ABSTRACT

An Efficient Guaranteed Bandwidth and Balancing Mechanism for High Speed MANs

by
Venediktos Hadjisavvas

The Distributed Queue Dual Bus (DQDB) has become the IEEE 802.6 standard for Metropolitan Area Networks (MANs). The main advantage of DQDB is that its throughput performance is not affected by the network parameters such as size, number of connected stations, or channel bandwidth. Its main drawback is that the location of the stations on the bus strongly affects their performance. For this reason a Bandwidth Balancing Mechanism (BBM_DQDB) has been proposed and included in the 802.6 standard that can provide the requested bandwidth by the lightly loaded stations and evenly distribute the remaining bandwidth among the overloaded stations. The guaranteed bandwidth required by some applications has also motivated the recent introduction of another mechanism, the Guaranteed Bandwidth (GBW_DQDB) mechanism, that can guarantee the required level of throughput to certain high priority stations. In this thesis we first discuss the main advantages and disadvantages of BBM_DQDB and GBW_DQDB and then we introduce a new mechanism, the Guaranteed Bandwidth and Balancing Mechanism (GBBM), that combines the advantages of the previous two mechanisms and can significantly improve the throughput and delay performance of the stations. We provide a detailed description of the new mechanism and we investigate its performance through simulation results. Furthermore, we compare its performance with the corresponding performance of BBM_DQDB and GBW_DQDB.

AN EFFICIENT GUARANTEED BANDWIDTH AND BALANCING MECHANISM FOR HIGH SPEED MANs

by
Venediktos Hadjisavvas

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

October 1993

APPROVAL PAGE

An Efficient Guaranteed Bandwidth and Balancing Mechanism for High Speed MANs

Venediktos Hadjisavvas

August 9, 1993

Dr. Dennis Karvelas, Thesis Adviser
Assistant Professor of Computer Science, NJIT

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

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

BIOGRAPHICAL SKETCH

Author: Venediktos Hadjisavvas

Degree: Masters of Science in Electrical Engineering

Date: October 1993

Undergraduate and Graduate Education

- Master of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 1993

- Bachelor of Science in Electromichanical Computer Technology,
  New York Institute of Technology, New York, NY, 1990

Major: Electrical Engineering

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.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 High Capacity Networks

Data communication networks are essential for providing information exchange among various non homogeneous communicating devices such as personal computers, workstations, peripheral devices, facsimiles, etc. Local Area Networks (LANs) interconnect these devices within a room, a single building or a group of buildings located close to each other.

An essential characteristic of the local area networks which usually differentiates them from other kinds of networks is that the transmission from any station is received by all other stations (packet broadcasting). Typically, in a local area network there are no central control stations and all stations cooperate to ensure fair access to the transmission medium. Today's LANs operating at 1-10 Mbps support a quite variety of services such as file transfers, graphics applications, word processing, electronic mail, distributed databases and interconnection to other LANs.

The recent advances in fiber optics technology provide a huge bandwidth which enables service integration and opens up new prospects to the network users. It is now expected from the networks of the future to support massive data transfer between supercomputers as well as voice, video, and other types of real or non-real time services. In fact, although we can easily envision a large number of new applications, there will be others we cannot currently foresee. Such diverse services will generate flows of information with very different traffic characteristics, throughput and delay requirements. Therefore, one of the major challenges for the network designer is the efficient distribution of the available enormous channel bandwidth among the competing users.

There have been many interesting proposals on how to access and share efficiently a high capacity channel in the local area environment. Among them EX-

PRESSNET [1], FASNET [2], and FDDI [3,4] have received a great amount of attention. The extension of local area network services over longer distances, provides significant advantages in terms of increasing efficiency and productivity. The so called metropolitan area networks (MANs) are optimized for a larger geographical area than the LANs, ranging from several blocks of buildings to entire cities. MANs share many characteristics with LANs and also provide means for inter-networking with LANs. They use packet broadcasting over a shared transmission medium and are intended to provide high capacity services to their users at a much lower cost.

In both LANs and MANs, the data rate, length, and medium access control techniques that are employed, are key factors in determining the effective capacity of the network. From the user's point of view, the performance of a network is given by the following two measures: throughput and packet delay. Delay is the time interval from the generation of a packet at a station until its transmission onto the medium. In some cases the actual transmission time and the propagation time to the destination are also included in the computation of the delay. Throughput is defined as the total rate of data being transmitted between nodes. Another important network performance measure is the utilization of the transmission medium which is defined as the fraction of total capacity being used.

A very important parameter for determining the performance of a LAN is the parameter α, which is defined as:

$$\alpha = \frac{propagation\,time}{length\,of\,frame} = \frac{RD}{VL} \tag{1.1}$$

where

R = data rate (Mbps)

D = distance of the communication path (km)

V = propagation velocity (m/s)

L = length of the frame (bits)

$\frac{D}{V}$ = propagation time on the medium (worst case)

Typical values of α, range from a 0.01 to 0.1. It has been shown that the theoretical maximum possible utilization of a LAN using an IEEE 802 MAC scheme can be expressed as:

$$u = \frac{1}{1+\alpha} \tag{1.2}$$

For instance an 1km long LAN, with a data rate R of 10 Mbps, a propagation delay $(\frac{1}{V})$of 5μ sec $/km$ and a packet size of 1000 bits will have a value of $\alpha = 0.05$ and corresponding maximum utilization of 0.95. However, as the channel capacity and size of the network increase, i.e. as we move from the local area to the metropolitan area environment, the value of α increases significantly and the corresponding utilization decreases drastically. For instance 20 km, 100 Mbps MAN, with the same propagation delay of 5 μ sec $/km$ and same packet size of 1000 bits will have a value of $\alpha = 10$ and a corresponding maximum utilization of 0.091. Therefore it becomes evident, that the IEEE 802 MAC schemes are not appropriate for high speed MANs, and that new more efficient medium access mechanisms are needed.

The above challenging problem has motivated a great amount of research in the area of high speed MANs and several mechanisms have been proposed. The most prominent among them is the Distributed Queue Dual Bus (DQDB) MAC mechanism [5,6,7] whose throughput performance is not affected by the network size, the number of connected stations, or the channel bandwidth. For this reason DQDB has been accepted as the IEEE 802.6 standard for MANs. However, this mechanism has a major fairness problem. That is, the location of the stations on the network strongly affects the bandwidth or perceive the delay that their messages will encounter [8,9,10]. In order to deal with this problem a Bandwidth Balancing Mechanism (BBM_DQDB) has been recently proposed [11], that can provide the lightly loaded stations with the

bandwidth they have requested and evenly distribute the remaining bandwidth among the overloaded stations. This mechanism has also been included in the 802.6 standard. Although BBM_DQDB can fairly distribute the channel bandwidth among the various stations in overloaded conditions, the location of the stations still strongly affects their delays. Furthermore, it can not guarantee that the performance characteristics of high priority users will always be better than those of lower priority users. Finally, it can not guarantee that high priority stations can receive a certain level of throughput which is required by the applications they support.

The above disadvantages have motivated the recent introduction of another mechanism [12], called the Guaranteed Bandwidth Mechanism (GBW_DQDB), that can guarantee the bandwidth requested by certain higher priority stations. Nevertheless GBW_DQDB has also a drawback. That is, it does not enable a station to acquire idle bandwidth not used by other stations and therefore improve its performance.

In this thesis, we first provide a discussion on the advantages and disadvantages of BBM_DQDB and GBW_DQDB. Then, we introduce a new medium access mechanism that can provide guaranteed bandwidth to higher priority stations and at the same time enable them to use the available idle bandwidth . The new mechanism, called Guaranteed Bandwidth and Balancing Mechanism (GBBM), combines features from both BBM_DQDB and GBW_DQDB and is expected to achieve a much better performance. Two variations of GBBM are introduced and their corresponding performances are investigated. The second variation (GBBM2) which is a minor modification of the first one (GBBM1) improves the delay performance.

The organization of the thesis is as follows. In Chapter 2 we provide a brief description of the medium access control mechanism of DQDB, BBM_DQDB and GBW_DQDB, and discuss their advantages and disadvantages. In Chapter 3 we introduce the Guaranteed Bandwidth and Balancing Mechanism (GBBM). In Chapter 4 we investigate the performance of GBBM and we compared it with the correspond-

ing performances of BBM_DQDB and GBW_DQDB. Finally, in Chapter 5, we present our conclusions.

CHAPTER 2

CURRENT ACCESS MECHANISM OF HIGH SPEED MAN

2.1 Distributed Queue Dual Bus MAC Mechanism

The high bandwidth and long distances used in Metropolitan Area Networks (MANs), have made Distributed Queue Dual Bus (DQDB) the most appropriate protocol for satisfying the current bandwidth demands. DQDB has been adopted by IEEE as the 802.6 standard for MANs.

Some Slotted systems efficiently use only the transmission medium by allowing nodes with data to transmit on empty slots. Uncontrollable transmission, however, can create a major problem in the case of a single unidirectional bus, because the nodes which are closest to the origin see the idle slots first, they then can write on all of them, and prevent other downstream nodes from transmitting in idle slots. Unlike such Slotted systems, the Distributed Queue Dual Bus (DQDB) uses two unidirectional buses in which slots are travelling in opposite directions. This configuration enables downstream nodes to use the reverse bus to make slot reservations.

DQDB is a totally distributed protocol, and its throughput is not affected by the network size or the number of stations connected to it. A major drawback of DQDB, as extensive research in the area has indicated [8,9,10], is that its Medium Access Mechanism (MAC) exhibits unfair behavior. That is, the location of the stations has a very strong effect on both, throughput and delay performance.

The architecture of DQDB is shown in figure 1 and consists of two unidirectional buses and a multiplicity of nodes connected to these buses. The buses, denoted in Figure 1 as bus A and bus B, support data transfer in opposite directions allowing full duplex communication between the nodes. Each node can transmit data to any other node by selecting the appropriate bus. Both buses operate at all times, therefore, the capacity of the network is twice the capacity of a single bus.

Every station of the network has an access unit for each bus with the corresponding attachment to that bus. The access unit is attached to the bus via one read and one write connection. The first station of each bus generates fixed size slots that are travelling downstream and can be written by all stations having data for transmission. These slots are destroyed at the end of the bus.

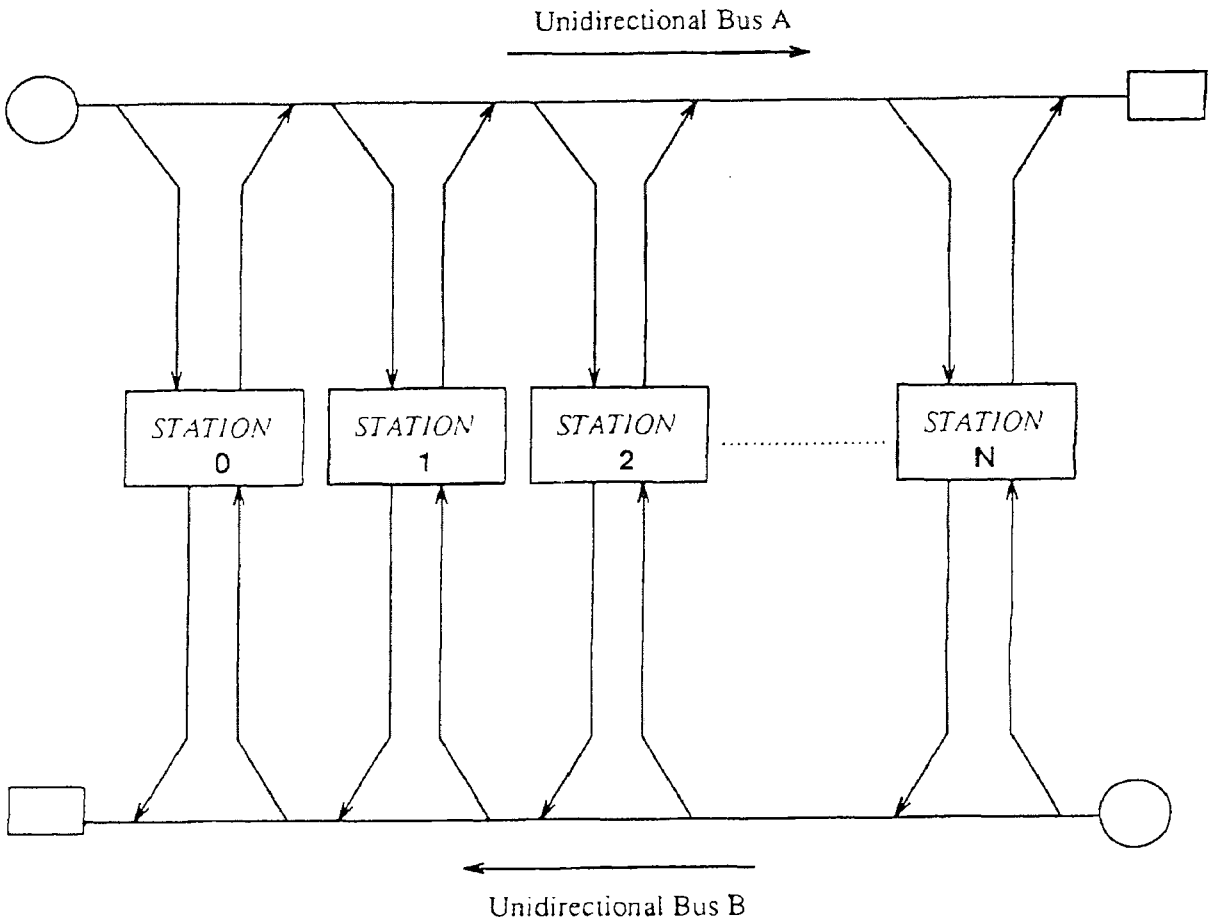

Figure 1 The DQDB Topology

The operation on the two busses is identical and therefore in the sequel we will focus on the operation of bus 'A'. We denote station '0' the first station on bus 'A' which is also responsible for generating the slots for that bus. In this case, bus A is called the forward bus and bus B is called the reverse bus.

When a station generates a message for transmission, it appends some header and trailer information to it and creates what is known as the Initial MAC Protocol

Data Unit (IMPDU). This overhead information contains among other things the source and destination address of the transmission, control bytes that have been reserved for future use, a CRC field, and control bytes that can facilitate the detection of lost packets or inform the receiver about buffer requirements. The slot size in DQDB is 53 bytes out of which 1 byte is used as Access Control Field, 8 bytes carry overhead information, and only the remaining 44 bytes carry useful information. Therefore each station once it has created an IMPDU it has to break it into fragments of 44 bytes and attach the appropriate headers to each fragment. If the last fragment of the original IMPDU is smaller that 44 bytes the station will add some padding information to extend it to 44 bytes.

The operation of DQDB protocol is based on two control bits: the BUSY bit and the REQUEST bit. The Busy bit indicates whether a slot travelling on the forward bus has already been written by another upstream station. The Request bit indicates whether a slot travelling on the reverse bus carries a request from a downstream station. Each station, counts both the number of requests it receives on bus 'B' and the unused slots that pass by on bus 'A' and in this way it can determine the number of empty slots that must be allowed to pass before it transmits its own segment.

