in this case. The reason for the low throughput of DMAC is the busy slots that all nodes will see on the forward channel, which will prevent any node from becoming a head node. Therefore, the number of requests that will go around the ring will increase with end result empty slots to remain unused for larger

intervals of time (since number of nodes that will see each request will increase).

The above case is extremely unlikely to happen in MANs for two reasons. First, MAN traffic tends to be burst. Therefore, one would never expect all nodes to be simultaneously busy for a long period of time. Second, any network attempting to operate at 100 percent of capacity for an extended period of time will quickly grind to halt because of queuing delay.

CHAPTER 6

CONCLUSIONS

In this thesis we have investigated the performance of GFM and LFM, two recently proposed MAC mechanisms, for high speed dual ring networks with destination release. We have shown that the performance of both mechanisms are sensitive to network parameters such as channel capacity, load distribution, and ring size. Their sensitivity is due to the dependency of their operation on feedback control signals, the SAT in the case of GFM and the GNT in the case of LFM. We have shown the similarity of GFM with the Token Ring which clearly demonstrates why GFM is inappropriate for high speed MANs. We have also shown the fairness and bandwidth wastage problems of LFM, and we have derived analytic estimates for its throughput under certain cases of loading.

The limitations of LFM and GFM have motivated us to introduce a new access mechanism, the DMAC, which provides bandwidth fairness by allowing downstream nodes to make direct reservations to upstream nodes without involving feedback signals like SAT, which degrade the system performance. As a result DMAC introduces fairness in a very effective and efficient way, provides high throughput, and it is insensitive to the various system parameters.

Finally, we have investigated and compared the performance of the three mechanisms (GFM, LFM and DMAC) using simulation results. We have looked at different traffic scenarios and we have found that in all cases DMAC and GFM could distribute the bandwidth in a fair way among the competing nodes. In contrast, LFM exhibited an unfair behavior in the case

where the distances between the active nodes were not the same. In terms of throughput performance, we have found that the total throughput of GFM can be extremely low in some cases. In contrast, DMAC can achieve higher throughput than GFM or LFM, except in the case where all nodes are heavily loaded under a uniform traffic distribution. We point out, however, that this scenario is not very likely in the case of LANs and MANs because the traffic tends to be burst. Furthermore, even in this case, DMAC could achieve a throughput of 2.2 times the single channel bandwidth.

APPENDICES

We have included in the appendices the code in C for GFM, LFM, and DMAC simulation programs respectively.

A- GFM Simulation Program

```

/*                                GFM  SIMULATION                                */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define n 10
#define c 100000
#define slotsize 1000
#define slotmax 2*n

main ()

{

int i,SlotN[n+1],SlotNO[n+1],NP[n+1],MdestN[n+1];
int SLdest[slotmax +1];
int nslot,oslot,nmsgSAT[n+1],flag1[n+1],flag2[n+1];
int slotSATB[slotmax], k=5,l=1,j;
long int tmsg=0;
long int nmsg[n+1];
float Tar[n+1],delay[n+1],u;
float tar=0,Arate,treal=0,tslot=0.01,averdelay=0;

float arriv_time(float);
int destination (int , int);

printf("Please Enter the Utilization .....");
scanf("%f", &u);
printf("c = %ld , utiliz =%f , slotsize = %d
",c,u,slotsize );
Arate = (u*c)/(slotsize * n * 10.0 );
printf ( "\n Arate = %f ",Arate );
/* initializaton part */

for (i=1;i <= n;i++)
{
/* Init. the SAT protocol variables */

flag1[i] = 0;
flag2[i] = 0;
SlotNO[i] = 2*i - 1;
delay[i] = 0;
nmsg[i] = 0;
nmsgSAT[i]=0;
Tar[i]= arriv_time(Arate);
MdestN[i]= destination(i,n);

```



```

        NP[i]=10;
        SlotN[i]= 2*i - 1;
    }

for(i=1;i <= slotmax; i++)
{
    SLdest[i] = 0;
    slotSATB[i] = 0;
}
slotSATB[1] = 1; /* assign the SAT with slot 1 on the outer
                  Ring. */

/* Start the Simulation until Tx. 500000 messages. */

while ( tmsg <500000)
{
    /* Do this in tslot steps */

    for(i=1; i <=n ; i++)
    {
        /* check if station i recieved the SAT */

        oslot = SlotNO[i];
        if ( slotSATB[oslot] == 1 )

        {
            /* Check if the node is satisfied or buffer empty */

            if ( (nmsgSAT[i] >= 1) || (Tar[i] > treal) )
            {
                nmsgSAT[i]=0; /* # of messages between 2
                               sucs. SAT = 0 */
            }
            else /* station not satisfied */

            {
                slotSATB[oslot] = 0 ; /* station i deleted the
                                       SAT */
                flag1[i] = 1; /* indicate station i deleted the
                               SAT*/
            }
        }

        /* if station i did not recieved the SAT check if it
           is the one who deleted the SAT & it is now
           satisfied */

    }

    else

