Cycle Compensation Protocol: A Fair Protocol
for the Unidirectional Twin-Bus Architecture

Yean-Shiang Leu, Member, IEEE, and David H.C. Du

Abstract—The IEEE 802.6 Standard—Distributed Queue Dual Bus (DQDB)1—for metropolitan area networks (MAN's) has been proposed. It is based on a unidirectional twin bus architecture. The DQDB protocol lays more emphasis on the overall channel utilization than the fair sharing of channel bandwidth by all the stations. In this paper, we first describe the unfairness problem in which the upstream stations occupy most of the channel bandwidth while the downstream stations get fewer chances to transmit their packets. Many proposed possible fixes are also discussed. We propose a protocol called Cycle Compensation Protocol (CCP), which ensures fairness regardless of the ratio of end-to-end propagation delay to the slot size and also achieves almost the same throughput and delay as those of DQDB. CCP also guarantees that the channel bandwidth acquired by a station is inversely proportional to the number of busy stations and will reach this state within a limited time delay.

Index Terms—Channel access protocols, fair protocol, fiber optics channel, local area networks, metropolitan area networks, unidirectional channel.

I. INTRODUCTION

A METROPOLITAN area network (MAN) is a network capable of integrating a variety of traffic types over a single communication channel and covering a geographical scope as wide as a metropolitan area. MAN's are assumed to extend from several kilometers to over 200 km with transmission rates in excess of 45 Mbit/s and capable of carrying information such as digitized voice/data/video. Essentially, a MAN is a very large local area network (LAN) using an access protocol less sensitive to network size than those used in LAN's. It can serve more users and be used for interconnecting LAN's and large number of computers [1].

Several network architectures for MAN's have been proposed. Sze proposed a multilevel hierarchy of rings interconnected by bridges and uses two rings (one for backup) for the more critical subnetworks [2]. Gerla and Fratta proposed TreeNet, which is based on a tree topology [3], [4]. Each branch takes turns in transmitting packets according to an implicit token protocol. Maxemchuk proposed a two-connected, regular mesh network with unidirectional links, the Manhattan Street Network [5]–[7]. There are multiple paths between each pair of source and destination. The routing overhead is minimal and no intermediate buffering is necessary.

The standard protocol for MAN has been proposed by the IEEE 802.6 committee [1], [8]–[11]. The Distributed Queue Dual Bus (DQDB) protocol describes the physical layer and the medium access layer for MAN's. It is based on a twin-bus architecture as shown in Fig. 1. It consists of two counterflowing unidirectional busses (Channels A and B). It is a synchronous slotted network and allows one station to transmit a packet to one of its left-end stations and a different station to transmit another packet on a different channel to one of its right-end stations at the same time. That is, two point-to-point packets may be transmitted at the same time. If station $S_i$ intends to send a packet to (left-end) stations $S_1$ to $S_{i-1}$, the packet will be sent through Channel B. On the other hand, a packet will be sent through Channel A by station $S_i$ to (right-end) stations $S_{i+1}$ to $S_N$, where $N$ is the total number of stations.

All stations can read and write on both channels. Each station connects to each bus via an OR-write tap and a passive read tap upstream of the write tap. Stations can only read the data passing on the bus but never remove it and only alter it when allowed by the access protocol. Two slot generators, one at the extreme upstream of each bus, generate fixed length empty slots that propagate downstream on each channel. Time on each bus is divided into fixed-length slots with a fixed number of slots allocated to each 125-μs frame. Therefore, a packet will be partitioned by a station into segments of fixed length such that one segment can fit into one slot for transmission. In the rest of the paper we shall use "segments" instead of "packets" as units of data to be transmitted. There are two types of slots, queue arbitrated (QA) and pre-arbitrated (PA), and they provide packet-switched (asynchronous) and channel-switched (synchronous) services, respectively. The empty slots are allocated among these two types of services.
The QA slots are controlled and arbitrated through the distributed queue. The distributed queue is a concept of forming a first-in first-out (FIFO) transmission queue among stations generating QA packets. By reading control information in slots passing by on both busses, each station locally maintains a queue which it can decide its turn of transmission. That is, a station has to know the number of segments that arrived earlier in other stations and which should be transmitted before it can transmit its own. The distributed queue is a new concept that differs from any of the currently existing IEEE 802 family of standards. The details about the implementation of a distributed queue are presented in Section II.

In this paper, we shall concentrate on the medium access control protocol for the QA slots. The proposed DQDB protocol standard emphasizes the overall channel utilization more than the fairness of accessing the channel by stations. Due to the relative location of each station to the head end station, there is an inherent unfairness embedded in terms of accessing the empty QA slots and transmitting segments. Under certain conditions, the downstream stations will have fewer chances to access the channel. Let \( a = \left( \frac{\tau}{T_P} \right) \) where \( \tau \) is the end-to-end propagation delay and \( T_P \) is the packet transmission time. The unfairness problem is particularly serious when \( a \) is large. Furthermore, if multiple service classes are provided, higher-class packets in the downstream stations can be delayed from accessing the channel because lower-class packets in the upstream stations occupy the channel bandwidth [12]. This compromises the purpose of supporting multiple service classes. Hence, a fair access protocol is essential to ensure the equal distribution of bandwidth among ready stations and classes.

During the process of searching for a fair access protocol we have looked into several simple ways of improving the fairness of the DQDB protocol, and for each approach we have implemented a simulation model and carried out some performance evaluations. Although these approaches can indeed improve the fairness of the DQDB protocol, the degree of their improvements is still limited. That is, when \( a \) is large, the situation of unfairness still exists. However, for small to medium \( a \) the fairness of DQDB can be easily improved. We only briefly introduce these approaches in Section III.