When a station has a segment ready for transmission, it will send a single Request on the reverse bus. The station can do that by setting to 1 the first 0 request bit observed on the reverse bus. All the upstream stations will see the request bit and will increase their Request Counter (RQ_CTR) by one. If a station is idle, it will decrease its RQ_CTR by one for each empty slot that passes on the forward bus. In this way, RQ_CTR will keep a record of the number of segments queued to the downstream stations.

When a station is active (it has a segment to transmit) it will transfer the current value of RQ_CTR to a Count Down Counter (CD_CTR) and reset RQ_CTR

to zero. In this way CD_CTR contains the number of requests from the downstream stations that have sent reservations before that station had received this segment in its Transmission Queue. CD_CTR is now decreased by one for every idle slot that passes by on the forward bus. When the CD_CTR becomes zero, the given station can transmit its segment at the next empty slot observed on bus 'A'. While the station is waiting to transmit its own segment, it will keep on increasing its RQ_CTR by one for every request that observes on the reverse bus (Figure 2).

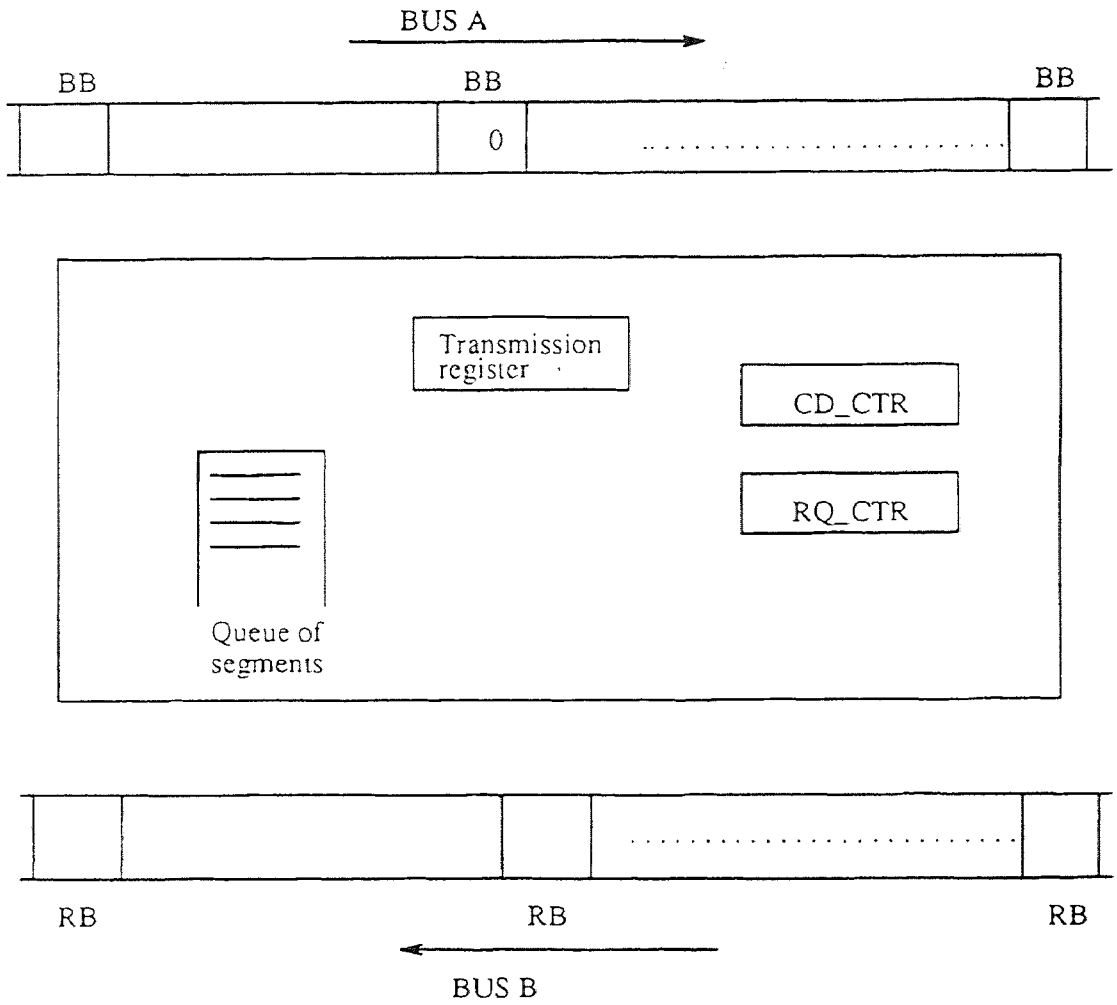

Figure 2 Internal structure of a station

Notice that if a station has transmitted a segment before it was able to send the request bit on the reverse bus, because other downstream stations have already set

these bits to 1, the station will send the request after the transmission of the segment. That is, the operations of writing request bits on the reverse bus and sending segments on the forward bus are independent.

A priority mechanism has also been included in the 802.6 standard that can support three priority classes of traffic. In this case a separate request bit is required for each priority class in the ACF of a slot. Furthermore, each station must have a separate request and count down counters. Priorities are introduced in the following way. The RQ_CTR of each class counts requests of similar and higher priority. Furthermore the CD_CTR of each class increases whenever a request bit of higher priority is observed on the reverse channel.

The objective of this priority mechanism is to enable segments of higher priority to have access on to the channel ahead of segments of lower priority. That is, to create three global queues and allow low priority segments to be transmitted only when the higher priority queues are empty. Although this is accomplished in the case of 0 latency, i.e., when the cable size is 0 or it is a good approximation in the case of a small network, as the network size increases this priority mechanism becomes ineffective. Higher priority users have absolute priority over lower priority users only inside the same station. Among users inside different stations, their relative location has a significant effect on their performance. That is, depending on their location on the bus, the performance characteristics of low priority users may be significantly better than those of high priority users [11].

The major advantage of DQDB is that it enables stations to use any idle slot, and for this reason it has a maximum throughput of 1 regardless of network size, number of connected stations or channel bandwidth. However, in its attempt to use every single slot on the bus, the protocol introduces unfairness in the sense that the bandwidth that a station can receive, or the delay its segments will encounter, strongly depends on its location on the bus. This unfairness of DQDB becomes more severe

when the end-to-end propagation delay, the network utilization, and the message size increase. For this reason a Bandwidth Balancing Mechanism has been recently proposed, and has been included in the standard, that can improve significantly the throughput fairness of the system.

2.2 Bandwidth Balancing Mechanism

The Bandwidth Balancing Mechanism (BBM_DQDB) is a modification of the basic DQDB MAC Mechanism [11]. It has the ability to provide the requested bandwidth by lightly loaded stations and evenly distribute the remaining bandwidth among the overloaded stations, regardless of their location on the bus. In DQDB a station can transmit a segment whenever its countdown counter (CD_CTR) is zero and the slot on the bus is idle. In the case of BBM_DQDB the station can transmit only on a fraction α of that time, where

$$\alpha = \frac{B}{1+B} \tag{2.1}$$

and B is the so called bandwidth balancing modulus.

For example, if the value of B=9, then $\alpha = 0.9$. In this case the station lets an extra empty slot to pass every time it transmits nine segments. This can be achieved by using an extra counter in every node called the Bandwidth Balancing counter (BBM_CTR). BBM_CTR is increased by one for every transmission of a segment. When it reaches the value of B, is decreased by B and the request counter (RQ_CTR) is increased by one. If the station is idle, the RQ_CTR will be decreased by one at the next empty slot that passes by on the forward bus. Otherwise, if the station has a segment to transmit, it will transfer the value of the RQ_CTR to the CD_CTR and will reset RQ_CTR to zero. The operation of CD_CTR is identical to that of the original DQDB protocol.

We see that according to this technique every B segments that a station transmits, it allows one empty slot to pass to the downstream stations. If an active downstream station with CD_CTR equal to zero sees this empty slot, it will transmit. It has been shown in [11] that in the case of overloaded stations, BBM_DQDB will eventually distribute fairly the channel bandwidth among the network stations. It has also been shown in [11] that the convergence speed towards the steady state, where the fair bandwidth allocation is achieved, depends strongly on the value of B. The smaller the value of B, the faster the convergence. However notice, that the above mechanism wastes some channel bandwidth which is given by the following equation.

$$Bandwidth\ loss = \frac{1}{1 + NB} \tag{2.2}$$

where N is the number of active stations.

Equation (2.2) shows that the smaller the value of B the more significant the bandwidth loss. Therefore we see that in the case of BBM_DQDB there is a trade off between convergence speed and bandwidth loss. A value of B between 8 and 10 is a reasonable compromise between the two and is being suggested in the standard. We also point out that BBM_DQDB can be used to distribute the available channel bandwidth in any arbitrary way among the competing stations by assigning different values of B to them [13]. For instance if B_i is the value of bandwidth balancing modulus assigned to station 'i', then in the case of overloaded stations the amount of bandwidth BW_i received by station 'i' will be:

$$BW_i = \frac{B_i}{1 + \sum_{i=1}^{N} B_i} \tag{2.3}$$

where N is the number of active and overloaded stations.

In the case where some stations are overloaded and other stations are under loaded BBM_DQDB will provide all the requested bandwidth to lightly loaded stations and evenly (or proportionally according to B_i) distribute the remaining bandwidth among the overloaded stations.

We emphasize that BBM_DQDB mainly provides a fair bandwidth allocation among overloaded stations. That is, it can not guarantee any level of bandwidth to a station since if another station becomes active it will receive a portion of the channel bandwidth. In addition, fair bandwidth allocation takes place in the steady state. Therefore if stations becomes active and inactive continuously its slow convergence to the steady state may render its presence ineffective. For these reasons another mechanism has been recently proposed that can provide guarantee throughput to certain stations supporting real time applications. The name of the proposed mechanism is Guaranteed Bandwidth mechanism [13] and we will discuss it in the sequel

2.3 Guaranteed Bandwidth Mechanism

The Guaranteed Bandwidth Mechanism (GBW_DQDB) provides guaranteed throughput and is going to be used by certain high priority stations that support applications which require a guaranteed amount of bandwidth. The rest of the stations will operate according to BBM_DQDB mechanism. The GBW_DQDB operation will guarantee a certain amount of bandwidth to a high priority station in the network regardless of whether the channel is heavily or lightly loaded. When the high priority station is not active, other stations (using BBM_DQDB) can use its idle bandwidth and improve their throughput and delay performance. When the high priority station becomes active again it will acquire its allocated bandwidth back. However, notice that if other stations are not active, the high priority station using GBW_DQDB can not use their idle bandwidth to improve even more, its delay performance.

The Guaranteed Bandwidth Mechanism (GBW_DQDB) is also very similar to

the DQDB and BBM_DQDB mechanisms, but instead of having a BBM counter (BBM_CTR), it uses a Credit counter (CR_CTR). Furthermore, its operation is based on three additional parameters: the segment cost (SGC), the credit maximum (CR-max) and the income per slot (INC) which have been assigned to the station.

According to GBW_DQDB mechanism, each station can only write on reserved slots. A station can send a request and reserve a slot only when it has accumulated enough income to pay for the slot. The cost of a slot (or segment) is provided by the value of SGC mentioned above. Each station accumulates income through the slots it observes on the reverse bus. That is, each station i has a credit counter CR_CTR_i which increases by INC_i whenever it observes a slot. The value of INC_i is determined by the amount of bandwidth that station i can reserve. For instance let us say that the channel bandwidth is 155 Mbps and we want to guarantee a bandwidth of 30 Mbps to station i. Then a straightforward way to do that is by selecting $INC_i = 30$ and SGC=155. In this way station i will increase CR_CTR_i by 30 for every slot that it observes on the reverse bus. When CR_CTR_i becomes greater than or equal to 155 and station i has a segment in its queue for which a request has not been sent, it will send a request and decrease the value of CR_CTR_i by 155. Notice that contrary to what happens in the case of DQDB or BBM_DQDB, where a station can send only one request at a time (i.e. a request can be sent only for the first segment in the queue), in the case of GBW_DQDB the station may transmit many requests and reserve many slots for the segments waiting in its queue. We also mention that in order to prevent CR_CTR_i to increase indefinitely during the periods station i is idle, a maximum value of credit CRmax is introduced. That is, if the value of CR_CTR_i exceeds CRmax station i will not continue to increase CR_CTR_i although it observes slots on the reverse bus. It is evident that the minimum value of CRmax that will provide the station with the guaranteed throughput is the one that satisfies the following inequality:

$$CR\max \geq \lceil \frac{SGC}{INC} \rceil * INC \tag{2.4}$$

Where $\lceil X \rceil$ is the smallest integer which is greater than or equal to X. Notice that higher values of CRmax than the one provided by (2.4) will not improve the bandwidth that a station can acquire. Simply they will slightly improve its delay performance.

A station can implement the GBW_DQDB operation by dividing its queue of segments into two parts. The Queue of Arrivals (QAR) part and the Transmission Queue (TQ) part. Whenever a message for transmission arrives at the station, it will be broken into appropriate number of segments and will be queued in QAR part of the station queue. At the same time the station will increase its CR_CTR by INC for every segment that observes on the reverse bus. When CR_CTR becomes equal to or greater than SGC and QAR> 0 the station will pass a segment in the TQ part, i.e. it will decrease QAR by 1 and will increase TQ by 1, it will send a request on the reverse bus, and decrease the value of CR_CTR by SGC. Therefore the TQ part contains the segments for which requests have been sent and the QAR part contains the segments for which requests have not been sent yet. The station is also provided with an array which holds the value with which the CD_CTR must be initialized whenever a segment becomes first in the TQ. That is, whenever a segment is transfered from QAR to TQ, because of accumulated income, the value of RQ_CTR is transfered to the appropriate location of this array. We have used the name PSAR (Paid Segments Array of Requests) for this array. Its i_{th} element contains the value with which CD_CTR must be initialized when the i_{th} segment of the TQ will become first in the TQ. Finally, the operation of CD_CTR and RQ_CTR is identical to that of the DQDB protocol.

GBW_DQDB guarantees a certain amount of bandwidth to each station that follows its operation. Consequently the delay performance also will not be signifi-

cantly affected by the network load, although the station location continues to have a minor effect. Whenever this station is not active other stations, following DQDB or BBM_DQDB, can acquire its bandwidth. However, the reverse is not possible. That is, if the rest of the stations are not active, a station following GBW_DQDB can not acquire their bandwidth and improve its own performance. For this reason in the next chapter we introduce a new mechanism, the Guaranteed Bandwidth and Balancing mechanism (GBBM), which can guarantee under overload conditions the required bandwidth to a high priority station. However, it enables a station in underload condition to utilize the bandwidth not used by the other stations.

CHAPTER 3

GUARANTEED BANDWIDTH AND BALANCING MECHANISM

In this chapter we introduce a new access mechanism that tries to combine the advantages of both BBM_DQDB and GBW_DQDB. We have used the name Guaranteed Bandwidth and Balancing Mechanism (GBBM) for this new mechanism.

We have seen that BBM_DQDB achieves fair bandwidth allocation, however, it converges slowly to the fair state. Therefore, it is not very appropriate for supporting real time traffic. GBW_DQDB on the other hand provides a guaranteed throughput to the high priority stations. However, it does not allow these stations to use the idle bandwidth that is available.