```

```

{
    if ( (flag1[i] == 1 ) && flag2[i] == 1)
    {
        slotSATB[oslot] = 1; /* put the SAT on the Ring */
        flag1[i] = 0;
        flag2[i] = 0;
    }
}

/* when the destination = i make the slot that i
   recieved empty */

nslot = SlotN[i];
if (SLdest[nslot] == i)
    SLdest[nslot] = 0;

/* station i will transmit. IF it has an arrival and #
   of messages between successive SATs < k and the
   slot it received is empty */

if (Tar[i] < treal )
{
    if ( (SLdest[nslot] == 0) && (nmsgSAT[i] < k ) )
    {
        SLdest[nslot]=MdestN[i]; /* put the
                                   dest.address. */
        NP[i]=NP[i] - 1;
        if (NP[i] == 0) /* check if the whole message
                           is Tx. */
        {
            delay[i]=delay[i]+treal+tslot-Tar[i];
            nmsg[i]=nmsg[i]+1;
            tmsg=tmsg+1;
            switch ( tmsg )
            {
                case 1000:
                    printf("tmsg = %ld ",tmsg);
                    break;
                case 5000:
                    printf("tmsg = %ld ",tmsg);
                    break;
                case 10000:
                    printf("tmsg = %ld ",tmsg);
                    break;
                case 25000:
                    printf("tmsg = %ld ",tmsg);
                    break;
            }
            tar=arriv_time(Arate);
            Tar[i]=Tar[i]+tar;
            MdestN[i]= destination(i,n);
            nmsgSAT[i]=nmsgSAT[i] + 1;
            NP[i]=10;
        }
    }
}

```

```

        /* if station i was the one who deleted the
           sat and it becomes sat. after the
           transmit of this message it will set
           flag2[i] to indicate that*/

        if ( flag1[i] == 1)
        {
            if ((Tar[i] > treal) || ( nmsgSAT[i] >=1 ))
                flag2[i]=1;
        }
    }
}

/* The state on the network for the next tslot */
treal=treal+tslot;

for (i=1;i<=n;i++)
{
    SlotN[i]=SlotN[i] - 1;
    if ( SlotN[i] == 0 )
        SlotN[i] = slotmax;

    SlotNO[i] = SlotNO[i] + 1;
    if ( SlotNO[i] == slotmax )
        SlotNO[i] = 1;
}

}

for(i=1; i <=n ;i++)
printf("nmsg[%d]= %ld          Average delay [%d] = %f\n",
       i,nmsg[i],i,delay[i]/nmsg[i]);
for(i=1;i<=n;i++)
averdelay=averdelay + (delay[i]/nmsg[i] );
averdelay= averdelay / n ;
printf("average delay in the system = %f \n",averdelay);
printf("treal = %f   total messages TX. = %ld \n",
       treal,tmsg);
printf(" # of stations = %d ",n);
}/* End main */

```

```

int destination (int j , int t)
{

```

```
int ndest,k;
float x;
float random1 ();
x=random1();
k = ( (n-1)/2.0)*x +1;
ndest = k + j;
if (ndest > t)
    ndest = ndest - t;
return(ndest);
}

float random1()
{
    float y,z;
    int i;
    z=RAND_MAX;
    y=rand()/(z+1);
    return(y);
}

float arriv_time(float rate)
{
    float x,y;
    y=random1();
    x= (-1) * (log(1-y)/rate);
    return(x);
}
```

B- LFM Simulation Program

```

/*                                LFM SIMULATION                                */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define n 10
#define slot_size 1000
#define slots_Bet_stations 6
#define slotmax n*slots_Bet_stations
#define c 1000000

main ()
{
    /*Definition of Variables */

    struct que {
        int dest;
        float Tar;
        int NP;
    };

    struct status {
        struct que Q[n+1];
        int slot_N_A;
        int slot_N_B;
        long int tnmsg;
        float del;
        int mode;
        int Req_ID;
        int nmsg;
    };

    struct status stat[n+1];

    struct slot {
        int Req;
        int Req_ID;
        int Gnt;
    };

    struct slot slot_Ring_B[slotmax+1];

