A Traffic Oriented Pre-Arbitrated Slot Reuse Scheme in DQDB Networks *

Yibin Yang and Ming T. Liu

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210-1277

E-mail: {yyang, liu}@cis.ohio-state.edu

Abstract

Distributed queue dual bus (DQDB) networks are high speed metropolitan area networks specified by the IEEE 802.6 standard. They may serve as local subnetworks to gather high speed data for transmission over wide area B-ISDN networks or as backbones in mobile networks to connect wireless infrastructures. In these situations, DQDB networks will carry a predominant portion of isochronous traffic. Thus we are motivated to investigate how to efficiently support isochronous traffic via the pre-arbitrated (PA) access method in DQDB networks. We propose a new PA slot reuse scheme which can much improve DQDB networks' utilization via arranging isochronous connections to share octets in PA slots. It is based on the idea of categorizing octets with octet types. A related problem of octet type selection is formulated and an algorithm is developed to solve the problem. Based on the algorithm's results, the proposed scheme is made efficient as well as traffic-oriented. Simulations are performed to show that the scheme is much better than the previous ones, in terms of utilization, drop rate and overhead. The new scheme is compatible with the current DQDB standard and can be implemented in the bandwidth manager and VCI server of a DQDB network.

1 Introduction

Distributed queue dual bus (DQDB) networks are high speed metropolitan area networks (MAN) specified by the IEEE 802.6 standard [4]. They intend to provide integrated services, such as data, voice and video, in urban areas. A DQDB network consists of two buses of opposite directions, as depicted in Figure 1. Each station is attached to both buses, and transmits data upstream via bus A and downstream bus B. Data are carried in 53-byte *slots* generated from two heads of buses (HOB). The slots are accessed via a queued arbitrated (QA) or pre-arbitrated (PA) method to support data services and isochronous services, respectively.

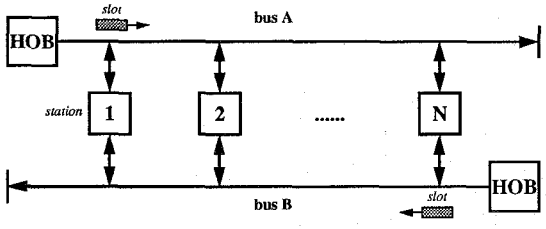


Figure 1. A DQDB network

DQDB networks have played an important role in the evolution to broadband ISDN services [5]. They are regarded as early implementations of B-ISDN. Furthermore, due to the similarity between DQDB slots and ATM cells, DQDB networks can be readily internetworked with wide-area ATM networks and serve as local subnetworks to gather high speed data for transmission over ATM backbones. Recently, DQDB networks also assume an important role in wireless communications. In a third generation wireless network architecture proposed by Goodman [2], DQDB networks are used to connect base stations, cellular control units, etc. Alternatively, a microcellular architecture based on DQDB networks as well is proposed in [7], where PA slots are employed to carry voice data. The PA access method is shown to be attractive in supporting isochronous traffic, such as voice and video, due to its bandwidth efficiency and absence of delay variance [6].

With the widespread support of multimedia applications in B-ISDN networks and wireless networks, isochronous traffic will become predominant [6]. Thus in this paper, we are motivated to investigate the issue of how to efficiently support isochronous traffic via the PA access method in DQDB networks. To carry isochronous connections, *PA slots* should be generated from HOBs at a constant rate. One PA slot has a payload of 48 octets, each of which may be assigned to one isochronous connection. It is also possible to

*Research reported herein was supported in part by U.S. Army Research Office under contract No. DAAL03-92-G-0184, and by Performance Analysis Lab of The Ohio State University. The views, opinions, and/or findings contained in this paper are those of the authors and should not be construed as an official Department of the Army position, policy or decision.