The objective of the Guaranteed Bandwidth and Balancing Mechanism (GBBM) is to provide a guaranteed bandwidth to the higher priority stations and enable them at the same time, to use the available idle bandwidth; evenly distributing this bandwidth among themselves. We also mention that whenever the high priority stations are idle, or they do not use all the bandwidth allocated to them, the lower priority stations can share this bandwidth among themselves.

The main advantage of GBBM is that it can reduce significantly the cost of the connection. It is evident that since the bandwidth provided by GBW_DQDB is guaranteed and no one else can use it, it will be expensive. Notice however that most sources of traffic alternate between an active state during which they generate a lot of traffic, and an idle state during which they do not generate any traffic at all. Furthermore, in most of the cases the duration of the active period is usually significantly lower than the duration of the idle period.

One approach for serving such a traffic source is to provide a guaranteed bandwidth to it equal to the bandwidth required during the active period. Since the source will use this bandwidth only during the active period, which is relatively small, the

cost of the connection will be high. This is actually the approach that GBW_DQDB takes. The other alternative will be to use statistical multiplexing. This approach is mainly based on the law of large numbers which states that if a large number of users compete for a channel then the instantaneous bandwidth demand will be very close to the average. For instance, consider an 400 Mbps channel and sources which generate traffic of 10 Mbps during the active period and no traffic during the idle period. Furthermore, let us assume that the active period is one fourth of the idle period, that is the average amount of traffic generated by each source is 2 Mbps. Then the law of large numbers indicates that if we accept 200 sources into the system the instantaneous bandwidth demand will be close to 400 Mbps, i.e. a bandwidth that the channel can provide. In practice due to statistical variations some times the bandwidth demand will exceed 400 Mbps and some times will be less than 400 Mbps. In the former case queues will start to build up inside the stations and the packet delay may become extremely high. For this reason the number of sources which must be allowed into the system must be kept to a significant lower level, so that the probability of the above event to be small. For instance a number of 100 sources may be appropriate in this case. Notice on the other hand that if we want to guarantee the 10 Mbps bandwidth to each source, then we can support only 40 of them. Therefore the cost of communications will be absorbed by 100 users in the first case and only 40 in the second. That is, the cost of the connection will be much higher when the bandwidth is guaranteed.

However, there are applications whose performance is very sensitive to throughput or delay and statistical multiplexing may not be appropriate for them. GBW_DQDB can be used in this case to guarantee them the required bandwidth at the expense of higher cost. The Guaranteed Bandwidth and Balancing Mechanism proposed here provides another alternative. It can guarantee part of the required bandwidth by these stations and allow them to compete for the rest. Such an approach

has the potential of providing the required bandwidth by these stations at a much lower cost.

Therefore, in the case of GBBM we can distinguish the segments inside a station in two types : a) those for which requests have been sent through income and b) those for which requests have been sent due to observed idle bandwidth. We have used the name *paid segments* for the first ones and *free segments* for the second ones.

3.1 GBBM Implementation

GBBM operates in both, the GBW_DQDB and BBM_DQDB modes. Each station has a queue of arriving segments (QAR), and a transmitting queue (TQ). In addition each station has a Credit Counter (CR_CTR), a Request Counter (RQ_CTR), a Countdown Counter (CD_CTR), a Free Segment Flag (FSFlag), a Free Segment First in Queue Flag (FSFQFlag), a BBM_CTR, a Free Segment Request Register (FSRR) and a Paid Segments Array of Requests (PSAR).

In Figure 3, we show the various components of the station whose operation control the transmission on the bus. These components are the following:

Queue of Arrivals (QAR): This is a part of the station queue that keeps all the segments for which a request has not been sent yet.

Transmission queue (TQ): Contains all the segments for which a requests has already been sent.

Request Queue: It contains all the requests that must be transmitted on the reverse bus. It is needed because a request may not be transmitted immediately on the reverse bus because the passing request bits may have already been set to 1 by other downstream stations.

Request Counter (RQ_CTR): Counts the requests sent for slot reservations by the downstream stations.

Count Down Counter (CD_CTR): Contains the number of empty slots that a

station must allow to pass before it transmits its first segment in queue.

Credit Counter (CR_CTR): Indicates the credit that has been accumulated by the station. CD_CTR increases its value by INC every time it observes a new slot passing on the reverse bus. The value of INC is determined by the amount of bandwidth which has been allocated to the station as well as by the value of SGC that has been decided for the segment (slot). For instance, if the channel bandwidth is 155 Mbps and we want to guarantee a 30 Mbps bandwidth to the station we can assign the value of 30 to INC and the value of 155 to SGC. Then every time the station observes a new slot on the reverse bus it will increase CR_CTR by 30. When CR_CTR becomes equal to or greater than 155, then the station has accumulated enough income to transmit a segment and if its QAR> 0 it will send an additional request to the request queue and decrease the value of CR_CTR by 155. In order to prevent CR_CTR for increasing indefinitely a maximum value of credit CRmax is also introduced here whose minimum value is provided by (2.4) in the previous section. For instance, in the case of INC=30 and SGC=155, the minimum value of CRmax will be $\lceil \frac{155}{30} \rceil \times 30 = 180$. Then after the station has observed 6 slots on the reverse bus its income will have become 180. If at this instant its QAR> 0, then the station will pass a segment to the TQ, decrease QAR by 1, and make CR_CTR= 180-155= 25.

Paid Segment Array of Requests (PSAR): This array provides for each segment which has joined the TQ, due to accumulated income, the value with which CD_CTR must be initialized, when it becomes first in queue. Every time a station has accumulated income, transfers a segment from QAR to TQ and this is the i_{th} segment with income in TQ, then the station also transfers the value of RQ_CTR to the PSAR(i) and resets RQ_CTR to zero. When a paid segment is transmitted and there are additional paid segments in the TQ, the station will transfer the value of PSAR(2) to CD_CTR and move all other values of PSAR by one position, i.e. PSAR(i) will

receive the value of PSAR(i+1).

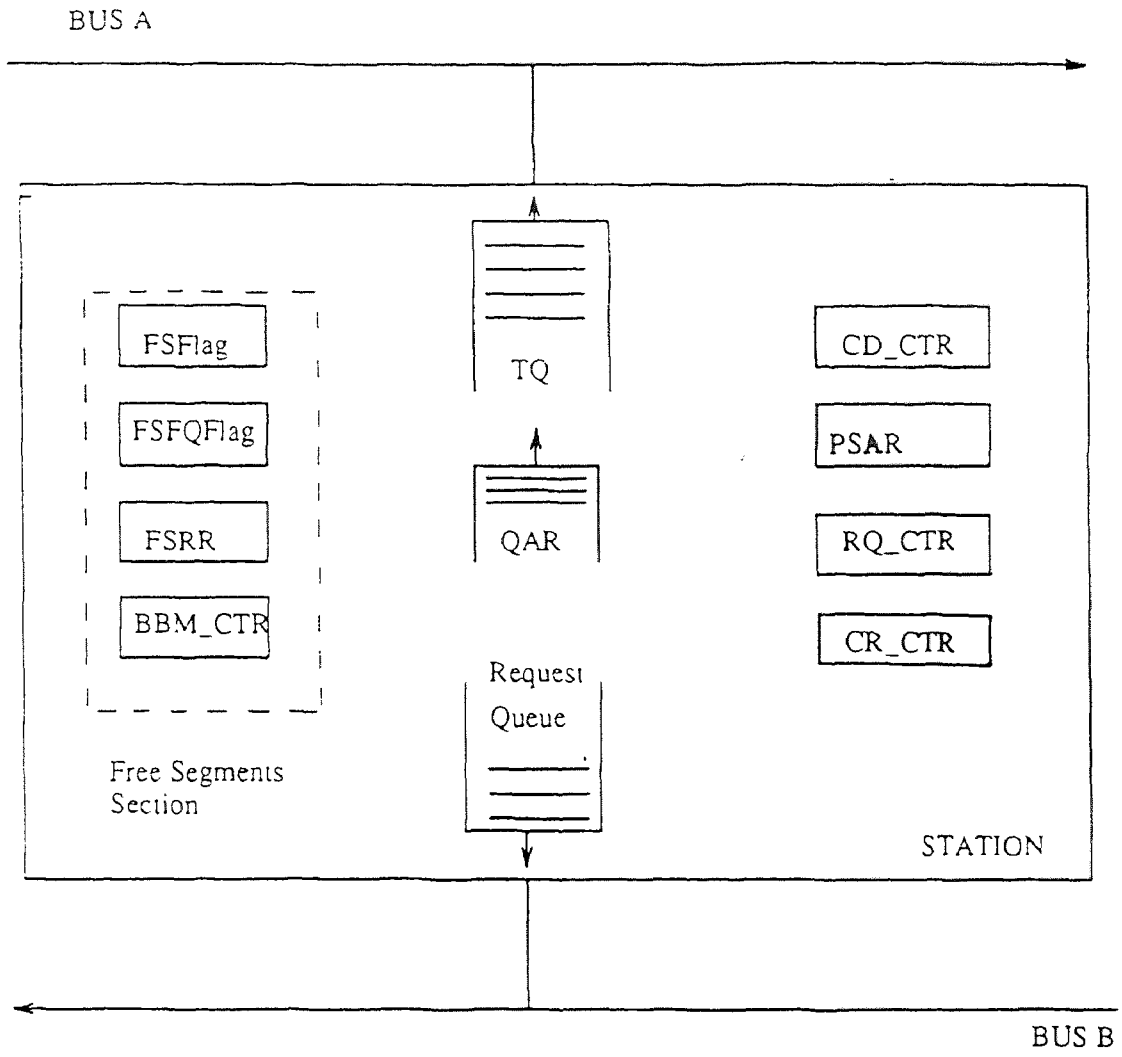

Figure 3 Internal structure of a station in the case of GBBM

Free Segment Flag (FSFlag): Indicates whether there is a free segment in the TQ. Initially, FSFlag is equal to 0. If the station has not accumulated enough income and QAR> 0, it will transfer a segment in the TQ, send a request in the Request queue and set FSFlag to 1. The free segment is always the last in the TQ in the sense that a free segment will be transmitted only if there are no paid segments in the queue. This does not mean that the segments are not transmitted in a first come first served order. If a paid segment has arrived after the free segment, the free segment will be transmitted first as a paid segment and the subsequent paid segment will be

considered as a free segment. In the case where there is only one segment in the TQ and FSFlag=1, which means that this is a free segment, the station will transmit it at the next empty slot and will set FSFlag=0. Now the station can pass another free segment (if it has one) in its transmission queue. We see that FSFlag ensures that the station will have one only free segment in its TQ.

Free Segment First in Queue Flag (FSFQFlag): Indicates whether a free segment has become the first in the TQ.

Bandwidth Balancing Counter (BBM_CTR): Counts the number of transmitted free segments by each station. It is increased by one, only when a free segment is transmitted onto the channel. Whenever the value of BBM_CTR becomes equal to B (BBM_CTR=B), the station increases the RQ_CTR by one and sets BBM_CTR=0.

Free Segment Request Register (FSRR): Indicates how many idle slots the station must allow to pass by before it transmits its free segment which has become first in the TQ. FSRR receives its value from RQ_CTR at the instant a free segment joins the TQ.

3.2 The GBBM Main Algorithm

In this section we describe two variations for the GBBM operation. The reason for introducing the second variation is because it improves the performance of GBBM. We first describe in detail the first variation indicated by GBBM1. Then, we provide the minor modification needed to produce the second variation, indicated by GBBM2. In the next chapter simulation results show that in certain load configurations the second version can significantly improve the performance. In the sequel we describe the main steps of the GBBM1 algorithm.

GBBM1 main algorithm:

In Figure 4 we show the main four steps of the GBBM1 algorithm.

1) In this step the station increases its CR_CTR by INC for each new slot

observed on the reverse bus. For instance, if the value of the income is 30, the CR_CTR will be increased by 30 for every slot that passes in front of the station. We point out here that CR_CTR increases by INC just before the slot has arrived at the station, so that the station can send a request on this slot if its income has exceeded SGC and its QAR> 0. If the value of CR_CTR is greater than CRmax the station will not increase its CR_CTR.

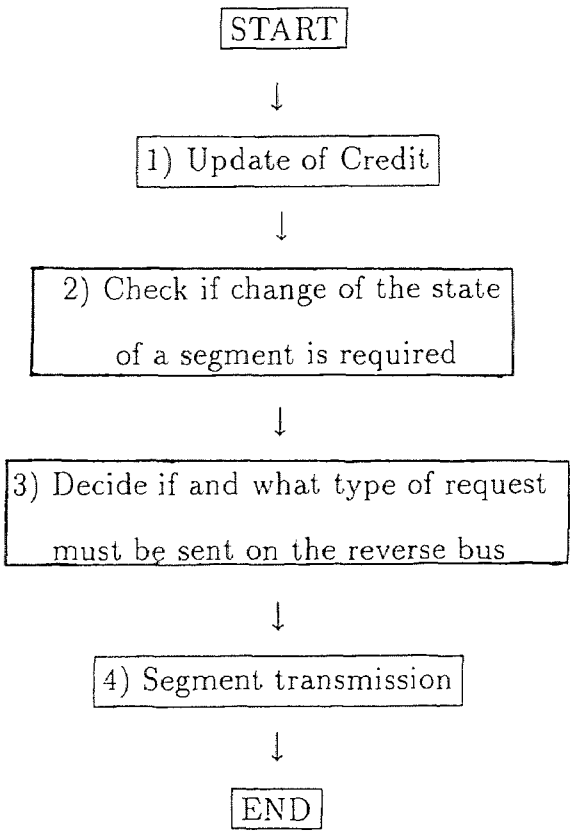

Figure 4: Main steps of GBBM algorithm

2) In this step the station checks whether a change of status for the first segment in the queue is required. If this segment is a paid segment, no change is required. This is also the case if this segment is a free segment (FSFlag=1, FSFQFlag=1) but CR_CTR<SGC. If however, this is a free segment but CR_CTR≥SGC, the station will change its status to a paid segment, i.e. FSFlag and FSFQFlag will become 0 and CR_CTR will decrease by SGC.

3) In this step the station decides whether a request must be sent on the request queue to the reverse bus. It first checks whether there is enough income accumulated to pay for a request. That is, if CR_CTR>SGC and QAR> 1 the station will decrease CR_CTR by SGC, increase TQ by 1, decrease QAR by 1, pass the value of RQ_CTR to PSAR(i) (if this is the i_{th} paid segment in TQ) and reset RQ_CTR to 0. Then the algorithm will move to step 4 which deals with the transmission. If however, CR_CTR<SGC, the station will check whether it can send a request for a free segment. This will happen if CR_CTR<SGC, FSFlag=0 and QAR> 0. In this case the station will decrease QAR by 1, increase TQ by 1, set FSFlag to 1, pass the content of RQ_CTR to FSRR and reset RQ_CTR to 0.

4) This step deals with the segment transmission. If the first segment in the queue is a free segment, (i.e. FSFQFlag=1), and CD_CTR=0 the station will transmit it, set both FSFQFlag and FSFlag to 0, and increase BBM_CTR by 1. If BBM_CTR becomes equal to B the station will increase RQ_CTR by 1 and reset BBM_CTR to 0. If on the other hand there is a paid segment in the queue (FSFQFlag=0) and the CD_CTR=0, the station will transmit it and will not increase the value of BBM_CTR. Then it will decide which must be the first segment in the TQ. If there are additional paid segments in the TQ, i.e. TQ$\geq$ 2, or TQ=1 and FSFlag=0, a paid segment will be first in queue. In this case the value of PSAR(2) will be transfered to CD_CTR and all other values in PSAR (if there any) will move up one position, i.e. the value of PSAR(i+1) will be transfered to PSAR(i). If there are no paid segments in the TQ but there is a free segment available, i.e. FSFlag=1 and TQ=1, a free segment will become first in queue. In this case the content of FSRR will be passed to CD_CTR and FSFQFlag will be set to 1. Finally, if the station does not have any segments, i.e. TQ=0, no action will be taken.