```

```

int i,j,SLdest_A[slotmax+1];
int nslot_A,nslot_B,quota=2;
long int tmsg=0;
float Arate=3.0,tar=0,treal=0,tslot=0.01,averdelay=0;

int H_to_B[n+1];      /* flag indicates station was H, and
                       upstream is busy */
int satisfied[n+1]; /* flag is set when station was T or
                       H/T and it become satisfied */
int starved[n+1];    /* flag is set when station was in
                       F.A.and it become starved */

float arriv_time(float);
int destination (int , int);
float randoml();

/* Initialization */

for (i=1;i <= n;i++)
{
  satisfied[i] = 0;
  starved[i] = 0;
  H_to_B[i] = 0;

  stat[i].del = 0;
  stat[i].tnmsg = 0;
  stat[i].mode = 0;
  stat[i].Req_ID = 0;
  stat[i].nmsg = 0;
  stat[i].slot_N_A = (slots_Bet_stations * i) - 1;
  stat[i].slot_N_B = (slots_Bet_stations * i) - 1;
  stat[i].Q[1].Tar = arriv_time(Arate) ;
  stat[i].Q[1].dest= destination(i,n);
  stat[i].Q[1].NP = 10;
  for (j=2;j <= 10;j++)
  {
    tar= arriv_time(Arate) ;
    stat[i].Q[j].Tar = stat[i].Q[j-1].Tar + tar;
    stat[i].Q[j].dest= destination(i,n);
    stat[i].Q[j].NP = 10;
  }
}

/* initialize the slots on both rings */

for(i=1;i <= slotmax; i++)
{
  SLdest_A[i]=0;
  slot_Ring_B[i].Req = 0;
  slot_Ring_B[i].Req_ID = 0;
  slot_Ring_B[i].Gnt = 0;
}

```

```

/*                starting of the simulation                */

while ( tmsg < 500000 )
{

for (i=1;i<=n;i++)
{
    /* *****
    *           Operation on Ring B           *
    ******/

nslot_B = stat[i].slot_N_B;

switch ( stat[i].mode )
{
    case 0 :    /* state is FA */

        if ( (slot_Ring_B[nslot_B].Req == 1) &&
              (starved[i] == 1) )
            {
                starved[i] = 0;
                stat[i].mode = 2 ;
                if (slot_Ring_B[nslot_B].Req_ID < i )
                    {
                        slot_Ring_B[nslot_B].Req_ID = i;
                        stat[i].Req_ID = i;
                    }
                else
                    {
                        stat[i].Req_ID = slot_Ring_B[nslot_B].Req_ID;
                    }
            }
        else
            if ( (slot_Ring_B[nslot_B].Req == 1 ) &&
                  (starved[i] == 0) )
                {
                    stat[i].mode = 3;
                    stat[i].Req_ID = slot_Ring_B[nslot_B].Req_ID;
                    slot_Ring_B[nslot_B].Req = 0;
                    slot_Ring_B[nslot_B].Req_ID = 0;
                }
        else
            if ( (slot_Ring_B[nslot_B].Req == 0 ) &&
                  (starved[i] == 1) )
                {
                    starved[i] = 0;
                    stat[i].mode = 1;
                    slot_Ring_B[nslot_B].Req = 1;
                }
            }
}
}

```



```

        stat[i].Req_ID = i;
        slot_Ring_B[nslot_B].Req_ID =
            stat[i].Req_ID;
    }

    break;

case 1 :    /* state is T */

    if ((slot_Ring_B[nslot_B].Req == 1) &&
        (satisfied[i] == 1))
    {
        satisfied[i] = 0;
        stat[i].mode = 3;
        slot_Ring_B[nslot_B].Gnt = 1;
        stat[i].nmsg = 0;
        stat[i].Req_ID = slot_Ring_B[nslot_B].Req_ID;
        slot_Ring_B[nslot_B].Req = 0;
        slot_Ring_B[nslot_B].Req_ID = 0;
    }

else
    if ( (slot_Ring_B[nslot_B].Req == 1) &&
        (satisfied[i] == 0) )
    {
        if (slot_Ring_B[nslot_B].Req_ID == stat[i].Req_ID)
        {
            stat[i].mode = 4;
            slot_Ring_B[nslot_B].Req = 0;
            slot_Ring_B[nslot_B].Req_ID = 0;
        }
        else
        {
            if (slot_Ring_B[nslot_B].Req_ID >
                stat[i].Req_ID)
            {
                stat[i].mode = 2;
                stat[i].Req_ID =
                    slot_Ring_B[nslot_B].Req_ID;
            }
            else
            {
                stat[i].mode = 2;
                slot_Ring_B[nslot_B].Req = 0;
                slot_Ring_B[nslot_B].Req_ID = 0;
            }
        }
        /* end else */
    } /* end if req = 1 && satisfied = 0 */

else
    if ( (slot_Ring_B[nslot_B].Req == 0) &&
        (satisfied[i] == 1) )
    {
        satisfied[i] = 0;
    }