0-8186-7453-9/96 $05.00 © 1996 IEEE

assign several octets to one connection for a higher bandwidth. However, if an octet is only assigned to one connection, the bandwidth may not be efficiently utilized. Data in the octet becomes useless once the connection's destination is reached. If there is another connection whose source is behind this destination, it is desirable for the connection to reuse this octet. An important issue here is how to arrange connections to share octets so that the bandwidth utilization can be improved as much as possible. We call it *the issue of PA slot reuse*.

A heuristic algorithm, *isochronous channel reuse algorithm* (ICRA), is provided in [3] to solve this issue. However, this algorithm is difficult to implement because it assumes every station can release the octets destined to itself. In a DQDB network, only some special stations, called *erasure nodes*, can rewrite the content of a slot [8]. Other stations can only put data into slots via *OR-write* process, which requires the original values of written bits to be zero. Since the erasure nodes cause latency and require complex hardware, it is unrealistic to assume every station can function as an erasure node. On the other hand, the QA slot reuse schemes based on erasure nodes [1, 8] are slot oriented. According to [4], if a station detects a QA slot destined to itself, it marks the "previous slot received" (PSR) bit in the next slot. An erasure node holds a QA slot and may erase it until the PSR bit in the next slot is checked. In the case of PA slot reuse, a slot cannot be erased as a whole because it may carry multiple connections whose destinations are different. To deal with these two problems, we propose a new PA slot reuse scheme which works with a limited number of erasure nodes and performs erasure by octet. We also show its effectiveness in improving utilization via simulations.

The rest of this paper is organized as follows. Section 2 first explains the PA access method and the isochronous connection management. Then the previous research on PA slot reuse is presented. A new PA slot reuse scheme is proposed in Section 3. A related problem of octet type selection is defined and solved in Section 4. Section 5 evaluates the proposed scheme's effectiveness via simulations. Finally, conclusions are drawn in Section 6.

2 Preliminary

In this section, we first present the PA access method and the isochronous connection management. Then we discuss the PA slot reuse scheme proposed in [3].

2.1 PA Access Method and Isochronous Connection Management

As mentioned in Section 1, HOBs are responsible for PA slot generation. In addition, they also write a *virtual circuit identifier* (VCI) into each PA slot. A *PA slot sequence* consists of all PA slots with the same VCI. One octet in a PA slot sequence is uniquely identified by a VCI and an offset within the slots. Each station maintains a table containing the VCIs and offsets of octets used by the isochronous connections with this station as source or destination. When a PA slot passes, the station tries to match the slot's VCI with one in its table. If succeeds, the station reads or writes to the octet of the slot corresponding to the offset in the table.

The procedure to set up an isochronous connection is given in the appendix of [4]. First the connection source sends a setup request to the *signaling termination* (ST). On its behalf, the ST requests bandwidth from the *bandwidth manager and VCI server* (BMVS). The BMVS assigns octets to the connection and passes the information of the octets to its source and destination as well as the HOB. The IEEE 802.6 standard does not specify how the BMVS should assign octets to connections. In this paper, we propose a scheme for the BMVS to assign octets. Since the standard does not limit the implementation to a specific scheme, the proposed scheme should be considered compatible with the current standard.

2.2 Previous PA Slot Reuse Scheme

The ICRA scheme is proposed in [3] for the BMVS to assign octets to connections. It assumes that there are some PA slot sequences whose slots are generated at the same rate of one slot per 0.125 ms. Some established connections have already been assigned octets in these sequences while some originating ones are to be assigned. The scheme only considers the simple case where all connections are of the same bandwidth, *i.e.*, one octet per connection. Let s_i and d_i denote the source and destination of connection i, respectively. Connections i and j are *intersected* if $s_i = s_j$ or $s_i < s_j < d_i$ or $s_j < s_i < d_j$. The scheme tries to arrange multiple connections into one octet while ensuring intersected connections do not share octets. It works as follows. Let X and Y denote the sets of established and originating connections, respectively. Construct a graph whose vertexes represent connections in sets X and Y. Two vertexes are connected by an edge if their corresponding connections are intersected. Color vertexes in X by the octets assigned to them. Then try to color the vertexes in Y one by one using the octets that cannot be used by their adjacent vertexes in Y.