Other schemes to fix the unfairness problem have also been proposed [13]-[19]. We also briefly discuss these schemes in Section III. However, these schemes are designed to compensate certain stations based on the estimated traffic. Therefore, they may take a longer time to reach a completely fair state or a completely fair state may never be reached when traffic fluctuates dynamically. Our objective is to design a protocol that can guarantee complete fairness regardless of the value of \( a \) and at the same time achieve similar performance (in terms of delay and throughput) as that of the DQDB protocol. After studying a number of ways to improve the fairness of the DQDB protocol, we believe that a simple way of determining how many segments a station should be allowed to transmit for a period of time is very critical to designing a fair protocol. That is, a station should be allowed to transmit several segments continuously if it has been transmitting fewer segments than it should, and a station should temporarily stop transmitting if it has been transmitting more than it should. The cycle concept used in FASNET [20], which allows only one segment to be transmitted by each station in one cycle, seems a simple way to know how many packets a station should be compensated. That is, we can easily count the number of cycles in which a station did not get a chance to transmit its packets. However, the protocols based on the cycle concept, when compared with the DQDB protocol, may suffer from lower channel utilization and longer delay, especially when traffic is light. Therefore, certain ways of improving both delay and utilization are needed. In this paper we propose a protocol called the Cycle Compensation Protocol (CCP) which guarantees complete fairness regardless of the value of \( a \) and also achieves similar channel utilization and delay as the DQDB protocol.

This paper is organized into six sections. In Section II, we present the DQDB protocol with single service class. Section III looks into the other fixes proposed. Section IV describes the proposed Cycle Compensation Protocol. We start from the very basic concept of cycle. We first briefly describe the FASNET protocol and ways to improve the performance of FASNET. The Cycle Compensation Scheme is then presented. The simulation results are provided in Section V. Some discussions on the Cycle Compensation protocol follow in Section VI as well as concluding remarks.

B. Overview of DQDB Protocol

Since the access protocols for Channel A and Channel B are identical, we shall only discuss the one for Channel A. We will assume that Channel A is the "transmission" (data) channel and Channel B is the "reservation" (request) channel. In the following, unless explicitly stated, when we say a downstream/upstream station we are referring to its relative location on Channel A. To better describe the protocol, we assume only a single service class, although the DQDB protocol can support up to four classes (now only three classes in the revised standard) of services for QA packets.

We first define several terminologies. We say "a bit is successfully set" if it was originally "0" and is now set to "1." Note that there is a small delay from the time a bit is set by a station and the result of "success" or "failure" is known by this station. It is usually of the order of few bit times. We say "a segment is ready" if the segment has arrived with all the necessary information such that it can be delivered to its destination. We say "a station is ready" if it has at least one ready segment.

The slot format of the DQDB is shown in Fig. 2. The slot format shows only the fields that are related to the medium access control protocol for the QA slots. From here on we shall refer to "QA slots" and "QA segments" simply as "slots"
and "segments." Each slot is of fixed size. The control field consists of BUSY bit and REQUEST (REQ) bit. The BUSY bit is initialized to 0 by the slot generator. The BUSY bit will be set to 1 by a station if it is transmitting a segment in that slot.

The Distributed Queue is a mechanism for forming a global first-in-first-out (FIFO) transmission queue among ready stations [88]. Every station has a request counter, a countdown counter (cd.counter) and a local request queue counter (local.request.queue counter) as shown in Fig. 3. They are initialized to zero when the stations are first brought up to operation. When a station is in operation, it continuously monitors both channels. If it sees a set REQ bit passing on Channel B, it increments the request counter value by 1. If an empty slot passes on Channel A, the station decrements the request counter value by 1 if the request counter value is greater than zero. When the station has a newly arrived segment and wishes to access the channel, the value in the request counter is the number of empty slots that must be allowed to pass this station before the station can access the channel. The station copies the value of request counter into cd.counter and sets the value of request counter to 0. It also increments the local.request.queue counter value by 1. From now on, the station will decrement the cd.counter by 1 for every passing empty slot. It will continuously do so until cd.counter becomes 0. The station will then transmit its segment in the next empty slot (and set the BUSY bit to 1). A slot is empty if its BUSY bit is 0. During this time period (from packet arrival to transmission), the station will also continuously monitor channel B and if a set REQ passes on Channel B, the value of request counter is incremented by one.

There is a condition under which a station can transmit its segment without copying the request counter to cd.counter. When a segment arrives at a station and both the request.counter and the cd.counter are zero, then if the next slot appearing on Channel A is empty, the station can transmit its segment in the slot without either copying the request.counter value (which is 0) to cd.counter or incrementing the local request queue counter. However, if one of the following two conditions occur before the station sees an empty slot on Channel A, it is required to make a request (increment the local.request.queue counter) by one:

1. The next slot passing on Channel A is busy.
2. The next slot passing on Channel B has a REQ bit set.

If (1) occurs, the station increments the local.request.queue counter value by one and will transmit its segment in the next available empty slot. If (2) occurs, the station increments both the request.counter and local.request.queue counter by one and will transmit its segment in the next available empty slot. Note that the station need not copy the request counter into cd.counter since request.counter is zero.

For every segment sent, the station is obligated to set one REQ bit (except for the exceptional case mentioned above). The local.request.queue counter indicates the number of segments that have been sent but have not yet set a REQ bit plus the one ready to be sent. If the local.request.queue counter is greater than zero, it tries to set the REQ bit in each passing slot on Channel B. If it can set a REQ bit successfully, it decrements the local.request.queue counter by one. In essence, a station can get service before it has successfully made a request and the local.request.queue counter records the number of slots that the station has occupied but has not yet made a request plus the one slot it will occupy.