GBBM2 algorithm:

This variation is very similar to GBBM1 and was motivated by some simu-

lation results showing that in some cases the delay of some stations in the case of GBBM1 was higher than the corresponding delay of the same stations in the case of GBW_DQDB. We found out that the reason of this behavior was the following. Upstream stations, favored by their location on the bus, could transmit a significantly higher number of free segments than the other stations. Consequently the requests, mainly due to paid segments, of the downstream stations had to travel longer distances on the bus before they could reserve empty slots thus increasing their delays.

The objective of GBBM2 is to reduce the rate at which a station can transmit free segments. Since upstream stations have the ability (due to location) to transmit free segments at a higher rate, the imposed control will penalize them more and improve the fairness of the system. The only modification that is needed in GBBM1 in order to provide GBBM2 is the following. In step 4 of the algorithm, whenever the first segment in the queue is a free segment (i.e. FSFlag=1 and FSFQFlag=1) the station will transmit it, not only when the CD_CTR=0 but also when RQ_CTR=0. In this way the rate of free segments transmitted by upstream stations will be reduced significantly and the delay variation among stations will decrease.

CHAPTER 4

PERFORMANCE ANALYSIS

In this chapter we investigate the performance of both variations of GBBM mechanism and we compare them with the corresponding performance of BBM_DQDB and GBW_DQDB.

We consider a DQDB network with channel capacity of 155 Mbps and slot size equal to 53 bytes, i.e. the transmission time of a slot is equal to 2.735 μsec. The network consists of 10 stations with a distance between neighbor stations equal to 6 slots. We have assumed a signal propagation delay of 5 μsec$/km$ which makes the total length of the cable, from the first station to the last, equal to $6 \times 9 \times \frac{2.735}{5} = 29.54km$. The network connects two types of stations. High priority stations which support real-time traffic with certain throughput and delay requirements and low priority stations which support data transmission without any particular delay requirement. The low priority stations use BBM_DQDB whereas the high priority stations can use any of the BBM_DQDB, GBW_DQDB, GBBM1 and GBBM2.

In order to compare the effectiveness of GBW_DQDB and GBBM, we have considered the following traffic model for high priority traffic, which has also been used in [12]. The high priority traffic source alternates between an active period of fixed size and equal to 16 msec, and an idle period which is exponentially distributed with mean value of 16 msec. During the active period the traffic source generates fixed size messages of N_p segments at a fixed rate of one message every 85.11 μsec,i.e. during the active period 189 messages are generated. No messages are generated during the idle period. Notice that with the above values for the system parameters, the value of N_p determines the bandwidth required by the source during the active period. That is, if $N_p = 1$ the traffic source generates 5 Mbps, if $N_p = 6$ the traffic source generates 30 Mbps. Since no messages are generated during the idle period,

the average traffic generated by the source is half of the one generated during the active period i.e. in the above two cases 2.5 Mbps and 15 Mbps, respectively.

In Table 1 we examine how effective GBBM is in supporting the high priority source. We consider that among the 10 network stations only one supports high priority traffic. The value of N_p is 6 which indicates that the throughput requirements of the high priority source is 30 Mbps. In order to examine the effectiveness of GBBM under pessimistic conditions, we assume that all other stations are overloaded with data, i.e. they try to write on every idle and unreserved slot which is passing by. All these low priority stations use BBM_DQDB with value of B=6. In table 1 we show what will be the delay of the high priority station if it is the first station on the bus, if it is the second station on the bus and so on, i.e. in the last row of Table 1 if it is the last station on the bus. Our objective is not only to show whether GBBM can provide low delay but also low how this delay is affected by the location of the low priority station. For comparison, we have also included the corresponding delay of the high priority station if instead of GBBM it uses GBW_DQDB or BBM_DQDB. We point out that in order to provide a low delay for the high priority station we have guaranteed a throughput of 30 Mbps to it. That is, during the active period this station can acquire all the bandwidth it needs. The reason is that if we do not do that the high priority queue will build up during the active period and this will result to significant delays. In order to guarantee the 30 Mbps bandwidth to the high priority station we have selected in the case of GBW_DQDB and GBBM as value for segment cost (SGC) 155 and we have provided this station with an income (INC) of 30. In the case of BBM_DQDB we have assigned to it the minimum value of B which will allocate to it a bandwidth of 30 Mbps.This is the value of 14.

Table 1 clearly shows the superiority of GBBM. Both variations of GBBM provide lower delays than GBW_DQDB. The reason is that GBBM can write on the slots which are left unused, by the bandwidth balancing mechanism of the other overloaded

stations on the channel. We also see that BBM_DQDB provides significantly higher delays which demonstrates its limitations in supporting traffic with very stringent delay requirement. Finally Table 1 shows that GBBM1 and GBBM2 provide similar delays.

Table 1 Effect of high priority station location on performance.

Overloaded low priority stations with B=6.

$S_{hp} = 30Mbps, INC = 30, CR\max = 180, SGC = 155$

Packet size $N_p = 6, B = 14$

	Average message delay (μsec)			
StationIndex	*GBW_DQDB*	*BBM_DQDB*	*GBBM1*	*GBBM2*
0	88.4	415.5	47.0	50.9
1	88.9	262.9	45.2	45.7
2	89.6	171.9	50.7	50.2
3	90.6	154.5	56.4	51.3
4	91.3	152.6	52.3	47.3
5	92.3	177.9	62.9	57.0
6	93.2	228.5	64.7	58.5
7	94.6	356.9	67.4	62.1
8	97.3	589.1	78.1	76.2
9	96.4	836.6	78.5	78.5

In table 2 we consider the case where there are two high priority stations; each with a bandwidth requirement of 30 Mbps during its active period. Our objective is to investigate whether the location of the upstream one, will significantly affect the performance of the downstream. We see that GBW_DQDB demonstrates in this case the smallest delay variation between the two high priority stations, however, it is GBBM that provides the lower delays. Again BBM_DQDB provides significantly higher delays. The main reason of the higher delays in the case of BBM_DQDB in tables 1 and 2 is its slow convergence to the fair state which enables other stations, during the transmission period, to acquire bandwidth and increase its delays. In

addition, the slots whose operation wastes, have also a negative effect on its delay performance. We finally point out that in the case where the two high priority stations use BBM_DQDB, we must increase their values of B to 16 (from 14, in table 1) in order to be able to provide them with at least 30 Mbps during their active periods.

Table 2 Effect of location performance. Higher priority are stations 1 and 8. Low priority stations are overloaded with $B = 6$

$S_{hp1} = 30Mbps, S_{hp2} = 30Mbps, INC = 30, CR\max = 180, SGC = 155$

Packet size $N_p = 6, B = 16$

	Average message delay (μ sec)			
StationIndex	*GBW_DQDB*	*BBM_DQDB*	*GBBM1*	*GBBM2*
1	90.3	220.6	47.5	47.3
8	94.2	376.7	92.9	79.5

In Table 3 we consider the two high priority stations system of Table 2 and investigate the effect of the message size on performance. We keep the bandwidth requirement of the high priority stations at 30 Mbps but we increase the message size by a factor of 3 i.e. $N_p = 18$. In order to do that we have to reduce the number of message per active period from 189 to 63 and increase the message interarrival time from 85.11 μ sec to 255.3 μ sec . Table 3 shows that in all cases the average message delay has significantly increased. We also see that the delay of station 8 in the case of GBBM1 is significantly higher. The reason is the following. Station 1, because of its location can transmit free segments of a much higher rate and prevent station 8 from seeing idle slots early. As a result the delay of station 1 is small whereas the delay of 8 increases significantly.

With GBBM2, however, the rate at which station 1 can transmit free slots is significantly reduced, since free segments can be transmitted not only when CR_CTR

is equal to 0, but also when RQ_CTR=0. Therefore, downstream stations, and of course station 8 see earlier the empty slots and the delay of 8 significantly decreases; at the cost of a minor increase in the delay of station 1.

Table 3 Effect of location performance. Higher priority are stations 1 and 8. Low priority stations are overloaded with $B = 6$

$S_{hp1} = 30Mbps, S_{hp2} = 30Mbps, INC = 30, CR\max = 180, SGC = 155$

Packet size $N_p = 18, B = 16$

	Average message delay (μ sec)			
$StationIndex$	GBW_DQDB	BBM_DQDB	$GBBM1$	$GBBM2$
1	250.5	386.6	184.3	200.7
8	253.1	565.1	437.6	239.5

In all the previous tables we have chosen such values for income and segment cost in the case of GBW_DQDB and GBBM, and such values of B in the case of BBM_DQDB, that guarantee the requested bandwidth by the high priority stations during their active periods. In table 4 we investigate what happens when this is not the case. Therefore, we consider the two stations system of table 3, but we now allocate only 25 Mbps to each of stations 1 and 8 during the active period. We do that by using as income the value of 25 in the case of GBW_DQDB and GBBM and the value of B=12 in the case of BBM_DQDB. Table 4 shows that in this case the performance of GBW_DQDB drastically deteriorates. The reason is that the only way GBW_QDB can receive bandwidth is through reservations.Therefore, during the active periods its queue will build up significantly and then will start decreasing during the idle periods. This behavior will increase drastically the delay encountered by its messages. In the case of GBBM however, where the station can write on unreserved slots the performance only slightly is affected. We see again in table 4 the superior performance of GBBM2 over GBBM1.

Table 4 Effect of location performance. Higher priority are stations 1 and 8. Low priority stations are overloaded with $B = 6$

$S_{hp1} = 30Mbps, S_{hp2} = 30Mbps, INC = 25, CR\max = 175, SGC = 155$

Packet size $N_p = 18, B = 12$

	Average message delay (μ sec)			
StationIndex	*GBW_DQDB*	*BBM_DQDB*	*GBBM1*	*GBBM2*
1	2411.5	389.6	210.6	226.4
8	2250.7	568.1	476.4	246.4

In table 5 we consider the system of table 4 but with idle low priority stations. We see that the delay of stations 1 and 8 in the case of GBW_DQDB only slightly is affected. The reason is that GBW_DQDB can acquire bandwidth only through requests and can not take advantage of the ample bandwidth which is now available. In contrast all other schemes can use this bandwidth and reduce drastically their delays.

Table 5 Effect of location performance. High priority are stations 1 and 8. Low priority stations are idle.

$S_{hp1} = 30Mbps, S_{hp2} = 30Mbps, INC = 25, CR\max = 175, SGC = 155$

Packet size $N_p = 18$,B=12

	Average message delay (μ sec)			
StationIndex	*GBW_DQDB*	*BBM_DQDB*	*GBBM1*	*GBBM2*
1	2409.1	60.1	61.9	59.2
8	2301.9	61.2	61.4	60.7

In all previous cases we have considered all low priority stations to be overloaded and we have focused on the delay performance of the high priority stations. In the

next two tables we investigate the delay performance of all stations both of high and low priority.

Table 6 Effect of location performance. High priority are stations 1 and 8. Low priority stations have rate of 10.213 Mbps and packet size of 20 segments. $S_{hp1} = 30Mbps$, $S_{hp2} = 30Mbps, INC = 30, CR\max = 180, SGC = 155$

Packet size $N_p = 6$, B=16

	Average message delay (μ sec)			
StationIndex	*GBW_DQDB*	*BBM_DQDB*	*GBBM1*	*GBBM2*
0	285.5	271.1	278.6	273.9
1	95.2	60.1	44.3	42.5
2	282.1	272.7	279.9	274.8
3	279.9	278.9	287.6	274.3
4	279.6	275.8	279.4	276.1
5	275.7	278.1	279.9	271.4
6	282.4	286.5	287.0	283.3
7	299.3	289.4	303.0	298.4
8	100.5	81.3	68.0	58.3
9	349.9	336.9	349.6	335.9

In Table 6 the high priority stations 1 and 8 generate, during their active period, fixed sizes messages of 6 segments, i.e. each one of them generates 30 Mbps of traffic. These two stations are guaranteed their bandwidth by assigning to them an income of 30 in the case of GBW_DQDB and GBBM, and a value of B equal to 16 in the case of BBM_DQDB. The low priority stations generate fixed size messages of 20 segments according to a Poisson distribution. We have assumed that all of them generate the same amount of traffic and that the total offered load by them is 86% of the channel bandwidth not used by the two high priority stations, i.e. $155 - 30 \times 2 = 95$ Mbps. The low priority stations use the BBM_DQDB mechanism with value of B=6. In table 6 we show the average message delay for all stations. We see that the various mechanisms

provide similar delays to the low priority stations although the performance of the high priority stations is in the case of GBBM (and especially GBBM2) better. That is GBBM significantly improves the performance of the high priority stations without discriminating against the lower priority stations.

In table 6 we have considered that the total offered load by the low priority stations is 86% of the idle bandwidth when both high priority stations are active. In table 7 we investigate whether low priority stations can utilized some of the bandwidth which is not used by the high priority stations. Since the active and idle periods of the traffic sources which supported by the high priority stations are equal, on the average one of them will be active. Therefore, on the average, in addition to 95 Mbps, which are always available to low priority stations, there are also 30 Mbps available because only one of the active stations will be idle. We now investigate whether 50% of this 30 Mbps can be utilized by the low priority stations, i.e. we consider the system of table 6 but we now assume that the total offered load by the low priority station is 86% of (95+15) Mbps. Then the offered low by each low priority station will be 11.825 Mbps. Table 7 shows the corresponding delays of both high and low priority stations.

We see that the delays of low priority stations have significantly increased, however, they are still less than half msec. The delays of the high priority stations in the case of BBM_DQDB have also significantly increased since BBM_DQDB can only guarantee proportionality of bandwidth distribution among the competing stations. GBW_DQDB does not seem to be affected by the offered load of the low priority stations. This is expected since its bandwidth is guaranteed. Finally, the increase in the offered load by the low priority stations has an effect, although not significant, in the performance of the high priority stations in the case of GBBM. It mainly affects the transmission of their free segments and results to a minor increase in the delay. However, GBBM (and especially GBBM2) still provides the lowest delays.