This heuristic algorithm has the following two problems besides the problem of unrealistic assumption mentioned in Section 1. (1) Its time complexity is very high, $O(m^3)$, where m is the total number of established and originating connections. (2) It only works well if more than one originating connection come together, which may not be the case for a real situation. For example, in a network with Poisson arrival of 10 connections per second, the possibility of having more than one originating connection within 10 ms is only 0.0047.

3 A New PA Slot Reuse Scheme

In this section, we propose a new PA slot reuse scheme which works with a limited number of erasure nodes and performs erasure by octet.

We consider a DQDB network of N stations, which are numbered from 1 to N. Due to symmetry, we only consider bus A. Assume that only K of the N stations are erasure nodes. Let E_i denote the position of the ith erasure node. In addition, we assume $E_0 = 1$ and $E_{K+1} = N$. Then the set $S_E = \{E_i | 0 \leq i \leq K+1\}$ uniquely defines the layout of erasure nodes. The erasure nodes divide the bus into segments, which are defined as follows.

Definition 1 In a bus of K erasure nodes S_E, a *segment* is a set of stations between two consecutive erasure nodes.

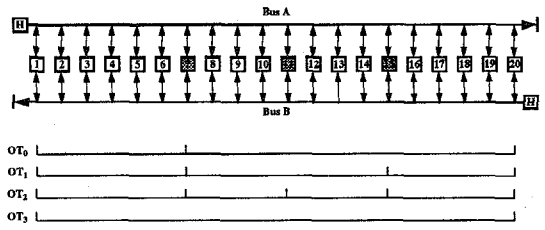


Figure 2. Octet types for a 20-station network

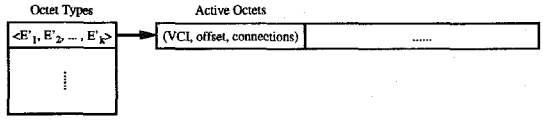


Figure 3. Table for active octets

For any $1 \leq i \leq K+1$, segment i includes stations $E_{i-1}, E_{i-1}+1, \ldots, E_i - 1$. A connection is from segment i to j if and only if its source s and destination d satisfy $E_{i-1} \leq s < E_i$ and $E_{j-1} < d \leq E_j$.

In Section 2.2, the definition of intersection is based on the assumption that every station is an erasure node. In the following, we give a new definition for the case where there are a limited number of erasure nodes, *i.e.*, $K < N$.

Definition 2 Let ss and ds denote a connection's source and destination segments, respectively. Two Connections i and j are *intersected* if $ss_i \leq ss_j \leq ds_i$ or $ss_j \leq ss_i \leq ds_j$.

In this network, we propose a PA slot reuse scheme which ensures that no intersected connections share an octet. On the other hand, it tries to arrange as many non-intersected connections as possible into one octet. In this way, bandwidth utilization can be improved with only a limited number of erasure nodes. As in [3], the new scheme is designed to handle the simple case of one octet per connection. However, it also works when connections need more octets. If a connection requests b octets, the scheme treats it like b connections, each of which requests one octet only. Before we present the scheme, we first give the following definitions.

Definition 3 An octet in a PA slot sequence is an *active octet* if it carries at least one connection's data. Otherwise, it is an *inactive octet*.

Definition 4 Segment i is *empty for an octet* if it carries no connection from segment j to k, where $j \leq i \leq k$.