The protocol can be described in detail with the station state diagram as shown in Fig. 4. Each station is in one of the following three states: Idle, Standby or Countdown. An Idle station is transferred to a Standby state when it has a segment ready and the request.counter value is zero. A station in a Standby state will transmit its segment in an empty slot. After transmitting its segment, the station returns to an Idle state. If the station in a Standby state sees a busy slot (BUSY = 1) on Channel A, it increments its local.request.queue counter by one and moves to a Countdown state. If a station sees a REQ bit set on Channel B while in a Standby state, it increments both the request.counter and local.request.queue counter by 1 and moves to a Countdown state. In the diagram, the statement above the arrow connecting states is the event that the station has just experienced. The statements below the arrow are actions taken by the station with a transition to another state. If the request.counter value is greater than zero when a segment arrives, the station copies the value of the request.counter into cd.counter and initializes the request.counter to 0. It also increments the local.request.queue counter by 1 and then moves to a Countdown state. When the cd.counter becomes zero the station will transmit its segment in the next empty slot.
Note that every station needs to monitor both channels continuously and update the three counter values according to the content of the control field in every passing slot. The four symbols a, b, c, and d in Fig. 4 indicate actions to be taken according to the control bits read on the corresponding channel.

Segments in each ready station are scheduled to be transmitted according to their positions in the distributed queue. However, the exact FIFO order can not be truly maintained because of the propagation delay between stations. Since an upstream station on Channel A, S_i, sees an empty slot before a downstream station S_j, i < j, it has a better chance of grabbing one. On the other hand, station S_i sees an unset REQ bit before station S_j, so it is easier for S_j to make a request. Assume that Station S_j sets a REQ bit to reserve a slot. When station S_i sees this REQ bit being set, it knows that a downstream station has reserved a slot and should allow an empty slot to pass by. When station S_i has a packet ready, it copies the value of request_counter to cd_counter. The value of cd_counter is the segment's position in the distributed queue. If \( a > 1 \), a station S_j's request (i.e., a REQ bit in a slot being set by station S_j) could be on the way flowing down the Channel B while a packet has just arrived at a downstream station S_i (i < j) on Channel B. If the segment arrives before the set REQ bit passes S_i, station S_i will not take into account the fact that station S_j has made a request before it. This embedded propagation delay introduces unfairness among stations.

The second cause of unfairness is that in the DQDB protocol a station can get service first before it has successfully made a request. This will improve the channel utilization and reduce the delay. However, a station that sees empty slots earlier will be favored in terms of having better chances of accessing the channel.

The DQDB protocol has been designed to improve the channel utilization. It can achieve full channel utilization. However, a station's share of channel bandwidth is based on its location on the channel. If an upstream station always has segments ready it can occupy almost all the channel bandwidth and leave the downstream stations with very little bandwidth. This is especially serious when \( a \) is large. We also verified this by simulation. The simulation results presented in Table 1 indicate that this phenomenon indeed occurs.

In Table 1, we show the percent channel utilization for various stations. We assume packet arrival rate is exponentially distributed in each station with \( \lambda_i \) being the segment arrival rate at station S_i. Since \( S_N \) does not have a downstream station, it does not generate traffic on Channel A. Hence we only show the channel utilization of the first \( N - 1 \) stations. The segment size is 424 bits and the channel bandwidth is 100 Mbps/s. The stations are positioned at equal distances on the cable.

Note that we are analyzing a MAN with a small number of stations generating traffic equally. In reality there may be hundreds of stations in the network and they may generate traffic in a uniformly distributed fashion. That is, they generate traffic proportional to the number of downstream stations.

Note that \( a > 1 \) means that several slots are transmitting and occupying different portions of the bus at the same time.

The results presented here are simply to illustrate the unfair situation that the DQDB protocol could deliver if some stations are transmitting a large number of segments continuously. This will be the case if several stations have relative large packets to send, since each packet has to be partitioned into many segments. More simulation results are presented in Section V.

As shown in Table I(a), when \( a = 0.25 \) and \( \lambda_i = 0.4 \), the asymmetric situation begins to show up. When \( a = 100 \), the unfairness is the worst. In Tables I(b) and I(c), we are assuming that the stations are transmitting a large number of packets, e.g., long files, so they are always ready. A small \( a = 0.25 \) would have created the unfair situation already.

II. POSSIBLE APPROACHES TO FIX THE UNFAIRNESS PROBLEM

There have been several proposed fixes to solve the unfairness problem [13–19], [21]. These proposals can be classified into two major categories based on the way the system load is calculated by each station: (a) statistics-based and (b) non-statistics-based. If each station has to continuously "collect" traffic statistics on the channels to calculate its probability of transmission, it is considered statistics-based. Otherwise, it is considered non-statistics-based.

A. Statistics Based Control

**P_1-Persistent Protocol:** Mukherjee and Meditch [16], [17] proposed a P_1-Persistent Protocol. Under this protocol, a ready station persists with its attempts to transmit its packet in an empty slot with probability P_1 until the transmission is complete. In order to increase the channel utilization and to be fair to all the stations each individual station needs to modify its P_1 based on the channel activities, the estimated number of active stations, and their traffic loads.

**Load-Controlled Scheduling of Traffic:** Limb [15] proposed the Load-Controlled Scheduling of Traffic (LOCOST) scheme. Every ready station measures the traffic intensity on the channels and then, based on this measurement, it determines its transmission rate until the next measurement is made. The station estimates the traffic rate Y by counting the
number of busy slots passing over a period of time. A gain
factor $g(Y)$ is calculated from $Y$ at the end of the period. The
$g(Y)$ is then used to update the packet allowance $X_t$ used in
the $i$th period. A decay factor $\beta$ is applied to “drag” $X_t$
toward a decay value $X_d$ to prevent $X_t$ of individual stations
from diverging.

The main idea in these approaches [15]–[17] is requiring
each station to monitor the traffic on the channels and, based
on the statistics observed, to throttle its transmission rate
accordingly. They can indeed improve the fairness of the
protocol. However, there are two potential problems associated
with these schemes. First, each station adjusts its transmission rate
according to the estimated traffic load. It is very likely the
estimated load is different from the real load. Therefore,
a full channel utilization may not be achieved. That is, when
compared with the DQDB protocol the performance of these
schemes may not be as good. Second, it may take a longer
period of time for the system to reach a completely fair state. It
is also possible, especially when traffic fluctuates dynamically,
that a complete fair state may never be reached.