```

```

        slot_Ring_B[nslot_B].Gnt = 1;
        stat[i].mode = 0;
        stat[i].Req_ID = 0;
        stat[i].nmsg = 0;
    }

    break;

case 2 : /* state is B */

    if (slot_Ring_B[nslot_B].Gnt == 1)
    {
        stat[i].mode = 1;
        slot_Ring_B[nslot_B].Gnt = 0;
    }

    if (slot_Ring_B[nslot_B].Req == 1)
    {
        if ( slot_Ring_B[nslot_B].Req_ID ==
                stat[i].Req_ID)
        {
            stat[i].mode = 4;
            slot_Ring_B[nslot_B].Req_ID = 0;
            slot_Ring_B[nslot_B].Req = 0;
        }
        else
            if ( slot_Ring_B[nslot_B].Req_ID >
                    stat[i].Req_ID )
            {
                stat[i].Req_ID = slot_Ring_B[nslot_B].Req_ID;
            }
        else
            if ( slot_Ring_B[nslot_B].Req_ID <
                    stat[i].Req_ID )
            {
                slot_Ring_B[nslot_B].Req_ID = 0;
                slot_Ring_B[nslot_B].Req_ID = 0;
            }
    }

    } /* end if (Req = 1 & satisfied = 0) */

    break;

case 3 : /* state is H */

    if ((slot_Ring_B[nslot_B].Req==0) &&
        (slot_Ring_B[nslot_B].Gnt==0))
    {
        if ( H_to_B[i] == 1)
        {
            H_to_B[i] = 0;
            stat[i].mode = 2;
            slot_Ring_B[nslot_B].Req = 1;
            slot_Ring_B[nslot_B].Req_ID = stat[i].Req_ID;
        }
    }

```

```

    }
  } /* end if (req = 0) && (gnt = 0) but upstream
      was busy */

else
{
  if ( slot_Ring_B[nslot_B].Gnt == 1)
  {
    if ( H_to_B[i] == 1 )
    {
      H_to_B[i] = 0;
      stat[i].mode = 1;
      slot_Ring_B[nslot_B].Gnt = 0;
      slot_Ring_B[nslot_B].Req_ID = stat[i].Req_ID ;
      slot_Ring_B[nslot_B].Req = 1 ;
    }
    else
    {
      stat[i].mode = 0;
      stat[i].Req_ID = 0;
      slot_Ring_B[nslot_B].Gnt = 0;
      stat[i].nmsg = 0;
    }
  } /* end Gnt = 1 */

  if ( slot_Ring_B[nslot_B].Req == 1 )
  {
    if ( H_to_B[i] == 1 )
    {
      H_to_B[i] = 0;
      if ( slot_Ring_B[nslot_B].Req_ID ==
          stat[i].Req_ID )
      {
        stat[i].mode = 4;
        slot_Ring_B[nslot_B].Req_ID = 0;
        slot_Ring_B[nslot_B].Req = 0;
      }
      else
      {
        if ( slot_Ring_B[nslot_B].Req_ID >
            stat[i].Req_ID )
          stat[i].Req_ID =
            slot_Ring_B[nslot_B].Req_ID ;
        else
        {
          slot_Ring_B[nslot_B].Req = 0;
          slot_Ring_B[nslot_B].Req_ID = 0;
        }
      }
    }

    } /* end if H_to_B = 1 */

    else /* i.e if(Req = 1) & (H_to_B = 0) */
    {

```

```

        if (slot_Ring_B[nslot_B].Req_ID >
            stat[i].Req_ID )
        {
            stat[i].Req_ID =
                slot_Ring_B[nslot_B].Req_ID;
            slot_Ring_B[nslot_B].Req_ID = 0;
            slot_Ring_B[nslot_B].Req = 0;
        }
        else
        {
            slot_Ring_B[nslot_B].Req_ID = 0;
            slot_Ring_B[nslot_B].Req = 0;
        }
    } /* end else */

} /* end if ( Req = 1) */

} /* end else of if req =0 & gnt = 0 */

break;

case 4 : /* state is H/T */

    if ( satisfied[i] == 1 )
    {
        satisfied[i] = 0;
        stat[i].mode = 3;
        slot_Ring_B[nslot_B].Gnt = 1;
    }

    if (slot_Ring_B[nslot_B].Req == 1 )
    {
        if (slot_Ring_B[nslot_B].Req_ID > stat[i].Req_ID
            )
        {
            stat[i].Req_ID = slot_Ring_B[nslot_B].Req_ID;
            slot_Ring_B[nslot_B].Req_ID = 0;
            slot_Ring_B[nslot_B].Req = 0;
        }
        else
        {
            slot_Ring_B[nslot_B].Req_ID = 0;
            slot_Ring_B[nslot_B].Req = 0;
        }
    } /* end if (Req = 1) */
    break;

} /* end of switch statement */

/* ***** *
 * Operations on Ring A *