Definition 5 A *octet type* OT is specified by a sequence of erasure nodes, $\langle E'_1, E'_2, \ldots, E'_k \rangle$, where $E'_1 < E'_2 < \cdots < E'_k$. Connection c is *compatible* with OT if and only if $s_c \geq E'_k$ or $d_c \leq E'_1$ or $E'_i \leq s_c < d_c \leq E'_{i+1}$ for some $1 \leq i < k$. An octet can be *associated* with an octet type. For any octet associated with OT, it can be assigned to at most $k+1$ OT-compatible connections $c_1, c_2, \ldots, c_{k+1}$, where $s_{c_i} < E'_i \leq s_{c_{i+1}}$ for $1 \leq i \leq k$.

From Definition 5, we can see that an octet type defines a reuse pattern. For example, consider the 20-station DQDB network depicted in Figure 2. Suppose stations 7, 11 and 15 are erasure nodes. Four octet types are listed in the figure, *i.e.*, $OT_0 = \langle 7 \rangle$, $OT_1 = \langle 7, 15 \rangle$, $OT_2 = \langle 7, 11, 15 \rangle$ and $OT_3 = \langle \rangle$. A connection from station 1 to 4 is compatible with OT_0 while another one from station 5 to 8 is not. Any octet of OT_0 can be assigned to two connections, *e.g.*, one from station 1 to 4 and another from station 8 to 18. In the same way, an octet of OT_1, OT_2 or OT_3 can be assigned to 3, 4 or 1 connection, respectively.

To work in the situation where erasure nodes are sparse and erasure should be done octet by octet, a PA slot reuse scheme should address two problems: (1) How to notify erasure nodes of VCIs and offsets of the octets they have to erase? (2) If more than one active octet can be used to carry an originating connection, which one should be chosen? A straightforward solution to the first problem is that whenever the BMVS assigns an octet to a connection, it also notifies the erasure node immediately behind the connection's destination to erase the octet. One slot is needed to pass the information of the octet. Therefore, this solution incurs an overhead of one slot per connection. For the second problem, the ICRA scheme of Section 2.2 provides a solution. As we have discussed before, it has many problems.

The new PA slot reuse scheme provides a better solution to the two aforementioned problems. The basic idea behind the new scheme is that every active octet is associated with an octet type. Suppose active octet o is associated with the octet type $\langle E'_1, E'_2, \ldots, E'_k \rangle$. The BMVS notifies the erasure nodes $E'_1, E'_2, \ldots, E'_k$ to erase octet o only when it becomes active. Afterward when it is assigned to other connections, no notification is needed because o can only carry the connections compatible with its octet type. The overhead is one slot per active octet because a notification can be sent out via one multicast slot. Due to reuse, the number of active octets should be much smaller than that of originating connections. Therefore, the new scheme provides a solution of lower overhead to the first problem. For the second problem, the new scheme chooses an octet by its octet type. It works even if there is only one originating connection. Furthermore, since octet types can be selected according to traffic pattern, as will be seen in Section 4, the scheme can be made traffic-oriented and therefore achieves higher bandwidth utilization. We depict the new scheme in Figure 4. We assume active octets are organized into a table as depicted in Figure 3. It is indexed by octet types and the active octets of the same octet type are put into an array pointed to by this type. The information of an active octet consists of three parts: VCI, offset and the connections carried by it.

The time complexity of the new scheme is analyzed as follows. Let M and n be the number of octet types and active octets, respectively. Step 1 executes M times. Step 3 has n iterations and for each active octet, at most $K+1$ connections are checked. Therefore the total time is $O(M + Kn)$.