B. Non-Statistics-Based Control

Reservation Request Control: Khalil and Koblentz [14]
proposed the Reservation Request Control (RRC) mechanism.
Every station knows the number of active upstream stations so
that it can defer to those stations by allowing the same number of
empty REQUESTs to pass on bus B before its next request. It
requires each station to announce to all downstream stations
when it becomes active. Similarly, when a station becomes
idle, it must announce its inactive state. In order to implement
the mechanism, each station needs two additional counters
and each slot has two additional bits for each priority class.

Due to the propagation delay, when a station announces
that it becomes idle on Channel A, there will be unet
requests passed by the unaware downstream stations on Channel
B. Therefore, several slots potentially are wasted. Also since
a station needs to make an announcement before it becomes
active, the delay to transmit the first segment can be longer.
The problem is even worse when a station only transmits very
few segments each time.

Bandwidth-Balancing Scheme: Hahne, Choudhury and
Maxemchuk [13] proposed the Bandwidth-Balancing Scheme.
Instead of allowing a station to transmit a segment whenever
its countdown-counter is 0 and empty slots become available,
it requires that a station transmit only a fraction $\alpha$ of the
available empty slots during a period of time. This is achieved
by artificially incrementing the request counter by 1 after
every $\beta$ packets transmitted were $\alpha = \beta/(1 + \beta)$. This forces
the station to send one extra empty slot downstream. For example,
if $\alpha = 0.9$ ($\beta = 9$) then a station transmits segments
in only nine out of the ten opportunities. This is implemented
by adding one additional counter, a trigger counter. For each
station there is a single trigger counter (now called bandwidth-
balancing counter) shared by all priority classes. When a
station transmits a segment, the counter is incremented by
1. Whenever the counter reaches $\beta$, it resets the counter and
increments the (request) counter by 1. This scheme has been
chosen by the IEEE 802.6 Standard Committee to incorporate
into the DQDB protocol to remedy the unfairness problem.
Therefore, we shall present more comparisons between this
scheme and the proposed protocol later.

Other Approaches: Leu and Du [21] proposes several possible
approaches. We briefly describe some of them in this
subsection.

a) Have an upper bound on the number of pending
requests: In order to increase the throughput and to decrease
the transmission delay, a station can get service before it
successfully sets a REQ bit. It can also accumulate its requests
in the req_counter. Intuitively, if we restrict the maximum
number of pending requests for a station or require that each
station transmit its segment only after it has successfully set
a REQ bit, the downstream stations would have a better
chance to access the channel as they can set a REQ bit
before upstream stations. This approach has been implemented
and the simulation results have shown improved fairness than
DQDB. However, when $a$ is large unfairness still exists. Note
also that the local.request_queue_counter is no longer needed
if a request has to be made before transmission. However,
the channel utilization can be degraded when there are few
stations wanting to transmit.

b) Applying distributed queue concept on reservations:
To improve the previous approach we began by applying the
distributed queue concept on both reservation and transmission
as opposed to applying it only on transmission in DQDB.
Every slot has one more bit than the original format of DQDB:
A REQ,REQ bit. Every station has three counters as in DQDB:
request_counter, cd_counter, and local.request_queue_counter.
The request_counter is used the same way as in DQDB.
However, the local.request_queue_counter is used in a different
way. Therefore, we shall concentrate on the operations
of local.request_queue_counter only. A station must set
a REQ,REQ bit on Channel A to reserve a REQ bit. In
this approach, the REQ bit is set according to a REQ
distributed queue. Every station monitors both channels
continuously. For every set REQ,REQ bit on Channel A, it increments
local.request_queue_counter by one. For every passing slot with
unet REQ bit on Channel B, it decrements the local.request_queue_counter
by one if local.request_queue_counter is greater than zero. When a
station is ready, it tries to set REQ,REQ bit on Channel A. Once
a REQ,REQ bit is set successfully on Channel A, it copies the
content of local.request_queue_counter into cd_counter
and initializes local.request_queue_counter to 0. The station then
decrements the cd_counter for every unet REQ bit seen on
Channel B. When cd_counter is 0, the station sets the REQ bit
in the next unet REQ bit on Channel B. If it has successfully
set a REQ bit, it copies the request_counter into cd_counter,
and initializes request_counter to 0. Then it follows the same
protocol as in DQDB: decrements cd_counter by 1 for every
empty slot passing on Channel A and transmitting its segment
in the next empty slot when cd_counter is 0.

The simulation results have shown that under uniform and
equal traffic distribution this scheme performs as well as
DQDB and, for small $a$, fairness can be achieved. However,
when $a$ becomes larger, unfairness still exists. The reason for
the unfairness is that downstream stations still need to set Req,Req bit before setting a Req bit. The upstream stations see empty Req,Req bits before downstream stations. If we relax the restriction on the downstream stations of setting the Req,Req bit first by allowing them (from $S(N/2)+1$ to $S(N-1)$ to set the Req bit without first setting the Req,Req bit—as long as those downstream stations still copy the local request,counter to cd.counter and let cd.counter drop to 0 before they set a Req bit—then we may balance the channel utilizations among all stations. The results have shown further improvements when this is done. However, when a is very large unfairness still exists.

c) Cycle-distributed queue dual bus: This is based on the cycle concept, which is further discussed in the next section. The main idea is that a ready station can only transmit a single segment in a cycle. After transmitting a segment, a station has to wait for another cycle to start before it can transmit another segment. The two end stations $S_1$ and $S_N$ control the cycle. A ready station is required to “successfully” make a request before it can copy the content of request,counter into cd.counter. In addition, we let a station clear a set Req bit when the following two conditions are both true.

1) The station has no segment ready (idle).

2) request,counter = 0 when an empty slot passes.