Table 7 Effect of location performance. High priority are stations 1 and 8. Low priority stations have rate of 11.825 Mbps and packet size of 20 segments. $S_{hp1} = 30Mbps$, $S_{hp2} = 30Mbps, INC = 30, CR\max = 180, SGC = 155$

Packet size $N_p = 6$, B=16

	Average message delay (μ sec)			
$StationIndex$	GBW_DQDB	BBM_DQDB	$GBBM1$	$GBBM2$
0	461.8	420.2	454.9	452.1
1	94.3	122.4	47.2	45.1
2	467.6	436.3	453.8	488.0
3	468.7	430.7	454.6	456.1
4	462.4	470.5	462.2	467.5
5	465.1	455.1	469.5	471.5
6	492.9	457.7	484.6	485.4
7	522.1	486.4	533.3	534.6
8	99.1	112.1	90.1	69.4
9	595.5	565.1	632.6	605.9

CHAPTER 5

CONCLUSIONS

In this thesis we have first discussed the advantages and disadvantages of two access mechanisms that have been recently proposed for DQDB to address its fairness problems. The first of them, called BBM_DQDB can provide the requested bandwidth by lightly loaded stations and evenly for proportionally according to the values of B, distribute the remaining bandwidth among the overloaded stations. Its main problem is that it slowly converges to the steady state where fair bandwidth allocation is achieved. Consequently, it is not appropriate for supporting real-time traffic since transient overloads at low priority users may temporarily prohibit high priority users from accessing the channel, significantly increasing their delays, thus preventing them from meeting their stringent delay constraints. For this reason the GBW_DQDB mechanism was recently introduced to guarantee a certain amount of bandwidth to high priority users. However, the stations that use this mechanism can receive the requested bandwidth only through reservations. Consequently they have to reserve the amount of bandwidth which require during their active periods, although they will not use during their idle periods. As a result of this lost bandwidth, the cost of the connection will be high.

The above problems of BBM_DQDB and GBW_DQDB have motivated us to introduce a new mechanism that combines the advantages of the above two mechanisms. The proposed mechanism, called GBBM, can guarantee a certain amount of bandwidth to high priority stations and at the same time enable them to compete for the remaining channel bandwidth. As a result their performance can significantly improve. We have looked at two variations of GBBM. The motivation for the second variation,called GBBM2, was the strong effect that the location of the high priority stations could have or its delay performance in the case of the first variation, called

GBBM1. We have found that in the case of GBBM1, high priority stations located at the beginning of the bus can transmit a large number of free segments reducing significantly the rate at which other high priority stations, located downstream, can have access on to the channel and in this way drastically increasing their delay. According to GBBM2 a high priority station can transmit a free segment not only when its CD_CTR is 0 but also when its RQ_CTR is 0. Since upstream stations see the requests from all downstream stations their free segment transmission rate decreases. Downstream high priority stations see a much greater number of idle slots and their delay significantly decreases.

We have also compared the performance of the two variation of GBBM with BBM_DQDB and GBW_DQDB. We have found that the effect of the location of the high priority stations performance is minor in the case of GBW_DQDB, however, GBBM2 provides always significantly lower delays. The delays in the case of GBBM, are usually smaller than the corresponding delay in the case of GBW_DQDB, but always. In the case of GBBM1, if long messages are transmitted, then the delay of high priority stations which are located far away from the bus origin may be significantly higher than the corresponding delay in the case of GBW_DQDB. BBM_DQDB on the other hand, provides underloaded low priority stations, the highest delays. This behavior clearly demonstrates its limitations in terms of satisfying the stringent delay requirements of real-time traffic. We have finally seen that if high priority stations using GBW_DQDB are not provided with the bandwidth required during their active periods, their delays will drastically increase. In contrast, the effect of lower than required guaranteed bandwidth in the case of the other mechanisms will not be that significant. This behavior demonstrates the ability of GBBM (and especially GBBM2) to support traffic with certain throughput or delay requirements without the need of guaranteeing all the requested bandwidth. Since guaranteed bandwidth is expected to be expensive, GBBM2 has the potential of supporting real-time applications as a

much lower cost GBW_DQDB.

We have seen that GBBM combines the advantages of BBM_DQDB and GBW_DQDB. Recently, a new Bandwidth Balancing Mechanism, called the No Slot Waisting Bandwidth Balancing (NSW_BWB) mechanism, has been introduced in [14,15]. The main advantage of NSW_BWB is that it can introduce a similar to BWB_DQDB fairness into DQDB network but without waisting channel slots. This property enables NSW_BWB to converge very fast to the steady state where the fair bandwidth is achieved. Therefore, it will be very interesting, as a future research, to investigate for a mechanism that combines the advantages of NSW_BWB and GBW_DQDB, as well to compare its performance with the corresponding performance of GBBM.

APPENDIX A

FORMULA DERIVATION

We have the following assumptions for performance analysis.

Channel capacity: C=155.5 [$Mbps$]

Number of nodes: N=20

The bus length is D [km] and nodes locates at every D/N km.

The signal propagation delay between nodes : $t_{prop.}$

The signal propagation delay : $V_{prop.}$

$l_{slot} = 53\ bytes$

$t_{slot} = \frac{l_{slot}}{C}$

$t_{prop.} = V_{prop.} \times D$

The frame arrivals follows Poisson distribution.

A frame has a constant length L_p bits.

Arrival time of a message is:

$P(x > t) = \exp^{-\lambda t}$

$P(x \leq t) = F_x(t) = 1 - \exp^{-\lambda t}$

$\exp^{-\lambda t} = 1 - F_x(t)$ where $F_x(t)$ is a unit function (u)

$-\lambda t = \log(1-u)$

$\Rightarrow \boxed{t = -\frac{\log(1-u)}{\lambda}}$

APPENDIX B

ARRAY DESCRIPTION

THE FOLLOWING ARRAYS ARE USED IN THE SIMULATION:

STASTATUS(0:9,4):

stastatus(i,1):CD-CTR

stastatus(i,2):RQ-CTR

stastatus(i,3)= 0 : station isidle

stastatus(i,3)> 0 : station is active

SLOTSTATUS(200,2,3):

200: total number of slots(i) on both buses

2: two buses , bus A , bus B

3: Busy Bit , Request Bit , Station Number

slotstatus(i,1,1): Busy Bit(BB) on bus A

slotstatus(i,2,1): BB on bus B

slotstatus(i,1,2): Request Bit(RB) on bus A

slotstatus(i,2,2): RB on bus B

slotstatus(i,1,3): which slot on bus A is over that station

slotstatus(i,2,3): which slot on bus B is over that station

DELAY(4,0:9):

0:9: is the number of stations

delay(1,i): waiting time of segments

delay(2,i): number of transmitted segments

delay(3,i): waiting time of messages

delay(4,i): number of transmitted messages

ARRIVALS(10,2):

10: number of stations (i)

arrivals(i,1): arrival time of the message

arrivals(i,2): message size

COUNTER(0:9): Provides the size of Request Queue at each station.

BBM-CTR(0:9): Bandwidth Balancing Counter for each station.

CR-CTR:(0:9): Credit Counter for each station.

rmean(0:9): Is the number of segments in each message.

Sgmq(0:9): Provides the Transmission Queue (TQ) for each station.

Sgmw(0:9): Provides the size of QAR for each station.

kval(0:9): Is the value of B in each station.

Tflag(0:9): FSFlag for each station.

Rgflag(0:9):FSFQFlag for each station.

APPENDIX C

BBM_DQDB Simulation Program

```
c ***********************************************
c  PROGRAM SIMULATION
c  All stations use BWB mechanism
c ***********************************************

      COMMON arrivals,nuofslots,coverage,DELAY,RMEAN
      REAL   ARRIVALS(10,3,200),DELAY(4,0:9),rrate(0:9),
     1 coverage,RMEAN(0:9),RPROPAG
         integer nss,inc,ncrmx,nmean

         ICSEED=135
         ITSEED=479
      CALL RANSET(ICSEED,ITSEED)

      nss=0
      inc=30
      ncrmx=180
      nmean=6

         do 2 i=0,9
         if (i.eq.0) rmean(i)=1000
         if (i.eq.1) rmean(i)=1000
         if (i.eq.2) rmean(i)=1000
         if (i.eq.3) rmean(i)=1000
         if (i.eq.4) rmean(i)=1000
         if (i.eq.5) rmean(i)=1000
         if (i.eq.6) rmean(i)=1000
         if (i.eq.7) rmean(i)=1000
         if (i.eq.8) rmean(i)=1000
         if (i.eq.9) rmean(i)=1000
         if (i.eq.nss) rmean(i)=nmean
 2       continue

         do 5 i=0,9
      if (i.eq.0) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.1) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.2) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.3) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.4) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.5) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.6) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.7) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.8) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.9) rrate(i)=1000.7/(rmean(i)*424)
         if (i.eq.nss) rrate(i)=1/16000.0
      do 15 j=1,200
         arrivals(i+1,3,j)=0
         arrivals(i+1,2,j)=0
         arrivals(i+1,1,j)=0
15     continue
 5       continue
```

```
   DO 10 I=0,9
         If (i.eq.nss) then
          Arrivals(i+1,2,1)=rmean(i)
          arrivals(i+1,3,1)=189
       ARRIVALS(I+1,1,2)=ARRIVALS(I+1,1,1)+188*84.5+
    1    (-1)*ALOG(1-UNI(K))/(RRATE(i))
        else
         arrivals(i+1,2,1)=rmean(i)
        endif
c       print*, ARRIVALS(I+1,1,1),ARRIVALS(I+1,2,1)
 10   CONTINUE

c      PRINT*,'Give the slot coverage(number of stations):'
c      READ*,coverage
          coverage=6.0

C   RPROPAG is the propagation from station to station exprassed
C   in microseconds  (propagation speed = 5 10-6 sec/km)
        RPROPAG=2.726
      print*,'PROPAGATION TIME:',RPROPAG
C   Capacity = 155.520 Mbs, rslottime is slottime in microseconds
c       print*,arrivals
      nuofslots=int(9*coverage)+1
      print*,'NUMBER OF SLOTS : ',nuofslots
      print*,'special st=0 with BWB , rest BWB'
      print*,'nss=',nss,'  nmean=',nmean
      CALL EXECUTE(RPROPAG,RRATE,nss,inc,ncrmx)
      do 22 i=0,9
       if (i .eq. nss) then
        print*,i,delay(1,i),delay(2,i),delay(1,i)/delay(2,i)
        print*,i,delay(3,i),delay(4,i),delay(3,i)/delay(4,i)
       else
        print*,i,delay(2,i)
       end if
 22   continue
      END


 ************************************************************
      SUBROUTINE EXECUTE(RINCREMENT,RRATE,nss,inc,ncrmx)

      common table,nuofslots,coverage,DELAY,rmean
      REAL TABLE(10,3,200),RINCREMENT,DELAY(4,0:9),coverage,
     1  SLOTSTATUS(200,2,3),rmean(0:9)
      DOUBLE PRECISION TIME
      INTEGER STASTATUS(0:9,4),STAT(160),IFLAG(160),
     1   kval(0:9),credit(0:9),iter,nss,inc,ncrmx,
     1   sgmq(0:9),sgmw(0:9),tflag(0:9)
      NOFTRANS=0
      TIME=0
      CALL INITIALIZE(SLOTSTATUS)
```

```
      do 5 i=0,9
         if (i.eq.0) kval(i)=6
            if (i.eq.1) kval(i)=6
         if (i.eq.2) kval(i)=6
            if (i.eq.3) kval(i)=6
            if (i.eq.4) kval(i)=6
            if (i.eq.5) kval(i)=6
            if (i.eq.6) kval(i)=6
            if (i.eq.7) kval(i)=6
            If (i.eq.8) kval(i)=6
            If (i.eq.9) kval(i)=6
            if (i.eq.nss) kval(i)=14
 5    continue
      DO 20 I=0,9
       STASTATUS(I,1)=0
       STASTATUS(I,2)=0
       STASTATUS(I,3)=0
       STASTATUS(I,4)=-1
       sgmw(i)=0
c      if((i .eq. nss) .or. (i .eq. 3) .or. (i .eq. 8))  then
       sgmw(i)=rmean(i)
       STASTATUS(i,1)=STASTATUS(i,2)
c      end if
       sgmq(i)=0
       tflag(i)=0
 20    CONTINUE
 10    DO 30 sl=1,nuofslots,coverage
       i=sl
       STAT(I)=int(SLOTSTATUS(I,1,3)+0.05)
 30    CONTINUE
      CALL NEXTSTEP(SLOTSTATUS,STASTATUS,IFLAG,KVAL,credit,sgmw,
     1          sgmq,tflag,nss,inc,ncrmx)
      CALL MODIFY(IFLAG,STAT,TIME,STASTATUS,RRATE,sgmw,sgmq,nss)

       DO 40 sl=1,nuofslots,coverage
       i=sl
       IF (IFLAG(I).EQ.1) NOFTRANS=NOFTRANS+1
 40    CONTINUE
      iter=iter+1
      TIME=TIME+RINCREMENT
      DO 60 I=1,nuofslots
      SLOTSTATUS(I,1,3)=SLOTSTATUS(I,1,3)+(1/coverage)
      SLOTSTATUS(I,2,3)=SLOTSTATUS(I,2,3)-(1/coverage)
 60    CONTINUE

      if (mod(noftrans,1000000).eq.0)  print*,noftrans,time,
     1                    delay(1,0)/delay(2,0)
      IF (NOFTRANS.LT.4000000) GOTO 10
      print*,'TOTAL TIME IS EQUAL TO:',TIME
      RETURN
      END
```

```
C ***************************************************************

      SUBROUTINE INITIALIZE(SLOTSTATUS)


      common arrivals,nuofslots,coverage,DELAY

      REAL DELAY(4,0:9),coverage,SLOTSTATUS(200,2,3),
     1     arrivals(10,3,200)

      DO 10 I=nuofslots,1,-1
       DO 20 J=1,2
        SLOTSTATUS(I,1,J)=0
        SLOTSTATUS(I,2,J)=0
 20    CONTINUE
        SLOTSTATUS(I,1,3)=(i-1)*(1/coverage)
        SLOTSTATUS(I,2,3)=9-((i-1)*(1/coverage))
 10    CONTINUE
      RETURN
      END


*************************************************************
      SUBROUTINE GENERATESLOT(SLOTSTATUS)


      common arrivals,nuofslots
      REAL SLOTSTATUS(200,2,3),arrivals(10,3,200)

      DO 10 I=nuofslots-1,1,-1
        DO 20 J=1,3
         SLOTSTATUS(I+1,1,J)=SLOTSTATUS(I,1,J)
         SLOTSTATUS(I+1,2,J)=SLOTSTATUS(I,2,J)
20       CONTINUE
10    CONTINUE
      SLOTSTATUS(1,1,1)=0
      SLOTSTATUS(1,1,2)=0
      SLOTSTATUS(1,1,3)=0
      SLOTSTATUS(1,2,1)=0
      SLOTSTATUS(1,2,2)=0
      SLOTSTATUS(1,2,3)=9
      RETURN
      END
```

```
C*************************************************************
      SUBROUTINE NEXTSTEP(SLOTSTATUS,STASTATUS,ISFLAG,KVAL,credit,
     1                    sgmw,sgmq,tflag,nss,inc,ncrmx)
      save counters,bbm,nrqctr
      common arrivals,nuofslots,coverage,DELAY
      INTEGER STASTATUS(0:9,4),ISFLAG(160),STATIONA(160),nss,
     1  STATIONB(160),counters(0:9),kval(0:9),bbm(0:9),inc,ncrmx,
     1  credit(0:9),sgmw(0:9),sgmq(0:9),tflag(0:9),nrqctr(300)
      REAL SLOTSTATUS(200,2,3),DELAY(4,0:9),COVERAGE,
     1    arrivals(10,3,200)


      DO 10 sl=1,nuofslots,coverage
       i=sl
       ISFLAG(I)=0
       STATIONA(I)=int(SLOTSTATUS(I,1,3)+0.05)
       STATIONB(I)=int(SLOTSTATUS(nuofslots-i+1,2,3)+0.05)
10    CONTINUE


      DO 20 sl=1,nuofslots,coverage
       i=sl
          j=nuofslots-i+1

      if (sgmw(stationb(i)).gt.0) then

        if (tflag(stationb(i)).eq.0) then
         counters(stationb(i))=counters(stationb(i))+1
            sgmw(stationb(i))=sgmw(stationb(i))-1
            sgmq(stationb(i))=sgmq(stationb(i))+1
         stastatus(stationb(i),1)=stastatus(stationb(i),2)
         stastatus(stationb(i),2)=0
            tflag(stationb(i))=1
        endif
      endif

c change in RQ_CTR is introduced here because station knows, before it sees the
c  next slot whether it should transfer the content of RQ_CTR to CD_CTR

      STASTATUS(STATIONB(I),2)=
     1 STASTATUS(STATIONB(I),2)+SLOTSTATUS(J,2,2)

      IF ((SLOTSTATUS(J,2,2).EQ.0).AND.
     1   (counters(stationb(I)).ge.1)) THEN
         SLOTSTATUS(J,2,2)=1
         counters(stationb(I))=counters(stationb(I))-1
      ENDIF
       IF (sgmq(STATIONA(I)).EQ.0) THEN
         IF (STASTATUS(STATIONA(I),2).GT.0) THEN
           STASTATUS(STATIONA(I),2)=
     1      STASTATUS(STATIONA(I),2)+SLOTSTATUS(I,1,1)-1
           end if
       end if
```

```
      IF (sgmq(STATIONA(I)).gt.0) THEN
       IF ((SLOTSTATUS(I,1,1).EQ.0).AND.
     1       (STASTATUS(STATIONA(I),1).EQ.0)) then
         ISFLAG(I)=1
         SLOTSTATUS(I,1,1)=1
            sgmq(stationa(i))=sgmq(stationa(i))-1
            bbm(stationa(i))=bbm(stationa(i))+1
            tflag(stationa(i))=0
            if(bbm(stationa(i)).eq.KVAL(stationa(i))) then
          STASTATUS(STATIONA(I),2)=STASTATUS(STATIONA(I),2)+1
             bbm(stationa(i))=0
         end if

       ELSE

        IF (STASTATUS(STATIONA(I),1).GT.0) THEN
         STASTATUS(STATIONA(I),1)=STASTATUS(STATIONA(I),1)+
     1    SLOTSTATUS(I,1,1)-1
        ENDIF
       ENDIF
      ENDIF

 20    CONTINUE
      RETURN
      END

C *****************************************************************

      SUBROUTINE MODIFY(FLAG,ST,CURRTIM,STASTATUS,LAMDA,sgmw,
     1   sgmq,nss)


      COMMON arrivals,nuofslots,COVERAGE,DELAY,RMEAN

      INTEGER FLAG(160),STASTATUS(0:9,4),ST(160)
      integer sgmw(0:9),sgmq(0:9),nss
      REAL ARRIVALS(10,3,200),DELAY(4,0:9),LAMDA(0:9),
     1        COVERAGE,RMEAN(0:9)
      DOUBLE PRECISION CURRTIM
      DO 30 sl=1,nuofslots,coverage
       i=sl

      IF (FLAG(I).EQ.1) THEN

        DELAY(2,ST(I))=DELAY(2,ST(I))+1
       if (st(i).eq.nss) then
        DELAY(1,ST(I))=DELAY(1,ST(I))+
     1        CURRTIM-ARRIVALS(ST(I)+1,1,1)+2.726
       end if
```

```
      .
     IF (ARRIVALS(ST(I)+1,2,1).EQ.1) THEN
       if (st(i).eq.nss) then
          DELAY(3,ST(I))=DELAY(3,ST(I))+
     1         CURRTIM-ARRIVALS(ST(I)+1,1,1)+2.726
        end if
          DELAY(4,ST(I))=DELAY(4,ST(I))+1
        if(st(i).ne.nss) then
c         ARRIVALS(ST(I)+1,1,1)=ARRIVALS(ST(I)+1,1,1)+
c    1              (-1)*ALOG(1-UNI(K))/(LAMDA(st(i)))
c         arrivals(st(I)+1,2,1)=rmean(st(i))
c
c       else if( (st(i).eq.3) .or. (st(i).eq.8)) then
         sgmw(st(i))=1000
        else
            if(st(i).eq.nss) then
          If(arrivals(st(i)+1,3,1).eq.1) then
              arrivals(st(i)+1,2,1)=rmean(st(i))
              arrivals(st(i)+1,3,1)=189
              arrivals(st(i)+1,1,1)=arrivals(st(i)+1,1,2)
              arrivals(st(i)+1,1,2)=arrivals(st(i)+1,1,1)+188*84.5+
     1              (-1)*ALOG(1-UNI(K))/(lamda(st(i)))
         else
              arrivals(st(i)+1,2,1)=rmean(st(i))
              arrivals(st(i)+1,1,1)=arrivals(st(i)+1,1,1)+84.5
              arrivals(st(i)+1,3,1)=arrivals(st(i)+1,3,1)-1
            endif
        end if
       end if
       ELSE
        ARRIVALS(ST(I)+1,2,1)=ARRIVALS(ST(I)+1,2,1)-1
       END IF
      END IF

      IF (st(i).eq.nss) then
       if(arrivals(st(I)+1,1,1) .lt. (CURRTIM+2.726)) then
        sgmw(st(i))=arrivals(st(I)+1,2,1)-sgmq(st(i))
       else
        sgmw(st(i))=0
       end if
      ELSE
         sgmw(st(i))=1000
         END IF
30      continue

     RETURN
     END
*****************************************************************
```