So far, three questions concerning the new scheme are left unanswered: (1) How many octet types do we need? For a network of K erasure nodes, there are $\sum_{i=0}^{K} \binom{K}{i} = 2^K$ different octet types. If all are used, the new scheme has a

Choose an octet to carry an originating connection from segment ss to ds via the following steps:

1 FOR each octet type OT_i in the table DO
2 IF the connection is compatible with OT_i
3 THEN Among all active octets of OT_i, find the smallest o which can carry the connection;
4 $S_o = S_o \cup \{o\}$;
5 Pick from S_o an octet of OT_u for the connection;
6 IF no active octet is found in Step 5
7 THEN Activate an octet to carry the connection;
8 Associate the octet with OT_v;
9 Put the octet into the table;
10 Notify erasure nodes in OT_v to erase the octet;

Figure 4. The new PA slot reuse scheme

high complexity of $O(2^K + Kn)$. Could we use a small portion of the 2^K types without sacrificing the performance? If so, what are they? (2) If there are several octets of different types in S_o, how to choose OT_u in Step 5? (3) When an octet is activated, if the connection is compatible with several octet types, how to choose OT_v in Step 8? These questions will be answered in the next section.

4 Algorithm for Octet Type Selection

In this section, we first formulate the octet type selection problem. Then we present an algorithm to solve this problem. Its correctness and properties are proved. Finally, we show how to use the algorithm's results to answer the questions mentioned in Section 3.

4.1 Octet Type Selection Problem

From Section 3, we can see that the most important question is what octet types should be selected to categorize octets. We should select few types so that the scheme can maintain a low complexity. However, if too few types were selected, the scheme's performance would degrade. It is easy to see that traffic plays an important role in the selection. We formally define the selection problem as follows.

Octet type selection problem Consider a bus with N stations and K erasure nodes S_E. Let $T_{i,j}$ be the number of connections from station i to j in an equilibrium state. The traffic pattern can be characterized by $S_T = \{T_{i,j} | 1 \le i < j \le N\}$. Let S_{OT} denote the set of all 2^K octet types. Arrange the connections of S_T into octets so that

1. The number of octets is minimal.
2. The octets can be categorized by octet types which constitute a polynomial subset of S_{OT}.

4.2 Selection Algorithm

In this section, we present an algorithm to solve the selection problem. It takes N, K, S_E, S_T as input and produces the arrangement of connections into octets as output. The algorithm consists of two parts. In the first part, a graph

• Let vertex set $V = \{v_{i,j} | 1 \le i \le j \le K+1\}$;
• For each vertex $v_{i,j}$, maintain two data fields conn($v_{i,j}$) and octet($v_{i,j}$);
• Initialize conn($v_{i,j}$) $= \sum_{x=e_{i-1}}^{e_i - 1} \sum_{y=e_{j-1}+1}^{e_j} T_{xy}$ and octet($v_{i,j}$) $= 0$;
• Impose an order $<$ on vertexes so that $v_{i_1,j_1} < v_{i_2,j_2}$ if and only if $j_1 < j_2$ or $j_1 = j_2$ and $i_1 < i_2$;
• Construct edges as follows. For each edge, maintain two data fields *conn* and *octet*.

1 FOR $v_{i,j} = v_{1,1}$ TO $v_{K,K}$ BY the order of $<$ DO
/* calc # of octets needed for $v_{i,j}$'s connections */
2 total_o = max(conn($v_{i,j}$), octet($v_{i,j}$));
/* share octets with connections of $v_{j+1,k}$ $(j+1 \le k \le K+1)$ */
3 FOR $v_{j+1,k}$ where $k = K+1$ DOWNTO $j+1$ DO
/* calc # of $v_{j+1,k}$'s connections for sharing */
4 sharing_c = max(0, conn($v_{j+1,k}$) − octet($v_{j+1,k}$));
/* calc # of the octets to be shared */
/* for $v_{j+1,j+1}$, all remaining octets are shared */
5 IF $j+1 = k$
6 THEN shared_o = total_o;
7 ELSE shared_o = min(total_o, sharing_c);
/* construct edge if $v_{i,j}$ shares octets with $v_{j+1,k}$ */
8 IF shared_o > 0
9 THEN Increase octet($v_{j+1,k}$) by shared_o;
10 Decrease total_o by shared_o;
11 Construct an edge $\langle v_{j+1,k}, v_{i,j} \rangle$;
12 octet($\langle v_{j+1,k}, v_{i,j} \rangle$) = shared_o;
13 conn($\langle v_{j+1,k}, v_{i,j} \rangle$) = min(shared_o, sharing_c);
/* if all octets considered, break the loop */
14 IF total_o = 0
15 THEN Break out of the loop of Step 3;

Figure 5. Graph construction

is constructed to specify how the connections share octets. In the second part, depth-first searches are performed on the graph to find the arrangement of the connections into octets.