The rationale behind this is that when request,counter = 0, this station knows that all requests, if any, from downstream stations on Channel A have been satisfied. If at this time an empty slot passes on Channel A and then it sees a Req = 1 in the next slot on Channel B, it is most likely that the passing empty slot on Channel A will be occupied by that requesting downstream station. It is not necessary to let this Req = 1 to further propagate down Channel B and make unnecessary reservations at the passing stations. But this Req-bit-clearing is limited to the next slot only. We also allow a station to transmit a segment without making a request (setting a Req bit) first when both request,counter and cd.counter are zero.

Intuitively, if the number of slots in a cycle is exactly the same as the number of stations ready to transmit a segment in that cycle, we can achieve both the optimal utilization and fairness. However, in reality, an optimal cycle length may never be obtained. Since upstream stations see a new cycle (i.e., indicated by a set START bit) before downstream stations, the shorter the cycle length the more favored the upstream stations will be. Based on this observation, we have tried the following two approaches.

a) The end station does not set START = 1 in any two consecutive slots in an attempt to extend the cycle length.

That is, the minimum cycle length $\geq 2$.

b) We allocate two Req bits in each slot. That is, two Req bits can be set by two stations in a slot.

The end station sets START = 1 only when both Req bits in the incoming slots are unset and request,counter is zero. The

---

Simulation results of the two approaches have shown certain degree of improvement. However, when a is large, unfairness still exists.

From the above discussion it is clear that there is a trade-off between performance and fairness, especially when a is large. We also believe that a temporary unfair situation is unavoidable if high performance is one of the design goals. The question is how to find a simple and quick way to compensate disadvantaged stations. In the next section, we propose such a scheme called the Cycle Compensation Protocol. This protocol is based on a different concept, which can eliminate the unfairness problem and also achieve a full channel utilization.

III. CYCLE COMPENSATION PROTOCOL

Before introducing the Cycle Compensation protocol, we first present the FASNET protocol, which uses the cycle concept and is also based on the unidirectional twin-bus architecture.

In FASNET [20], [22] the two end stations coordinate in synchronization to allow all ready stations to transmit in a round-robin fashion. The slot format is as shown in Fig. 5. Station $S_1$ initiates a cycle by setting the START bit to 1. When a station wishes to transmit a segment, it tries to find an empty slot (i.e., a slot with BUSY bit = 0). It will set the BUSY bit to 1 in every passing slot until it has successfully done so. The bit is successfully set if it has not been set before by any upstream station. This station can then transmit its packet in the slot. Note that if the bit has been set before by an upstream station, then the setting of the bit by this station does not change the content of the BUSY bit.

Ready stations are allowed to transmit one segment in a cycle. When a station transmits a segment in the current cycle, it needs to wait for the beginning of the next cycle to transmit the next segment. A cycle starts when the START bit is set to 1 in a slot by the head end station. When the downstream end station $S_N$ sees an empty slot on Channel A, it sends the END bit in the next slot on Channel B as an indication to $S_1$ that a cycle has ended and a new cycle should start. On receiving a set END bit on Channel B, station $S_1$ sets the START bit in the next slot generated on Channel A. A ready station will try to transmit its segment in an empty slot when it sees the set START bit passing.

The protocol can maintain a round-robin transmission order among all the ready stations since, in the course of a cycle, ready stations get their chances to see both the beginning of a cycle and a first available empty slot from left to right. Therefore, there will be no gaps (empty slots) between the slots occupied by ready stations. However, there is a gap between two cycles. In the worst case, the gap between two cycles is
2τ + 2 slot time. This means a waste of channel bandwidth (low utilization) and possibly higher delay. This situation is the worst when \( a \) is large and very few stations are ready in a cycle.

Limb briefly mentioned three methods to improve the utilization.

1) A ready station waiting for a new cycle to start can attempt to seize any empty slots on the transmission channel once it sees an \( \text{END} = 1 \) on the reservation channel. The intercycle gap now depends on the round-trip propagation between the last active station and the end station.

2) The \( \text{END} \) bit is replaced by a \( \text{REQUEST} \) bit. A ready station writes \( \text{REQUEST} (\text{REQ}) = 1 \) on the reservation channel.

After all ready stations have been served, the head station will read \( \text{REQ} = 0 \) and initiate a new cycle. The average intercycle gap is now equal to the round-trip propagation delay from the head end station to the last active station, plus half a slot time. Due and Ghanta [23] proposed a similar improved scheme. It can guarantee 100% utilization if there are no less than \( \lceil 2τ / Tp \rceil + 2 \) stations ready in each cycle. (3) An additional bit—\( \text{REQUEST} \) bit—is added to the access control field. Every ready station indicates its desire to transmit by setting \( \text{REQ} = 1 \). The end station estimates the cycle length and transmits \( \text{END} = 1 \) timed to arrive at the head station just as the last slot to be used in the cycle is leaving the head station.

When \( a \) is large, the first and the third methods always have intercycle gaps. A full channel utilization can never be achieved. The second method is closely related to the proposed Cycle Compensation protocol. Therefore, we will discuss this method more. The second method can reduce the cycle gap. However, it does not guarantee round-robin transmission among ready stations. In other words, when the end-to-end propagation delay is large ( \( > \) slot time) and the number of ready stations is small, there can be a gap between the \( \text{REQ} \) bits set by ready stations. That is, there are unused \( \text{REQ} \) bits in between two set \( \text{REQ} \) bits. For example, Stations \( S_2 \) and \( S_{N-1} \) are the only two ready stations and the propagation delay between the two stations is greater than 10 slot times (i.e., \( a > 10 \)). That is, Station \( S_{N-1} \) realizes that a new cycle starts more than ten slot times later than Station \( S_2 \). Therefore, there will be more than ten slots with their \( \text{REQ} \) bits unset in between the \( \text{REQ} \) bits set by Stations \( S_2 \) and \( S_{N-1} \).