APPENDIX D

GBW_DQDB Simulation Program

```
**************************************************************
c  PROGRAM SIMULATION
c  Special station transmits only through income.
c  The rest use BWB mechanism
**************************************************************

      COMMON arrivals,nuofslots,coverage,DELAY,RMEAN
      REAL ARRIVALS(10,3,200),DELAY(4,0:9),rrate(0:9),
     1 coverage,RMEAN(0:9),rpropag
          integer nss,inc,ncrmx,nmean

          ICSEED=135
          ITSEED=479
      CALL RANSET(ICSEED,ITSEED)

      nss=0
      inc=30
      ncrmx=180
      nmean=6

          do 2 i=0,9
          if (i.eq.0) rmean(i)=1000
          if (i.eq.1) rmean(i)=1000
          if (i.eq.2) rmean(i)=1000
          if (i.eq.3) rmean(i)=1000
          if (i.eq.4) rmean(i)=1000
          if (i.eq.5) rmean(i)=1000
          if (i.eq.6) rmean(i)=1000
          if (i.eq.7) rmean(i)=1000
          if (i.eq.8) rmean(i)=1000
          if (i.eq.9) rmean(i)=1000
          if (i.eq.nss) rmean(i)=nmean
 2        continue

          do 5 i=0,9
      if (i.eq.0) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.1) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.2) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.3) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.4) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.5) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.6) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.7) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.8) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.9) rrate(i)=1000.7/(rmean(i)*424)
          if (i.eq.nss) rrate(i)=1/16000.0
      do 15 j=1,200
          arrivals(i+1,3,j)=0
          arrivals(i+1,2,j)=0
          arrivals(i+1,1,j)=0
15     continue
 5        continue
```

```
      DO 10 I=0,9
         If (i.eq.nss) then
          Arrivals(i+1,2,1)=rmean(i)
          arrivals(i+1,3,1)=189
       ARRIVALS(I+1,1,2)=ARRIVALS(I+1,1,1)+188*84.5+
     1    (-1)*ALOG(1-UNI(K))/(RRATE(i))
       else
        arrivals(i+1,2,1)=rmean(i)
       endif
c      print*, ARRIVALS(I+1,1,1),ARRIVALS(I+1,2,1)
  10   CONTINUE

c      PRINT*,'Give the slot coverage(number of stations):'
c      READ*,coverage
          coverage=6.0

C   RPROPAG is the propagation from station to station exprassed
C   in microseconds  (propagation speed = 5 10-6 sec/km)
        RPROPAG=2.726
       print*,'PROPAGATION TIME:',RPROPAG
C   Capacity = 155.520 Mbs, rslottime is slottime in microseconds
c       print*,arrivals
       nuofslots=int(9*coverage)+1
       print*,'NUMBER OF SLOTS : ',nuofslots  .
       print*,'special st=0 with INC , rest BWB.'
       print*,'nss=',nss,'  INC=',inc,'  CRmax=',ncrmx,
     1                                 '  nmean=',nmean
       CALL EXECUTE(RPROPAG,RRATE,nss,inc,ncrmx)
       do 22 i=0,9
        if (i .eq. nss) then
         print*,i,delay(1,i),delay(2,i),delay(1,i)/delay(2,i)
         print*,i,delay(3,i),delay(4,i),delay(3,i)/delay(4,i)
        else
         print*,i,delay(2,i)
        end if
  22   continue
       END

C  ************************************************************
       SUBROUTINE EXECUTE(RINCREMENT,RRATE,nss,inc,ncrmx)

       common table,nuofslots,coverage,DELAY,rmean
       REAL TABLE(10,3,200),RINCREMENT,DELAY(4,0:9),coverage,
     1  SLOTSTATUS(200,2,3),rmean(0:9)
       DOUBLE PRECISION TIME
       INTEGER STASTATUS(0:9,4),STAT(160),IFLAG(160),
     1   kval(0:9),credit(0:9),iter,nss,inc,ncrmx,
     1   sgmq(0:9),sgmw(0:9),tflag(0:9)
       NOFTRANS=0
       TIME=0
       CALL INITIALIZE(SLOTSTATUS)
```

```
      dó 5 i=0,9
      if (i.eq.0) kval(i)=6
         if (i.eq.1) kval(i)=6
         if (i.eq.2) kval(i)=6
         if (i.eq.3) kval(i)=6
         if (i.eq.4) kval(i)=6
         if (i.eq.5) kval(i)=6
         if (i.eq.6) kval(i)=6
         if (i.eq.7) kval(i)=6
         If (i.eq.8) kval(i)=6
         If (i.eq.9) kval(i)=6
         if (i.eq.nss) kval(i)=14
 5    continue
      DO 20 I=0,9
       STASTATUS(I,1)=0
       STASTATUS(I,2)=0
       STASTATUS(I,3)=0
       STASTATUS(I,4)=-1
       sgmw(i)=0
c      if((i .eq. nss) .or. (i .eq. 3) .or. (i .eq. 8))  then
       sgmw(i)=rmean(i)
       STASTATUS(i,1)=STASTATUS(i,2)
c      end if
       sgmq(i)=0
       tflag(i)=0
 20    CONTINUE
 10    DO 30 sl=1,nuofslots,coverage
       i=sl
       STAT(I)=int(SLOTSTATUS(I,1,3)+0.05)

 30    CONTINUE
       CALL NEXTSTEP(SLOTSTATUS,STASTATUS,IFLAG,KVAL,credit,sgmw,
     1          sgmq,tflag,nss,inc,ncrmx)
       CALL MODIFY(IFLAG,STAT,TIME,STASTATUS,RRATE,sgmw,sgmq,nss)

       DO 40 sl=1,nuofslots,coverage
        i=sl
        IF (IFLAG(I).EQ.1) NOFTRANS=NOFTRANS+1
 40    CONTINUE
       iter=iter+1
       TIME=TIME+RINCREMENT
       DO 60 I=1,nuofslots
       SLOTSTATUS(I,1,3)=SLOTSTATUS(I,1,3)+(1/coverage)
       SLOTSTATUS(I,2,3)=SLOTSTATUS(I,2,3)-(1/coverage)

 60    CONTINUE
       CALL GENERATESLOT(SLOTSTATUS)

       if (mod(noftrans,1000000).eq.0)  print*,noftrans,time,
     1                    delay(1,0)/delay(2,0)
       IF (NOFTRANS.LT.4000000) GOTO 10
       print*,'TOTAL TIME IS EQUAL TO:',TIME
       RETURN
       END
```

```
C ***************************************************************

      SUBROUTINE INITIALIZE(SLOTSTATUS)


      common arrivals,nuofslots,coverage,DELAY

      REAL DELAY(4,0:9),coverage,SLOTSTATUS(200,2,3),
     1     arrivals(10,3,200)

      DO 10 I=nuofslots,1,-1
       DO 20 J=1,2
        SLOTSTATUS(I,1,J)=0
        SLOTSTATUS(I,2,J)=0
 20    CONTINUE
        SLOTSTATUS(I,1,3)=(i-1)*(1/coverage)
        SLOTSTATUS(I,2,3)=9-((i-1)*(1/coverage))
 10    CONTINUE
      RETURN
      END

C ***************************************************************
      SUBROUTINE GENERATESLOT(SLOTSTATUS)


      common arrivals,nuofslots
      REAL SLOTSTATUS(200,2,3),arrivals(10,3,200)

      DO 10 I=nuofslots-1,1,-1
        DO 20 J=1,3
        SLOTSTATUS(I+1,1,J)=SLOTSTATUS(I,1,J)
        SLOTSTATUS(I+1,2,J)=SLOTSTATUS(I,2,J)
 20     CONTINUE
 10    CONTINUE
      SLOTSTATUS(1,1,1)=0
      SLOTSTATUS(1,1,2)=0
      SLOTSTATUS(1,1,3)=0
      SLOTSTATUS(1,2,1)=0
      SLOTSTATUS(1,2,2)=0
      SLOTSTATUS(1,2,3)=9
      RETURN
      END
C ***************************************************************

      SUBROUTINE NEXTSTEP(SLOTSTATUS,STASTATUS,ISFLAG,KVAL,credit,
     1                    sgmw,sgmq,tflag,nss,inc,ncrmx)

      save counters,bbm,nrqctr
      common arrivals,nuofslots,coverage,DELAY
      INTEGER STASTATUS(0:9,4),ISFLAG(160),STATIONA(160),nss,
     1  STATIONB(160),counters(0:9),kval(0:9),bbm(0:9),inc,ncrmx,
     1  credit(0:9),sgmw(0:9),sgmq(0:9),tflag(0:9),nrqctr(300)
      REAL SLOTSTATUS(200,2,3),DELAY(4,0:9),COVERAGE,
```

```
     1   ' arrivals(10,3,200)


         DO 10 sl=1,nuofslots,coverage
          i=sl
          ISFLAG(I)=0
          STATIONA(I)=int(SLOTSTATUS(I,1,3)+0.05)
          STATIONB(I)=int(SLOTSTATUS(nuofslots-i+1,2,3)+0.05)
    10    CONTINUE

         DO 20 sl=1,nuofslots,coverage
          i=sl
             j=nuofslots-i+1
    c       print*,stationa(i),stationb(i)
    c       read*,iiik

         if (stationb(i).eq.nss) then
          if(credit(stationb(i)) .lt. ncrmx)
       1     credit(stationb(i))=credit(stationb(i))+inc
    c     credit(stationb(i))=min(credit(stationb(i))+inc,ncrmx)
          endif

         if (sgmw(stationb(i)).gt.0) then
          if (stationb(i).eq.nss) then
              if (credit(stationb(i)).ge.155) then
            credit(stationb(i))=credit(stationb(i))-155
            counters(stationb(i))=counters(stationb(i))+1
            sgmw(stationb(i))=sgmw(stationb(i))-1
            sgmq(stationb(i))=sgmq(stationb(i))+1
            if(sgmq(stationb(i)) .eq. 1) then
             stastatus(stationb(i),1)=stastatus(stationb(i),2)
            else
             nrqctr(sgmq(stationb(i)))=stastatus(stationb(i),2)
              if(sgmq(stationb(i)) .gt. 300) then
               print*, 'high value for segmnts in Tx queue'
               stop
              end if
            end if
            stastatus(stationb(i),2)=0
           end if
          else
           if (tflag(stationb(i)).eq.0) then
            counters(stationb(i))=counters(stationb(i))+1
               sgmw(stationb(i))=sgmw(stationb(i))-1
               sgmq(stationb(i))=sgmq(stationb(i))+1
            stastatus(stationb(i),1)=stastatus(stationb(i),2)
            stastatus(stationb(i),2)=0
               tflag(stationb(i))=1
           endif
          endif
         endif

    c change in RQ_CTR is introduced here because station knows, before it sees the
    c  next slot whether it should transfer the content of RQ_CTR to CD_CTR
```

```
      .
      STASTATUS(STATIONB(I),2)=
     1 STASTATUS(STATIONB(I),2)+SLOTSTATUS(J,2,2)

      IF ((SLOTSTATUS(J,2,2).EQ.0).AND.
     1   (counters(stationb(I)).ge.1)) THEN
          SLOTSTATUS(J,2,2)=1
          counters(stationb(I))=counters(stationb(I))-1
      ENDIF

      IF (sgmq(STATIONA(I)).EQ.0) THEN
        IF (STASTATUS(STATIONA(I),2).GT.0) THEN
          STASTATUS(STATIONA(I),2)=
     1     STASTATUS(STATIONA(I),2)+SLOTSTATUS(I,1,1)-1
          end if
      end if

      IF (sgmq(STATIONA(I)).gt.0) THEN
       IF ((SLOTSTATUS(I,1,1).EQ.0).AND.
     1     (STASTATUS(STATIONA(I),1).EQ.0)) then
        ISFLAG(I)=1
        SLOTSTATUS(I,1,1)=1
           sgmq(stationa(i))=sgmq(stationa(i))-1
       if(stationa(i) .ne. nss) then
            bbm(stationa(i))=bbm(stationa(i))+1
            tflag(stationa(i))=0
            if(bbm(stationa(i)).eq.KVAL(stationa(i))) then
         STASTATUS(STATIONA(I),2)=STASTATUS(STATIONA(I),2)+1
            bbm(stationa(i))=0
        end if
       else
        if(sgmq(stationa(i)) .gt. 0) then
         stastatus(stationb(i),1)=nrqctr(2)
         if(sgmq(stationa(i)) .gt. 1) then
          do 111 j=2,sgmq(stationa(i))
          nrqctr(j)= nrqctr(j+1)
111         continue
         end if
        end if
       end if
      ELSE
       IF (STASTATUS(STATIONA(I),1).GT.0) THEN
       STASTATUS(STATIONA(I),1)=STASTATUS(STATIONA(I),1)+
     1   SLOTSTATUS(I,1,1)-1
       ENDIF
      ENDIF
      ENDIF

20     CONTINUE
      RETURN
      END
```

```
C ***************************************************************

      SUBROUTINE MODIFY(FLAG,ST,CURRTIM,STASTATUS,LAMDA,sgmw,
     1   sgmq,nss)


      COMMON arrivals,nuofslots,COVERAGE,DELAY,RMEAN

      INTEGER FLAG(160),STASTATUS(0:9,4),ST(160)
      integer sgmw(0:9),sgmq(0:9),nuofp,nss
      REAL ARRIVALS(10,3,200),DELAY(4,0:9),LAMDA(0:9),
     1        COVERAGE,RMEAN(0:9)
      DOUBLE PRECISION CURRTIM,time0,time1
      DO 30 sl=1,nuofslots,coverage
       i=sl

      IF (FLAG(I).EQ.1) THEN

       DELAY(2,ST(I))=DELAY(2,ST(I))+1
      if (st(i).eq.nss) then
       DELAY(1,ST(I))=DELAY(1,ST(I))+
     1        CURRTIM-ARRIVALS(ST(I)+1,1,1)+2.726
      end if

       IF (ARRIVALS(ST(I)+1,2,1).EQ.1) THEN
      if (st(i).eq.nss) then
         DELAY(3,ST(I))=DELAY(3,ST(I))+
     1        CURRTIM-ARRIVALS(ST(I)+1,1,1)+2.726
        end if
         DELAY(4,ST(I))=DELAY(4,ST(I))+1
        if(st(i).ne.nss) then
c         ARRIVALS(ST(I)+1,1,1)=ARRIVALS(ST(I)+1,1,1)+
c    1              (-1)*ALOG(1-UNI(K))/(LAMDA(st(i)))
c         arrivals(st(I)+1,2,1)=rmean(st(i))
c
c       else if( (st(i).eq.3) .or. (st(i).eq.8)) then
         sgmw(st(i))=1000
        else
            if(st(i).eq.nss) then
           If(arrivals(st(i)+1,3,1).eq.1) then
               arrivals(st(i)+1,2,1)=rmean(st(i))
               arrivals(st(i)+1,3,1)=189
               arrivals(st(i)+1,1,1)=arrivals(st(i)+1,1,2)
               arrivals(st(i)+1,1,2)=arrivals(st(i)+1,1,1)+188*84.5+
     1              (-1)*ALOG(1-UNI(K))/(lamda(st(i)))
          else
               arrivals(st(i)+1,2,1)=rmean(st(i))
               arrivals(st(i)+1,1,1)=arrivals(st(i)+1,1,1)+84.5
               arrivals(st(i)+1,3,1)=arrivals(st(i)+1,3,1)-1
              endif
         end if
        end if
        ELSE
         ARRIVALS(ST(I)+1,2,1)=ARRIVALS(ST(I)+1,2,1)-1
```

```
      END IF
     END IF

     IF(st(i).ne.nss) then
c      if(arrivals(st(I)+1,1,1) .lt. (CURRTIM+2.726)) then
c        sgmw(st(i))=arrivals(st(I)+1,2,1)-sgmq(st(i))
c       else
c        sgmw(st(i))=0
c       end if
c     ELSE IF((st(i).eq.3) .or. (st(i).eq.8)) then
        sgmw(st(i))=1000
        ELSE
      IF(arrivals(st(I)+1,1,1) .lt. (CURRTIM+2.726)) then
       if((arrivals(st(I)+1,1,2)+188*84.5) .lt. (CURRTIM+2.726)) then
        print*, 'The computation of swgm(nss) is not correct'
        stop
       else
        time1=arrivals(st(i)+1,1,1)+(arrivals(st(i)+1,3,1)-1)*84.5
        if(time1 .gt. (CURRTIM+2.726)) then
         time0=currtim+2.726-arrivals(st(i)+1,1,1)
         nuofp=int(time0/84.5)+1
         sgmw(st(i))=(nuofp-1)*rmean(st(i))+
     1        arrivals(st(i)+1,2,1)-sgmq(st(i))
        else if(arrivals(st(i)+1,1,2) .gt. (CURRTIM+2.726)) then
         sgmw(st(i))=(arrivals(st(i)+1,3,1)-1)*rmean(st(i))+
     1        arrivals(st(i)+1,2,1)-sgmq(st(i))
        else
         time0=currtim+2.726-arrivals(st(i)+1,1,2)
         nuofp=int(time0/84.5)+1
         sgmw(st(i))=(arrivals(st(i)+1,3,1)-1)*rmean(st(i))+
     1      arrivals(st(i)+1,2,1)-sgmq(st(i))+nuofp*rmean(st(i))
        end if
       end if
      ELSE
       sgmw(st(i))=0
      END IF
     END IF

30     continue

     RETURN
     END

C ***************************************************************
```