```

```

* *****/

/* check if the dest. of the slot on ring A is
   station i */

nslot_A = stat[i].slot_N_A;
if (SLdest_A[nslot_A] == i)
    SLdest_A[nslot_A] = 0;

/*****
* Check if slot is busy, then if state = H then set H_to_B *
* flag and if the state is F.A. set starved flag *
*****/

if (SLdest_A[nslot_A] != 0)
{
    if (stat[i].mode == 3)
    {
        H_to_B[i] = 1;
    }
    if ((stat[i].mode == 0) && (stat[i].Q[10].Tar <
                               treal ) )
    {
        starved[i] = 1;
    }
}

/* Tx. of the packets */

if (stat[i].Q[1].Tar < treal )
{
    if (SLdest_A[nslot_A] == 0)
    {
        if((stat[i].mode == 0) || ((stat[i].mode != 0)
                                   && (stat[i].nmsg < quota)) )
        {
            SLdest_A[nslot_A] = stat[i].Q[1].dest;
            stat[i].Q[1].NP = stat[i].Q[1].NP - 1;
            if (stat[i].Q[1].NP == 0)
            {
                stat[i].del=stat[i].del+treal+tslot-
                           stat[i].Q[1].Tar;
                stat[i].tnmsg=stat[i].tnmsg + 1;

                if (stat[i].mode != 0)

```

```

        /* only if station in Restricted mode
           then increase nmsg */
        {
            stat[i].nmsg = stat[i].nmsg + 1;
        }

        tmsg=tmsg+1;
        for (j=1;j <= 9 ;j++)
        {
            stat[i].Q[j] = stat[i].Q[j+1];
        }

        tar=arriv_time(Arate);
        stat[i].Q[10].Tar = stat[i].Q[9].Tar +
                                                                    tar;
        stat[i].Q[10].dest= destination(i,n);
        stat[i].Q[10].NP=10;
    }
}

} /* end if slot is empty */

} /* end if (stat[i].Q[1].Tar < treal ) */

/*****
*check if the station is T or H/T and it become satisfied *
* set satisfied flag *
*****/

        if ((stat[i].mode == 1) || (stat[i].mode == 4))
        {
            if((stat[i].nmsg >= quota) ||
                (stat[i].Q[1].Tar > treal))
                satisfied[i] = 1;
        }

    } /* end for (i=1;i<=n;i++) */
/*****
* Change the state of the system for next tslot *
*****/

        treal=trealt+tslot;
        for (i=1;i<=n;i++)
        {
            stat[i].slot_N_A = stat[i].slot_N_A - 1;
            if ( stat[i].slot_N_A == 0 )
                stat[i].slot_N_A = slotmax;

            stat[i].slot_N_B = stat[i].slot_N_B + 1;
            if ( stat[i].slot_N_B > slotmax )

```

```

        stat[i].slot_N_B = 1 ;
    }

} /* end While */

for(i=1; i <=n ;i++)
{
    stat[i].del = stat[i].del / stat[i].tnmsg;
    printf("tnmsg[%d]= %ld          Average delay [%d] = %f\n",
           i,stat[i].tnmsg,i,stat[i].del);
}
for(i=1;i<=n;i++)
averdelay=averdelay + stat[i].del;
printf("average delay in the system = %f\n",averdelay/10.0);
printf("arrival rate for each station = %f\n",Arate);
printf("treal = %f          quota= %d\n" ,treal,quota);
printf("tmsg = %ld      slots between stations = %d\n",tmsg,slots_Bet_stations);
} /* end main */

```

```

int destination (int j , int t)

```

```

{
    int ndest,k;
    float x;
    float random1 ();
    x=random1();
    k = ((t-1.0)/2.0)*x +1;
    ndest = k + j;
    if (ndest > t)
        ndest = ndest - t;
    return(ndest);
}

```

```

float random1()

```

```

{
    float y,z;
    int i;
    z=RAND_MAX;
    y=rand()/(z+1);
    return(y);
}

```

```
float arriv_time(float rate)
{
    float x,y;
    y=random1();
    x= (-1) * (log(1-y)/rate);
    return(x);
}
```