As a result, the cycle can be terminated prematurely and some ready stations will not get a chance to transmit in the current cycle. Since stations follow the rule of one transmission per cycle, those stations seeing the new cycle start after they have transmitted a segment can transmit their next segment. Those downstream will lag behind since they may not be in time to transmit a segment before a new cycle starts. As a result, unfairness occurs.

On the other hand, if we do not let a new cycle start each time a \( \text{REQ} = 0 \) arrives, in order to be fair to the downstream stations we require a minimum number of slots in a cycle. This certainly could improve the unfairness of the protocol, but it can potentially waste some slots too. Another potential problem with the second method is that it still requires a station to first make a request before it can transmit. This certainly will increase the packet delay. In the following, we present the Cycle Compensation (CC) protocol, which solves the unfairness problem, and its performance is almost the same as that of the DQDB.

From the previous discussion, it is clear that the protocols based on the cycle concept may suffer lower utilization and longer delay when \( a \) is large and the number of ready stations is small. In order to improve the channel utilization and to reduce the delay, ready stations need to become more aggressive to transmit their segments. However, this will intensify the unfairness among the stations. In the proposed CC protocol we plan to use the cycle concept as a simple way for a ready station to know how much it needs to be compensated if it has been denied its right to transmit. According to the cycle concept, a ready station is allowed to transmit a segment in a cycle, and if a ready station indeed gets its chance to transmit a segment, nothing needs to be done. On the other hand, if a ready station does not get a chance to transmit its segment, and the next cycle starts, then the station needs to be compensated later. Thus, we allow stations to accumulate their turns of transmission. When a station eventually gets its chance to transmit, it is allowed to make up for the turns it missed before. The amount of compensation that a station deserves is the number of cycles in which it has failed to transmit its segments because the cycles were ended prematurely. The station that has fallen behind then tries to make up for that number of requests for slots and transmit its segments.

The slot format is shown in Fig. 6. The control field contains four bits: \( \text{START} \) bit, \( \text{BUSY} \) bit, \( \text{REQ} \) bit, and \( \text{SUP} \) bit. \( \text{START} \) bit is set by the end station to indicate the start of a new cycle on that particular channel and \( \text{SUP} \) bit is used to suppress among the stations exactly one reservation to achieve the full channel utilization.

Each station keeps two counters, reservation counter \( \text{RES} \) (\( \text{counter} \)) and transmit counter \( \text{XMT} \) (\( \text{counter} \)) as shown in Fig. 7. Notes that we still assume a single service class in the network. In order to improve the channel utilization and to reduce the packet delay, we allow a station to transmit its packets before requests have been made. Therefore, the \( \text{RES} \) (\( \text{counter} \)) records the number of reservations that this station needs to make. The \( \text{XMT} \) (\( \text{counter} \)) contains the number of transmission opportunities that the station has accumulated.

We assume that every station has certain buffer spaces to store its arriving segments. Note that when a packet arrives, it needs to be partitioned into segments of fixed length. The basic idea is to allow those stations that have segments ready but failed to transmit their segments before a cycle has ended to get their compensation by transmitting multiple segments in...
Fig. 7. Each Station in Cycle Compensation protocol.

Fig. 8. Cycle Compensation protocol station state diagram (single class)

one cycle. The protocol can be described by the state diagram as shown in Fig. 8. The setting of the START bit is discussed later.

In normal operation, a station continuously monitors both channels. A station moves to a Ready state when it has segment ready. A ready station will try to transmit a segment in every passing empty slot (i.e., a slot with BUSY = 0) until its xmit.counter = 0. Every time it successfully transmits a segment in a slot, it decrements xmit.counter by 1 (xmit). The station returns to an Idle state when it has no more segments ready.

A station (note that it does not have to be a ready station) with its res.counter > 0 has to set a REQ bit in every passing slot on the reservation channel. Once it has successfully set one REQ bit in a slot, it decrements res.counter by 1. That is, the station has registered for an empty slot.

When a ready station sees a START = 1 on the transmission channel, it means that a new cycle starts and the station loses its turn to transmit one segment in the cycle. Therefore, if its queue length is larger than xmit.counter, it increments the xmit.counter by 1 and, if SUP = 0, then it sets SUP = 1. Otherwise, if SUP = 1 it increments the res.counter by 1. “Queue length” is the number of segments in the buffer spaces waiting for transmission. If its queue length is not larger than xmit.counter, it means that the station does not have enough segments and need not be compensated for this turn. We assume that queue length will increase if additional segments arrive or decrease if several segments have been transmitted.

When a station first becomes ready and moves to the Ready state from the Idle state, we require that the station wait for a new cycle before it can transmit a segment. This is one type of service. There can be another type of service, i.e., a newly ready station can participate in the current cycle and transmit one segment. There are trade-offs between these two alternatives. The first type tends to make the upstream stations less aggressive so that it is fair to all the ready downstream stations. On the other hand, the second type of service allows a segment to be transmitted in this cycle that could reduce the packet delay at the station. However, it increases the delay at the downstream stations.

When the network is under heavy load, the average overall delay will be about the same in the two type of services. When the load is light, new cycles will start very often. The delay of a segment waiting for a new cycle as in the first type of service will be very small since it will see START = 1 in almost every slot. Hence, it is preferred over the second type of service. With it, the station state diagram can be simplified as shown in Fig. 8 to only two states.

The CC protocol can achieve complete fairness among ready users regardless of the value of α. Since we allow a station to be compensated when it fails to transmit a segment in a cycle, it can accumulate the transmission opportunities. In addition, we require that each station waits for a new cycle after it has transmitted a segment unless it has been behind. A station fallen behind (lagging) will try to catch up with the other stations by transmitting more than one segment, while those non-lagging stations wait for the start of a new cycle. With the combination of cycle and compensation, complete fairness is achieved. The simulation results presented later show that this is indeed the case.