APPENDIX E

GBBM Simulation Program

```
c ***************************************************************
c     PROGRAM SIMULATION
c  Special station transmits through income and BWB mechanism. The rest, use
c  BWB mechanism.  Free segment can be changed to paid segment.
c ***************************************************************

      COMMON arrivals,nuofslots,coverage,DELAY,RMEAN
      REAL ARRIVALS(10,3,200),DELAY(4,0:9),rrate(0:9),rpropag,
     1 coverage,RMEAN(0:9)
          integer nss,inc,ncrmx,nmean,nbsp

          ICSEED=135
          ITSEED=479
       CALL RANSET(ICSEED,ITSEED)

      nss=0
      inc=30
      ncrmx=180
      nmean=6
      nbsp=0

          do 2 i=0,9
          if (i.eq.0) rmean(i)=1000
          if (i.eq.1) rmean(i)=1000
          if (i.eq.2) rmean(i)=1000
          if (i.eq.3) rmean(i)=1000
          if (i.eq.4) rmean(i)=1000
          if (i.eq.5) rmean(i)=1000
          if (i.eq.6) rmean(i)=1000
          if (i.eq.7) rmean(i)=1000
          if (i.eq.8) rmean(i)=1000
          if (i.eq.9) rmean(i)=1000
          if (i.eq.nss) rmean(i)=nmean
 2        continue

          do 5 i=0,9
      if (i.eq.0) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.1) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.2) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.3) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.4) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.5) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.6) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.7) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.8) rrate(i)=1000.7/(rmean(i)*424)
      if (i.eq.9) rrate(i)=1000.7/(rmean(i)*424)
          if (i.eq.nss) rrate(i)=1/16000.0
      do 15 j=1,200
          arrivals(i+1,3,j)=0
          arrivals(i+1,2,j)=0
          arrivals(i+1,1,j)=0
15     continue
 5        continue
```

```
      DO 10 I=0,9
         If (i.eq.nss) then
          Arrivals(i+1,2,1)=rmean(i)
          arrivals(i+1,3,1)=189
       ARRIVALS(I+1,1,2)=ARRIVALS(I+1,1,1)+188*84.5+
     1    (-1)*ALOG(1-UNI(K))/(RRATE(i))
       else
        arrivals(i+1,2,1)=rmean(i)
       endif
c      print*, ARRIVALS(I+1,1,1),ARRIVALS(I+1,2,1)
  10   CONTINUE

c      PRINT*,'Give the slot coverage(number of stations):'
c      READ*,coverage
          coverage=6.0

C   RPROPAG is the propagation from station to station exprassed
C   in microseconds  (propagation speed = 5 10-6 sec/km)
        RPROPAG=2.726
       print*,'PROPAGATION TIME:',RPROPAG
C   Capacity = 155.520 Mbs, rslottime is slottime in microseconds
c       print*,arrivals
       nuofslots=int(9*coverage)+1
       print*,'NUMBER OF SLOTS : ',nuofslots
       CALL EXECUTE(RPROPAG,RRATE,nss,inc,ncrmx,nbsp)
       do 22 i=0,9
        if (i .eq. nss) then
         print*,i,delay(1,i),delay(2,i),delay(1,i)/delay(2,i)
         print*,i,delay(3,i),delay(4,i),delay(3,i)/delay(4,i)
        else
         print*,i,delay(2,i)
        end if
  22   continue
       print*,'number of txed blnc segm from sp.st',nbsp
       END

C  ************************************************************
       SUBROUTINE EXECUTE(RINCREMENT,RRATE,nss,inc,ncrmx,nbsp)

       common table,nuofslots,coverage,DELAY,rmean
       REAL TABLE(10,3,200),RINCREMENT,DELAY(4,0:9),coverage,
     1  SLOTSTATUS(200,2,3),rmean(0:9)
       DOUBLE PRECISION TIME
       INTEGER STASTATUS(0:9,4),STAT(160),IFLAG(160),
     1  kval(0:9),credit(0:9),iter,nss,inc,ncrmx,
     1  sgmq(0:9),sgmw(0:9),tflag(0:9),rgflag(0:9),nbsp,nqctr
       NOFTRANS=0
       TIME=0
       nqctr=0
       CALL INITIALIZE(SLOTSTATUS)
```

```
      do 5 i=0,9
        if (i.eq.0) kval(i)=6
            if (i.eq.1) kval(i)=6
            if (i.eq.2) kval(i)=6
            if (i.eq.3) kval(i)=6
            if (i.eq.4) kval(i)=6
            if (i.eq.5) kval(i)=6
            if (i.eq.6) kval(i)=6
            if (i.eq.7) kval(i)=6
            If (i.eq.8) kval(i)=6
            If (i.eq.9) kval(i)=6
            if (i.eq.nss) kval(i)=14
        rgflag(i)=0
 5    continue
      DO 20 I=0,9
        STASTATUS(I,1)=0
        STASTATUS(I,2)=0
        STASTATUS(I,3)=0
        STASTATUS(I,4)=-1
        sgmw(i)=0
c       if((i .eq. nss) .or. (i .eq. 3) .or. (i .eq. 8))  then
        sgmw(i)=rmean(i)
        STASTATUS(i,1)=STASTATUS(i,2)
c       end if
        sgmq(i)=0
        tflag(i)=0
 20   CONTINUE
 10   DO 30 sl=1,nuofslots,coverage
        i=sl
        STAT(I)=int(SLOTSTATUS(I,1,3)+0.05)

 30   CONTINUE
      CALL NEXTSTEP(SLOTSTATUS,STASTATUS,IFLAG,KVAL,credit,sgmw,
     1          sgmq,tflag,nss,inc,ncrmx,rgflag,nbsp,nqctr)
      CALL MODIFY(IFLAG,STAT,TIME,STASTATUS,RRATE,sgmw,sgmq,nss)

      DO 40 sl=1,nuofslots,coverage
        i=sl
        IF (IFLAG(I).EQ.1) NOFTRANS=NOFTRANS+1
 40   CONTINUE
      iter=iter+1
      TIME=TIME+RINCREMENT
      DO 60 I=1,nuofslots
      SLOTSTATUS(I,1,3)=SLOTSTATUS(I,1,3)+(1/coverage)
      SLOTSTATUS(I,2,3)=SLOTSTATUS(I,2,3)-(1/coverage)
 60   CONTINUE
      CALL GENERATESLOT(SLOTSTATUS)
        if (mod(noftrans,1000000).eq.0)  print*,noftrans,time,
     1                  delay(1,0)/delay(2,0)
      IF (NOFTRANS.LT.4000000) GOTO 10
      print*,'TOTAL TIME IS EQUAL TO:',TIME
      RETURN
      END
```

```
C ********************************************************************

      SUBROUTINE INITIALIZE(SLOTSTATUS)


      common arrivals,nuofslots,coverage,DELAY

      REAL DELAY(4,0:9),coverage,SLOTSTATUS(200,2,3),
     1     arrivals(10,3,200)

      DO 10 I=nuofslots,1,-1
       DO 20 J=1,2
        SLOTSTATUS(I,1,J)=0
        SLOTSTATUS(I,2,J)=0
 20    CONTINUE
        SLOTSTATUS(I,1,3)=(i-1)*(1/coverage)
        SLOTSTATUS(I,2,3)=9-((i-1)*(1/coverage))
 10    CONTINUE
      RETURN
      END

C ********************************************************************
      SUBROUTINE GENERATESLOT(SLOTSTATUS)


      common arrivals,nuofslots
      REAL SLOTSTATUS(200,2,3),arrivals(10,3,200)

      DO 10 I=nuofslots-1,1,-1
        DO 20 J=1,3
         SLOTSTATUS(I+1,1,J)=SLOTSTATUS(I,1,J)
         SLOTSTATUS(I+1,2,J)=SLOTSTATUS(I,2,J)
 20      CONTINUE
 10    CONTINUE
      SLOTSTATUS(1,1,1)=0
      SLOTSTATUS(1,1,2)=0
      SLOTSTATUS(1,1,3)=0
      SLOTSTATUS(1,2,1)=0
      SLOTSTATUS(1,2,2)=0
      SLOTSTATUS(1,2,3)=9
      RETURN
      END
```

```
C ****************************************************************

      SUBROUTINE NEXTSTEP(SLOTSTATUS,STASTATUS,ISFLAG,KVAL,credit,
     1            sgmw,sgmq,tflag,nss,inc,ncrmx,rgflag,nbsp,nqctr)

      save counters,bbm,nrqctr
      common arrivals,nuofslots,coverage,DELAY
      INTEGER STASTATUS(0:9,4),ISFLAG(160),STATIONA(160),nss,
     1  STATIONB(160),counters(0:9),kval(0:9),bbm(0:9),inc,ncrmx,
     1  credit(0:9),sgmw(0:9),sgmq(0:9),tflag(0:9),nrqctr(300),
     1  rgflag(0:9),nbsp,nqctr
      REAL SLOTSTATUS(200,2,3),DELAY(4,0:9),COVERAGE,
     1    arrivals(10,3,200)


      DO 10 sl=1,nuofslots,coverage
       i=sl
       ISFLAG(I)=0
       STATIONA(I)=int(SLOTSTATUS(I,1,3)+0.05)
       STATIONB(I)=int(SLOTSTATUS(nuofslots-i+1,2,3)+0.05)
10    CONTINUE

      DO 20 sl=1,nuofslots,coverage
       i=sl
          j=nuofslots-i+1
       if(stationa(i).ne.stationb(i)) then
        print*,'slots see difr stat',stationa(i),stationb(i)
        stop
       end if
c        read*,iiik

      if (stationb(i).eq.nss) then
       if(credit(stationb(i)) .lt. ncrmx)
     1    credit(stationb(i))=credit(stationb(i))+inc
c       credit(stationb(i))=min(credit(stationb(i))+inc,ncrmx)
         endif

c ---------------------------------------------------------------
c     We change possible Free segment to a Paid segment

      if((stationb(i) .eq. nss) .and.
     1      (credit(stationb(i)).ge. 155) ) then
       IF(rgflag(stationb(i)) .eq. 1) then
        rgflag(stationb(i))=0
        tflag(stationb(i))=0
        credit(stationb(i))=credit(stationb(i))-155
        if(sgmq(stationb(i)).ne.1) then
       print*,'sgmq,sgmw',sgmq(stationb(i)),sgmw(stationb(i))
       stop
        end if
       ELSE
```

```
      .
   if(tflag(stationb(i)) .eq. 1) then
         tflag(stationb(i))=0
         credit(stationb(i))=credit(stationb(i))-155
         nrqctr(sgmq(stationb(i)))=nqctr
         nqctr=0
        end if
       END IF
      end if
c -------------------------------------------------------------

      if (sgmw(stationb(i)).gt.0) then
       if (stationb(i).eq.nss) then
            IF (credit(stationb(i)).ge.155) then
         credit(stationb(i))=credit(stationb(i))-155
         counters(stationb(i))=counters(stationb(i))+1
         sgmw(stationb(i))=sgmw(stationb(i))-1
         sgmq(stationb(i))=sgmq(stationb(i))+1
         if(sgmq(stationb(i)) .eq. 1) then
          stastatus(stationb(i),1)=stastatus(stationb(i),2)
         else
          nrqctr(sgmq(stationb(i)))=stastatus(stationb(i),2)
           if(sgmq(stationb(i)) .gt. 300) then
            print*, 'high value for segmnts in Tx queue'