C- DMAC Simulation Program

```

/*          DMAC SIMULATION          */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define n 10
#define slot_size 1000
#define slots_Bet_stations 2
#define slotmax n*slots_Bet_stations
#define c 1000000

main ()

{
        /*          Definition of Variables          */

        struct que {

                int      dest;
                float    Tar;
                int      NP;

        } ;

        struct status {

                struct que      Q[n+1];
                int      slot_N_A;
                int      slot_N_B;
                long int  tnmsg;
                float    del;
                int      UNRG_CTR;
                int      RG_CTR;
                int      RQ_CTR;
                int      BWB_CTR;
                int      URQ_CTRF;
                int      URQ_CTRP;
                int      DURQ_CTR;
                int      DURQ_CTR_ID;
                int      TAR_flag;
                int      CTRF_FLAG;
                int      M;
                int      ES_CTR;
                int      HEAD;

```

```

        int          RB_ZERO_CTR;
        int          TAR_CTR_ERASED;
        int          TAR_CTR_INSERTED;
    } ;

    struct status   stat[n+1];

    int
i, j, SL_Dest_A[slotmax+1], SL_TAR_A[slotmax+1], SL_RB_B[slotmax+
1];
    int nslot_A, nslot_B, SL_REQ_ID[slotmax+1];
    long int tmsg=0, REQ_DEL=0;
    float Arate=4000.0, tar=0, treat=0.001, averdelay=0.0;
    float tslot = 0.01;

    float arriv_time(float);
    int destination (int , int);
    float random1();

        /*          Initialization          */

    for (i=1; i <= n; i++)
    {

        stat[i].del = 0;
        stat[i].tnmsg = 0;
        stat[i].UNRG_CTR = 1;
        stat[i].RG_CTR = 1;
        stat[i].RQ_CTR = 0;
        stat[i].BWB_CTR = 0;
        stat[i].URQ_CTRF = 1;
        stat[i].URQ_CTRP = 0;
        stat[i].DURQ_CTR = 0;
        stat[i].DURQ_CTR_ID = 0;
        stat[i].TAR_flag = 0;
        stat[i].CTRF_FLAG = 0;
        stat[i].M = 4;
        stat[i].slot_N_A = (slots_Bet_stations * i) - 1;
        stat[i].slot_N_B = (slots_Bet_stations * i) - 1;
        stat[i].Q[1].Tar = arriv_time(Arate) ;
        stat[i].Q[1].dest= destination(i,n);
        stat[i].Q[1].NP = 1;
        stat[i].HEAD = 10;
        stat[i].RB_ZERO_CTR = 0;
        stat[i].TAR_CTR_ERASED = 0;
        stat[i].TAR_CTR_INSERTED = 0;
        stat[i].ES_CTR = 0;
    }

        /* Initialize the Slots on Both Rings */

```

```

for(i=1;i <= slotmax; i++)
{
    SL_Dest_A[i]=0;
    SL_TAR_A[i] = 0;
    SL_RB_B[i] = 0;
    SL_REQ_ID[i] = 0;
}

/*          Starting of the Simulation
*/

while ( tmsg < 50000 )
{

    for (i=1;i<=n;i++)
    {
        /* *****
        *          Operation on Ring B          *
        *          ***** **/
        nslot_B = stat[i].slot_N_B;

        /*** Remove the request if its circulating around
the ring ***/

        if ( SL_RB_B[nslot_B] == 1 )
        {
            if ( SL_REQ_ID[nslot_B] > i )
            {
                if ( (SL_REQ_ID[nslot_B] - i) == 0 )
                {
                    SL_RB_B[nslot_B] = 0;
                    SL_REQ_ID[nslot_B] = 0;
                }
            }
            else
            {
                if( (i - SL_REQ_ID[nslot_B]) == 0 )

                {
                    SL_RB_B[nslot_B] = 0;
                    SL_REQ_ID[nslot_B] = 0;
                }
            }
        }

        if (SL_RB_B[nslot_B] == 0)
        {

```



```

        SL_RB_B[nslot_B] = 0;
        stat[i].ES_CTR --;
    }
}
} /* end if ES_CTR > 0 */
else
{
    if ( ( stat[i].DURQ_CTR == 0 ) && (
stat[i].CTRF_FLAG == 0 ) )
    {
        SL_RB_B[nslot_B] = 0;
        stat[i].DURQ_CTR ++;
        stat[i].DURQ_CTR_ID = SL_REQ_ID[nslot_B];
    }
} /* end else */
} /* end else (i.e RB = 1 ) */

/* *****
*
*                               Operations on Ring A
*
* ***** */

nslot_A = stat[i].slot_N_A;

    /**** Erasing a slot *****/

    if (SL_Dest_A[nslot_A] == i)
    {
        SL_Dest_A[nslot_A] = 0;

        if ( ( stat[i].RQ_CTR > 0 ) || ( ( stat[i].RQ_CTR ==
0) &&
(stat[i].RG_CTR > 0) ) )
            stat[i].ES_CTR ++;

        if ( SL_TAR_A[nslot_A] == 1 )
        {
            SL_TAR_A[nslot_A] = 0 ;
            stat[i].TAR_CTR_ERASED ++;
        }
    }

    /**** Slot is empty event *****/

    if (SL_Dest_A[nslot_A] == 0)
    {
        if ( stat[i].HEAD > 0 )
            stat[i].HEAD --;
    }