In the following, we discuss the protocol for end stations.

We explain only the protocol for left-end station $S_1$. Station $S_N$ executes an identical protocol on the other channel. We shall assume that there is a slot generator as described in IEEE 802.6 Standard. The additional responsibility of the end station $S_1$ is to check whether those non-end stations is to set $START = 1$ in the next slot on Channel A if it sees a $REQ = 0$ on Channel B. Since it is the first station to see the empty slot and the start of a new cycle, it gets to transmit a segment in it. So, the xmit.counter is no longer necessary. Because it is also the last station on Channel B, the res.counter is not necessary either. The end station should set SUP = 1 in every slot in which it transmits a segment.

The CC protocol can guarantee complete utilization of the channel bandwidth when the network is under high load. This statement can be informally proved by the following: A slot is wasted when it propagates down the channel while no station can transmit. A ready station cannot transmit when it is waiting for a new cycle to start. A cycle is started by the end station when there is no request pending. When a $REQ$ is set, it tells the end station that one station has made a reservation so it cannot
start a new cycle yet and this empty slot will be occupied by a station. On the other hand, if the end station sees a \( \text{REQ} = 0 \), it knows that no one else has made a reservation, the new cycle should start now. Otherwise, the slot will be wasted. If there is only one station continuously ready, the full channel utilization is also guaranteed. This is because we allow a user to transmit its segment in an empty slot with \( \text{START} = 1 \) without making a request. If it is the only ready station in the network and it need not make a reservation, then the \( \text{REQ} \) bit will always be 0 when it arrives at the end station. So, the \( \text{START} \) bit will be set in every slot. As a result, every slot will be occupied by the ready station. From the above arguments, the full channel utilization is achieved with CC protocol.

As opposed to DQDB protocol, the CC protocol implements the FIFO queue based on a broader concept. It can accommodate multiple segments from the same station to occupy the same position in the global transmission queue. The CC protocol places an additional responsibility on the two end stations than does DQDB. The responsibility is simply to set the \( \text{START} \) bit when certain conditions are true. It requires no additional hardware and very little modification to the DQDB.

IV. SIMULATION RESULTS

In order to provide a better comparison on the DQDB and Cycle Compensation protocol we carried out some simulation runs. The analysis and simulations for multiple classes using DQDB and CC protocol will be discussed in [12]. All simulations assume the following.

1) The slot size is 424 bits.
2) There is only a single class of service and the slot generators generate only QA empty slots.
3) The channel bandwidth is 100 Mbits/s.
4) The stations are equally positioned on the cable, i.e., the distance in between any two adjacent stations is the same.
5) The most downstream station (\( S_N \) on Channel A and \( S_1 \) on Channel B) does not generate any packet on that channel.
6) The delay is the time period from segment arrival to segment departure, which does not include segment transmission time.
7) The segment arrival rate at each station \( i \) is assumed to be a Poisson process with mean \( \lambda_i \) and is normalized to segments/segment-time.
8) The overhead of the control field in each slot is ignored in the throughput computation.

In Fig. 9(a) and 9(b), we show the delay performance of Cycle Compensation versus that of DQDB. The total number of stations \( N = 11 \). Traffic is uniformly distributed, i.e., the segment arrival rate to a station is proportional to the number of stations downstream to that station. For station \( S_1 \), \( \lambda_1 = 10/55 \), for station \( S_2 \), \( \lambda_2 = 9/55 \) and so on. Station \( S_1 \) is the right end station. It does not have any downstream station on Channel A. Hence, \( \lambda_{11} = 0 \). The \( y \)-axis is percent-normalized overall system load \( \lambda = \sum_{i=1}^{10} \lambda_i \) in units of (segments/segment-time) \( \times 100\% \). For both schemes, the percent channel utilizations are almost identical and linearly proportional to the overall load applied to the network by stations, so they are not present here. The \( y \)-axis is the normalized average segment delay.

As shown in Fig. 9(a) and 9(b), the delay of DQDB and CC Protocol are almost identical, except at high load (>90%) situation. Note that only segments which have been transmitted are counted in the average delay computation.

In Figs. 10(a)-(c), we present how quickly the CC protocol responds to the traffic condition in the network and compare with the results in [13] (i.e., Bandwidth-Balancing Scheme). These simulations assume three stations \( S_1, S_2, S_3 \) each becoming active at times 0, 5600, and 1680, respectively. Each finishes transmission (becomes idle) at times 12000, 9960, and 7840, respectively. The propagation delay between any two neighboring stations is 28 slot times. The three figures show the average throughput of three stations with measurement interval = 112, 168, and 224, respectively.

When the measurement period is 112 slot times and more than one station is ready, the throughput in each station varies in the two consecutive observations. As shown, when \( S_3 \) (or \( S_2 \)) joins, the throughput of \( S_1 \) begins to fluctuate in Fig. 10(a). This implies that \( S_1 \) is adjusting to the new traffic condition and attempts to balance its empty slot access rate.

When the measurement period is 168 slot times, the amplitudes of fluctuation reduce and show complete fairness when all three stations are active. When the measurement period is increased to 224 slot times, the fluctuation disappears with \( S_1 \) and \( S_3 \) active. It then occurs when all three stations become active.
Based on the observation from Figs. 10(a)–(c), we know that the measurement period plays an important role in terms of showing the fair share of the overall channel utilization. The curves may be vibrating or be completely flat depending on when the statistics are collected. In essence, with the CC protocol, the average throughput has converged to a fair share of the total bandwidth.

In summary, comparing with the figure from [13] (as shown in Fig. 11), the CC protocol is more responsive to the traffic conditions on the network. It can reach fair state sooner than the Bandwidth-Balancing Scheme.

To compare the channel utilization of the CC protocol with DQDB as shown in Table I, we carried out the same simulation on the CC protocol. The results are shown in Table II. The channel bandwidth is equally shared among all ready stations regardless of the number of stations and the network sizes.