We use an example to help present the algorithm clearly. Consider a DQDB network of $N = 100$ stations and $K = 4$ erasure nodes, which are placed in the optimal locations $S_E = \{1, 36, 46, 55, 66, 100\}$ [1]. The traffic is uniform, with one connection from a station to each of its downstream stations. That is, $T_{i,j} = 1$ for all $1 \le i < j \le N$.

The graph construction part is depicted in Figure 5. In this part, a directed graph is generated. A vertex of the graph, $v_{i,j}$, represents the connections from segment i to j. An edge from $v_{i,j}$ to $v_{m,n}$ specifies how many connections from segment m to n should share octets with those from segment i to j. For each $v_{i,j}$, two data fields, *conn* and *octet*, are maintained. The first is the number of connections from segment i to j while the second is that of octets carrying these connections. During the initialization, *conn* could be calculated from S_T and *octet* is zero. Afterward, if

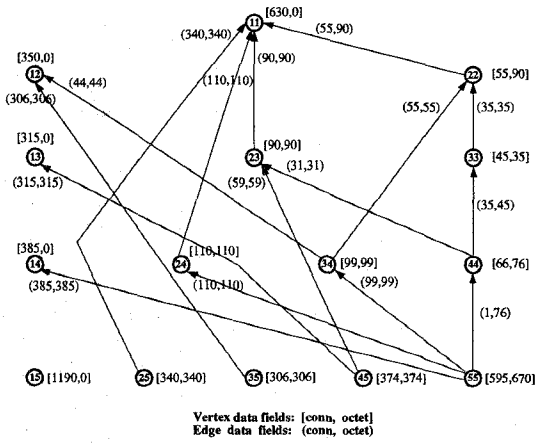


Figure 6. Graph constructed for the example

$conn > octet$, then $conn - octet$ is the number of connections not arranged into octets yet. An order $<$ is imposed on the vertexes according to their subscripts. In the example, $v_{1,1} < v_{1,2} < v_{2,2} < \cdots < v_{4,5} < v_{5,5}$. In this order, we consider each $v_{i,j}$ and try to share the octets carrying its connections with the connections of $v_{j+1,k}$, where $j+1 \leq k \leq K+1$. That is, the connections ended at segment j share octets with those starting from segment $j+1$. We depict the graph constructed for the example in Figure 6.

The graph thus constructed specifies how connections share octets. We want to prove that this sharing will lead to a minimal number of octets needed. It is easy to see the construction is a greedy method. First, connections whose destinations are in the smallest segment, segment 1, are considered to share octets with those starting from segment 2. Once these connections are arranged, the connections whose destinations are in the next smallest segment, segment 2, are considered for sharing. To prove the correctness of the greedy method, we have to show that the problem of arranging S_T into a minimal number of octets has both the greedy-choice and the optimal substructure properties.