```

```

        if(stat[i].RQ_CTR > 0)
        {
            stat[i].RQ_CTR --;
            if ( (stat[i].DURQ_CTR > 0) && (stat[i].RQ_CTR
== 0 ) &&
            (stat[i].HEAD<= 0 ) )
            {
                stat[i].DURQ_CTR --;
                if ( ( i != 1 ) )
                {
                    REQ_DEL ++;
                    if ( fmod(REQ_DEL,100.0) == 0.0 )
                        printf(" REQ_DEL = %ld ", REQ_DEL);
                    printf("\n %d  REQ_ID = %d ", i,
stat[i].DURQ_CTR_ID );
                }
            }
        }

    }
else /* (i.e RQ_CTR = 0 ) */
{

    if(stat[i].UNRG_CTR == 0)
        stat[i].UNRG_CTR = 10 ;

    if ( stat[i].RG_CTR + stat[i].UNRG_CTR > 0 )
    {

        /* Tx. of the packets */

        SL_Dest_A[nslot_A] = stat[i].Q[1].dest;
        stat[i].Q[1].NP = stat[i].Q[1].NP - 1;
        if (stat[i].Q[1].NP == 0)
        {
            stat[i].del=stat[i].del+treal+tslot-
stat[i].Q[1].Tar;

            stat[i].tnmsg=stat[i].tnmsg + 1;

            tmsg=tmsg+1;
            tar=arriv_time(Arate);
            stat[i].Q[1].Tar = stat[i].Q[1].Tar +
tar;

            stat[i].Q[1].dest= destination(i,n);
            stat[i].Q[1].NP=1;
            stat[i].CTRF_FLAG = 0;

        }
    }
}

```

```

    /**** Events after Segement Transmission.    ****/

        if ((stat[i].TAR_flag ==
1)&&(stat[i].RG_CTR == 0))
            stat[i].UNRG_CTR --;
        else
            stat[i].RG_CTR --;

        if((stat[i].TAR_flag == 1)  && (
stat[i].ES_CTR > 0 ) )
            stat[i].ES_CTR --;

        stat[i].TAR_flag = 0;
        stat[i].BWB_CTR ++;
        if (stat[i].BWB_CTR == stat[i].M)
        {
            SL_TAR_A[nslot_A] = 1;
            stat[i].TAR_CTR_INSERTED ++;
            stat[i].BWB_CTR = 0;
            stat[i].TAR_flag = 1;
        }

    /** Event segment becomes first in queue **/

        if ((stat[i].TAR_flag ==0)&&(stat[i].RG_CTR
            > 0)&& (stat[i].HEAD != 0))
            stat[i].URQ_CTRF ++;
        if ((stat[i].TAR_flag ==0)&&(stat[i].RG_CTR
            == 0)&& (stat[i].HEAD != 0))
        {
            stat[i].URQ_CTRF ++;
            stat[i].RG_CTR ++;
            stat[i].UNRG_CTR --;
        }

        } /*End if (UNRG_CTR + RG_CTR) = 0 */

    else /* i.e REQ_CTR = 0 and the station has no
        messages */
        {
            stat[i].URQ_CTRF = 0;
            stat[i].DURQ_CTR = 0;
        }

    } /* end else RQ_CTR = 0 */

} /* end if slot is empty */

    /***** Slot is busy event *****/

else /* i.e slot is busy */