            stop
           end if
         end if
         stastatus(stationb(i),2)=0
        ELSE
        if(tflag(stationb(i)).eq.0) then
         counters(stationb(i))=counters(stationb(i))+1
            sgmw(stationb(i))=sgmw(stationb(i))-1
            sgmq(stationb(i))=sgmq(stationb(i))+1
         nqctr=stastatus(stationb(i),2)
         stastatus(stationb(i),2)=0
            if(sgmq(stationb(i)) .eq. 1) then
          stastatus(stationb(i),1)=nqctr
          nqctr=0
          rgflag(stationb(i))=1
         end if
            tflag(stationb(i))=1
        endif
        END IF

       else
        if (tflag(stationb(i)).eq.0) then
         counters(stationb(i))=counters(stationb(i))+1
            sgmw(stationb(i))=sgmw(stationb(i))-1
            sgmq(stationb(i))=sgmq(stationb(i))+1
         stastatus(stationb(i),1)=stastatus(stationb(i),2)
         stastatus(stationb(i),2)=0
            tflag(stationb(i))=1
        endif
       endif
```

```
      endif

c change in RQ_CTR is introduced here because station knows, before it sees the
c  next slot whether it should transfer the content of RQ_CTR to CD_CTR

       STASTATUS(STATIONB(I),2)=
     1 STASTATUS(STATIONB(I),2)+SLOTSTATUS(J,2,2)

      IF ((SLOTSTATUS(J,2,2).EQ.0).AND.
     1   (counters(stationb(I)).ge.1)) THEN
         SLOTSTATUS(J,2,2)=1
         counters(stationb(I))=counters(stationb(I))-1
      ENDIF

       IF (sgmq(STATIONA(I)).EQ.0) THEN
        IF (STASTATUS(STATIONA(I),2).GT.0) THEN
          STASTATUS(STATIONA(I),2)=
     1     STASTATUS(STATIONA(I),2)+SLOTSTATUS(I,1,1)-1
         end if
       end if

       IF (sgmq(STATIONA(I)).gt.0) THEN
        IF ((SLOTSTATUS(I,1,1).EQ.0).AND.
     1       (STASTATUS(STATIONA(I),1).EQ.0)) then
         ISFLAG(I)=1
         SLOTSTATUS(I,1,1)=1
            sgmq(stationa(i))=sgmq(stationa(i))-1
        if( (stationa(i) .ne. nss) .or. ((stationa(i) .eq. nss)
     1    .and. (rgflag(stationa(i)) .eq. 1)) ) then
            bbm(stationa(i))=bbm(stationa(i))+1
            tflag(stationa(i))=0
         if(stationa(i) .eq. nss) then
          rgflag(stationa(i))=0
          nbsp=nbsp+1
         end if
            if(bbm(stationa(i)).eq.KVAL(stationa(i))) then
          STASTATUS(STATIONA(I),2)=STASTATUS(STATIONA(I),2)+1
             bbm(stationa(i))=0
         end if
        else
         if(sgmq(stationa(i)) .gt. 0) then
          if(sgmq(stationa(i)) .eq. 1) then
           if(tflag(stationb(i)) .eq. 1) then
            stastatus(stationb(i),1)=nqctr
            nqctr=0
               rgflag(stationa(i))=1
c            stastatus(stationb(i),2)=0
           else
            stastatus(stationb(i),1)=nrqctr(2)
           end if
          else
           stastatus(stationb(i),1)=nrqctr(2)
           if(tflag(stationb(i)) .eq. 1) ntrsf=sgmq(stationa(i))-1
           if(tflag(stationb(i)) .eq. 0) ntrsf=sgmq(stationa(i))
```

```
c
c       else if( (st(i).eq.3) .or. (st(i).eq.8)) then
         sgmw(st(i))=1000
        else
            if(st(i).eq.nss) then
           If(arrivals(st(i)+1,3,1).eq.1) then
               arrivals(st(i)+1,2,1)=rmean(st(i))
               arrivals(st(i)+1,3,1)=189
               arrivals(st(i)+1,1,1)=arrivals(st(i)+1,1,2)
               arrivals(st(i)+1,1,2)=arrivals(st(i)+1,1,1)+188*84.5+
     1               (-1)*ALOG(1-UNI(K))/(lamda(st(i)))
          else
               arrivals(st(i)+1,2,1)=rmean(st(i))
               arrivals(st(i)+1,1,1)=arrivals(st(i)+1,1,1)+84.5
               arrivals(st(i)+1,3,1)=arrivals(st(i)+1,3,1)-1
             endif
         end if
        end if
        ELSE
         ARRIVALS(ST(I)+1,2,1)=ARRIVALS(ST(I)+1,2,1)-1
        END IF
       END IF

       IF(st(i).ne.nss) then
c        if(arrivals(st(I)+1,1,1) .lt. (CURRTIM+2.726)) then
c         sgmw(st(i))=arrivals(st(I)+1,2,1)-sgmq(st(i))
c        else
c         sgmw(st(i))=0
c        end if
c       ELSE IF((st(i).eq.3) .or. (st(i).eq.8)) then
         sgmw(st(i))=1000
         ELSE
        IF(arrivals(st(I)+1,1,1) .lt. (CURRTIM+2.726)) then
         if((arrivals(st(I)+1,1,2)+188*84.5) .lt. (CURRTIM+2.726)) then
          print*, 'The computation of swgm(nss) is not correct'
          stop
         else
          time1=arrivals(st(i)+1,1,1)+(arrivals(st(i)+1,3,1)-1)*84.5
          if(time1 .gt. (CURRTIM+2.726)) then
           time0=currtim+2.726-arrivals(st(i)+1,1,1)
           nuofp=int(time0/84.5)+1
           sgmw(st(i))=(nuofp-1)*rmean(st(i))+
     1          arrivals(st(i)+1,2,1)-sgmq(st(i))
         else if(arrivals(st(i)+1,1,2) .gt. (CURRTIM+2.726)) then
          sgmw(st(i))=(arrivals(st(i)+1,3,1)-1)*rmean(st(i))+
     1          arrivals(st(i)+1,2,1)-sgmq(st(i))
         else
          time0=currtim+2.726-arrivals(st(i)+1,1,2)
          nuofp=int(time0/84.5)+1
          sgmw(st(i))=(arrivals(st(i)+1,3,1)-1)*rmean(st(i))+
     1       arrivals(st(i)+1,2,1)-sgmq(st(i))+nuofp*rmean(st(i))
         end if
        end if
       ELSE
```

```
            if(ntrsf .gt. 1) then
               do 111 j=2,ntrsf
                nrqctr(j)= nrqctr(j+1)
111             continue
              end if
            end if
           end if
          end if
         ELSE
          IF (STASTATUS(STATIONA(I),1).GT.0) THEN
           STASTATUS(STATIONA(I),1)=STASTATUS(STATIONA(I),1)+
     1     SLOTSTATUS(I,1,1)-1
          ENDIF
         ENDIF
        ENDIF

 20     CONTINUE
        RETURN
        END

C ***************************************************************

       SUBROUTINE MODIFY(FLAG,ST,CURRTIM,STASTATUS,LAMDA,sgmw,
     1   sgmq,nss)


       COMMON arrivals,nuofslots,COVERAGE,DELAY,RMEAN

       INTEGER FLAG(160),STASTATUS(0:9,4),ST(160)
       integer sgmw(0:9),sgmq(0:9),nuofp,nss
       REAL ARRIVALS(10,3,200),DELAY(4,0:9),LAMDA(0:9),
     1         COVERAGE,RMEAN(0:9)
       DOUBLE PRECISION CURRTIM,time0,time1
       DO 30 sl=1,nuofslots,coverage
        i=sl

       IF (FLAG(I).EQ.1) THEN

        DELAY(2,ST(I))=DELAY(2,ST(I))+1
       if (st(i).eq.nss) then
        DELAY(1,ST(I))=DELAY(1,ST(I))+
     1         CURRTIM-ARRIVALS(ST(I)+1,1,1)+2.726
       end if

       IF (ARRIVALS(ST(I)+1,2,1).EQ.1) THEN
        if (st(i).eq.nss) then
         DELAY(3,ST(I))=DELAY(3,ST(I))+
     1       CURRTIM-ARRIVALS(ST(I)+1,1,1)+2.726
        end if
         DELAY(4,ST(I))=DELAY(4,ST(I))+1
        if(st(i).ne.nss) then
c        ARRIVALS(ST(I)+1,1,1)=ARRIVALS(ST(I)+1,1,1)+
c    1            (-1)*ALOG(1-UNI(K))/(LAMDA(st(i)))
c        arrivals(st(I)+1,2,1)=rmean(st(i))
```

```
        'sgmw(st(i))=0
        END IF
       END IF

30      continue

      RETURN
      END

C ***************************************************************
```

BIBLIOGRAPHY

[1] F. Tobagi et.al., "EXPRESSNET: A High Performance Integrated Local Area Network", *Journal Select. Areas Commun.*, Vol. SAC-1, no.5, 1983.

[2] J. O. Limb and L. E. Flamm, "A Distributed LAN Packet Protocol for Combined Voice and Data Transmission", *Journal Select. Areas Commun.*, Vol. SAC-1,no.5, Nov. 1983.

[3] F. E. Ross, "FDDI-A Tutorial", *IEEE* Communication Magazine, May 1986.

[4] F. E. Ross, "An Overview of FDDI: The Fiber Distributed Data Interface", *Journal Select. Areas Commun.*, Vol. SAC-7, no.7, Sept. 1989.

[5] G. G. Kessler and D. L. Train, "Metropolitan Area Networks: concepts standard and services", Mc Graw Hill, 1991.

[6] R. M. Newman, et.al., "The QPSX MAN", *IEEE Communications Magazine*, June 1987

[7] IEEE DQDB Subnetwork of a Metropolitan Area Network, *ANSI/IEEE 802.6-1990* (ISO DIS 8802-6, 1991).

[8] M. Conti, et.al., "A Methodological Aproach to an Extensive Analysis of DQDB Performance Fairness", *IEEE JSAC*, Vol. SAC-9, no.1, January 1991.

[9] M. Conti, et.al., "DQDB under heavy load: Performance Evaluation and Unfairness Analysis", *Procced. INFOCOM'90*, San Fransisco, CA, June 1990.

[10] D. Karvelas and M. Papamichail, "A simulation model for DQDB", *Proceed. of 22nd Annual Pittsburgh Conference on Modeling and Simulation*, Pittsburgh, May 1991.

[11] E. L. Halne, et.al., "Improving the Fairness of DQDB", *Procced. INFOCOM'90*, San Fransisco, CA, June 1990.

Bibliography continued

[12] I. Martini, et.al., "Real Time Communication in DQDB A Comparison of Different Strategies", *Proceed. of IFth Conference on Local Computer Networks*, Minneapolis, Minnesota, September 1992.

[13] Shin-Fu Chang, et.al., "Adaptable-Bit-Rate Video Services on DQDB Access Networks" *Proceed. of ICC'91*, June 1991.

[14] D. Karvelas, M. Papamichail, "DQDB: A Fast Converging Bandwidth Balancing Mechanism that Requires No Bandwidth Loss", *Proceed. ICC '92*, Chicago, pp. 142-146, June 14-18, 1992.

[15] D. Karvelas, M. Papamichail, "Performance staudy of a New Bandwidth Balancing Mechanism under a Single and Multible Priority Classes of Traffic", *Procced. of First International Conference on Computer Communications and Networks* , San Diego, California, pp. 102-107, June 8-10, 1992.