We next look into the effect of the unfairness situation when only some of the stations are constantly ready. Figs. 12(a)–(d) show the percent channel utilization by stations in which five out of 51 stations are constantly generating packets ($\lambda_i = 1$) and for the rest of the stations $\lambda_i = 0.018$. We investigate four different configurations and in each configuration these five busy stations are in different relative locations. The five stations are:

1) concentrated upstream ($i = 1, 2, 3, 4, 5$);
2) concentrated downstream ($i = 46, 47, 48, 49, 50$);
3) concentrated in the midstream ($i = 24, 25, 26, 27, 28$);
4) equal-distantly distributed ($i = 2, 13, 25, 37, 49$).

It is obvious that if we can guarantee that each of those four lightly loaded stations is allocated 1.8% of channel bandwidth (with a total of 81%) and the rest of the bandwidth is shared equally by the five busy stations, then we can claim the complete fairness among all stations. In Fig. 12(a), the five busy stations are concentrated on the upstream. As shown, $S_1$ of DQDB occupies more than 25% of the bandwidth, $S_2$ 10%, etc. The Cycle Compensation protocol (CCP), on the other hand, demonstrates complete fairness.

In Fig. 12(b), the five busy stations are concentrated downstream. As shown, the unfairness situation for Stations $S_{40}–S_{50}$ has disappeared. However, the high spikes in DQDB at station $S_{40}$ demonstrates the existence of unfairness among Stations $S_{40}–S_{50}$. Again, CCP exhibits complete fairness in channel utilization.

In Fig. 12(c), the five busy stations are located in the midstream. The high spikes again indicate the unfairness of DQDB from station $S_{24}$ and after. In Fig. 12(d), the five busy stations are equally spaced along the channel. The five small spikes of CCP are the percent channel utilization by these five stations. Again, CCP demonstrates a complete fairness in channel utilization.

In order to show the relations of $xmt\_counter$ to the number of stations that is constantly ready, we present next the
simulation results as shown in Fig. 13. We consider that the busy stations are positioned at equal distances on the cable. Again, there can be many stations in the network but only some of them are generating a large amount of traffic. As shown, when there are more stations ready, the maximum xmt_counter value will be smaller. For a given number of busy stations, the maximum xmt_counter value is the same in each of them except the most upstream busy station. The most upstream busy station will have xmt_counter = 1 at most. For the rest of the busy stations, the maximum xmt_counter value will be the same and is about twice the propagation delay from the first busy station to the second busy station plus two slot times. The maximum value of the xmt_counter relates to the period length required for all stations to reach a fair state.

V. DISCUSSION AND CONCLUSION

In the Cycle Compensation protocol as presented in this paper, we assume that queue length information is readily available. That is, we assume several segments can be ready at the same time and they are stored in some buffer space. However, this requirement is not necessary. A variation of the Cycle Compensation protocol that does not need this requirement is presented in [23].

Now we would like to compare the hardware trade-off between DQDB and Cycle Compensation protocol. As described, the CC protocol requires two counters at each station as opposed to three counters in DQDB protocol (one additional counter is required and shared by all priority classes when the Bandwidth-Balancing Scheme is included in the revised standard). In the single class service, the CC protocol needs one less counter than DQDB. In [12], we present the multiple-class protocol: the Cycle Compensation Distributed Queue Dual Bus (CC-DQDB) protocol. In CC-DQDB, every station needs to have three counters for each class of service it supports (still one less than the revised standard). Every class has one cls_counter in addition to the xmt_counter and res_counter counters. Generally speaking, the protocol based on the proposed Cycle Compensation concept requires no additional hardware (counters) than the DQDB.

In this paper, we briefly described the access protocol of the proposed IEEE 802.6 Standard for Metropolitan Area Network. We identified the unfairness problem of the original DQDB protocol and proposed a new protocol called Cycle Compensation protocol. This protocol can adjust to traffic conditions in the network so that stations can fairly share the channel bandwidth. The CC protocol has been shown to achieve fairness regardless of the channel length (network size) and its performance in terms of throughput and packet delay is very close to that of DQDB.

ACKNOWLEDGMENT

The authors would like to thank the reviewers of this paper. Due to their comments and suggestions, both the quality and presentation of this paper have been greatly improved. They would also like to especially express our appreciation to Dr. Ellen Hahne at AT&T Bell Labs. Dr. Hahne has made many valuable suggestions to improve the quality of this paper. The idea of using a separate header bit, SUP, to manage the suppression of requests, originated from Dr. Hahne.

REFERENCES


Yean-Shiang Leu (S’86–M’90) received the B.S. degree from National Cheng Kung University, Taiwan, Taiwan, in 1978, the M.S. degree in aerospace engineering and in computer science in 1981 and 1984, respectively, and Ph.D. degree in 1990, all from the University of Minnesota, Minneapolis, Minnesota.

In 1984, he worked with Honeywell Inc., Bloomington, MN, as a Research Assistant. From 1985 to 1986, he worked as a Software Engineer with Beloit Corp., Beloit, Wisconsin. From 1987 to 1990, he served as a Teaching Assistant with the University of Minnesota and a Research Assistant with Honeywell Inc., Golden Valley, MN. Since 1990, he has been with San Jose Programming Lab, IBM Corp., San Jose, CA as an Advisory Programmer. His research interests include computer networks, distributed computer systems and multimedia services.

David Hung-Chang Du received the B.S. degree in mathematics from the National Tsing-Hua University, Taiwan, Republic of China, in 1974, and M.S. and Ph.D. degrees in computer science from the University of Washington, Seattle, in 1980 and 1981, respectively.

Since September 1981, he has been with the Department of Computer Science, University of Minnesota, Minneapolis, where he is now a Professor. His current research interests include CAD for VLSI circuits, high-speed networks, parallel/distributed processing, multimedia applications, and high-performance computing. He has published extensively in the above areas.