```



```

{
  if ( ( stat[i].RG_CTR + stat[i].UNRG_CTR) > 0)
    stat[i].CTRF_FLAG = 1;
  stat[i].HEAD = 100;
  if ( ( i >= 1 ) && ( i <= 10 ) )
    {
      if (SL_TAR_A[nslot_A] == 1)
        {
          SL_TAR_A[nslot_A] = 0;
          stat[i].TAR_CTR_ERASED ++;
          stat[i].URQ_CTRF ++;
          stat[i].RG_CTR = stat[i].RG_CTR + stat[i].M;
          /* if station i is a head of a path it
             should not Tx. Req. */

          if ( stat[i].HEAD == 0 )
            { stat[i].URQ_CTRF --;
              stat[i].RG_CTR = stat[i].RG_CTR -
                stat[i].M;
            }

        }

    }

} /* End slot is busy */

} /* end for (i=1;i<=n;i++) */

/* *****
 * Change the state of the system for the next tslot*
 * ***** */

treal=treal+tslot;
for (i=1;i<=n;i++)
{
  stat[i].slot_N_A = stat[i].slot_N_A - 1;
  if ( stat[i].slot_N_A == 0 )
    stat[i].slot_N_A = slotmax;

  stat[i].slot_N_B = stat[i].slot_N_B + 1;
  if ( stat[i].slot_N_B > slotmax )
    stat[i].slot_N_B = 1 ;

}

} /* end While */

for(i=1; i <=n ;i++)
{
  if (stat[i].tnmsg > 0)
  {
    stat[i].del = stat[i].del / stat[i].tnmsg;

```

```

printf("%d tmsg = %ld  delay = %f URQ_CTRF = %d\n
      ",i,stat[i].tmsg,stat[i].del,stat[i].URQ_CTRF);
printf("stat[%d] erased %d TARs & inserted %d TARs
      \n",i,stat[i].TAR_CTR_ERASED,stat[i].TAR_CTR_INSERTED);
}
}
for(i=1;i<=n;i++)
averdelay=averdelay + stat[i].del;

printf("treal = %f      \n" ,treal);
printf("tmsg = %ld      slots between stations =
      %d\n",tmsg,slots_Bet_stations);

} /*  end main  */

```

```

int destination (int j , int t)

```

```

{
  int ndest,k;
  float x;
  float random1 ();
  x=random1();
  k = ((t-1.0)/2.0)*x +1;
  ndest = k + j;
  if (ndest > t)
    ndest = ndest - t;
  return(ndest);
}

```

```

float random1()

```

```

{
  float y,z;
  int i;
  z=RAND_MAX;
  y=rand()/(z+1);
  return(y);
}

```

```

float arriv_time(float rate)

```

```

{
  float x,y;
  y=random1();
  x= (-1) * (log(1-y)/rate);
  x = 0.0;
  return(x);
}

```

REFERENCES

1. Kessler, Gary C. and David A. Train. Metropolitan Area Networks: Concepts, Standards, and Services. McGraw-Hill, Inc. New York, 1991.
2. Cidon, Israel and Yoram Ofek. "Metaring - A Full-Duplex Ring with Fairness and Spatial Reuse." IBM Research Report RC14961, September 22, 1989.
3. Ofek, Yoram. "Integration of Multi-ring on the Metaring Architecture." IBM Research Report RC16058, August 27, 1990.
4. Cidon, Israel and Yoram Ofek. "Distributed Fairness Algorithms for Local Area Networks with Concurrent Transmissions." IBM Research Report RC15051, October 17, 1989.
5. Chen, J., Israel Cidon and Yoram Ofek. "A Local Fairness Algorithm for Gigabit LANs/MANs with Spatial Reuse." IBM Research Report RC18114, June 24, 1992.
6. IEEE 802.6 Working Group. "Proposed Standards, Distributed Queue Dual Bus(DQDB) Metropolitan Area Network." Unapproved Draft D9, August 1989.
7. M. Conti et al. "A Methodological Approaches to an Extensive Analysis of DQDB Performance and Fairness." IEEE J. Select. Areas Commun., Vol. SAC-9, n0.1, pp. 76-87, January 1991.

8. E.L. Hahne et al. "Improving the Fairness of Distributed Queue Dual Bus Networks." Proceed. INFOCOM'90, San Francisco, pp. 175-184, June 1990.
9. Karvelas, D., and M. Papamichail, "DQDB : A FAST Converging Bandwidth Balancing Mechanism that Requires No Bandwidth loss." Proceed. ICC'92, Chicago, June 14-18, 1992.
10. Karvelas, D., and M. Papamichail, " The No Slot Wasting Bandwidth Mechanism for Dual Bus Architecture ." Technical Report CIS-92-15, CIS Dept., New Jersey Institute of Technology.
11. Karvelas, D., and M. Papamichail, "Performance Study of a New Bandwidth Balancing Mechanism Under a Single and Multiple Priority Classes of Traffic." Proceed. First International Conf. on Computer Commun. Networks, San Diego, June 8-10, 1992.
12. Karvelas, D., and M. Papamichail. "Performance Analysis of a New Bandwidth Balancing Mechanism Under the Presence of Erasure Nodes." Proceed. INFOCOMM'93, San Francisco, March 30 - April 1, 1993.