Lemma 1 *Let n_1 be the number of connections within segment 1 and n_2 be the number of connections whose sources are in segment 2. There is an optimal solution where $\min(n_1, n_2)$ octets are shared between connections within segment 1 and those starting from segment 2.*

Proof. Suppose O is an optimal solution. We try to construct another optimal solution O' where $\min(n_1, n_2)$ octets are shared in the aforementioned way. If $n_1 \leq n_2$, select from O a set of n_1 octets containing connections from segment 2 to $K+1, K, \ldots, 2$, consecutively. For each octet in the set, if segment 1 is empty, move to it a connection within segment 1 from another octet not in the set. Since there are n_1 connections within segment 1, we can make all n_1 octets in the set carry these connections. Let O' denote the new solution produced by the movements. O' has the same number of octets as O because moving a connection within segment 1 into an octet whose segment 1 is empty does not increase the number of octets needed. On the other hand, in O', $\min(n_1, n_2) = n_1$ octets are shared between connections within segment 1 and those starting from segment 2. Therefore for this case, the lemma is correct. Now consider $n_1 > n_2$. There are only n_2 octets carrying connections starting from segment 2 in O. Similar to the first case, we can produce a solution O' via moving n_2 connections within segment 1 into these octets. Again O' is an optimal solution where $\min(n_1, n_2) = n_2$ octets are shared as specified in the lemma. ■

Lemma 2 *Suppose O' is an optimal solution to the problem of arranging S_T into a minimal number of octets and it contains $\min(n_1, n_2)$ octets shared between connections within segment 1 and those starting from segment 2. We construct from O' another solution O'' with the same number of octets as follows. For any octet in O' carrying a connection within segment 1,*

- *If it also carries a connection from segment 2 to k, replace these two connections with a connection from segment 1 to k;*
- *Otherwise, replace the connection within segment 1 with a connection from segment 1 to 2;*

Let $S'_T = (S_T - \{\text{connections replaced}\}) \cup \{\text{new connections added}\}$. Then O'' is an optimal solution to the problem of arranging connections in S'_T into a minimal number of octets.

Proof. We prove by contradiction. Suppose $\overline{O}''$ is an optimal solution to the problem of arranging connections of S'_T into a minimal number of octets and $\overline{O}''$ contains fewer octets than O''. Then we try to construct a solution $\overline{O}'$ to the problem of arranging connections in S_T into a minimal number of octets as follows. It is easy to see that S'_T has n_1 more connections from segment 1 to k ($2 \leq k \leq K+1$) than S_T. If there are m_i more connections from segment 1 to i for $3 \leq i \leq K+1$, find m_i octets in $\overline{O}''$ carrying these connections and replace each of them with one connection within segment 1 and another from segment 2 to k. Now consider the connections from segment 1 to 2. Suppose S'_T has m_2 more such connections. If $n_1 > n_2$, find $n_1 - n_2$ octets in $\overline{O}''$ carrying these connections and replace them with connections within segment 1. Then find another $m_2 - n_1 + n_2$ octets in $\overline{O}''$ carrying these connections and replace each of them with one connection within segment 1 and another within segment 2. Otherwise, if $n_1 \leq n_2$, find m_2 octets in $\overline{O}''$ carrying connections from segment 1 to 2 and replace each of them with one connection within segment 1 and another within segment 2. From the construction, we know $\overline{O}'$ is a solution to the problem of arranging S_T into a minimal number of octets and it contains the same number of octets as $\overline{O}''$. Thus, it contains fewer octets than O'', which has the same number of octets as O'. However, this is in contradiction to the assumption that O' is an optimal solution. ■

Theorem 1 *If the connections of S_T share octets as specified in the graph, the number of octets needed is minimal.*

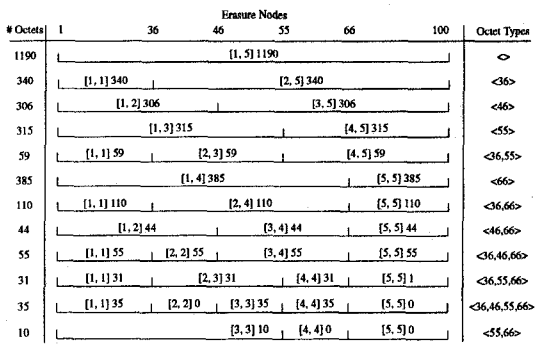
Proof. Immediate from Lemmas 1 and 2. ■

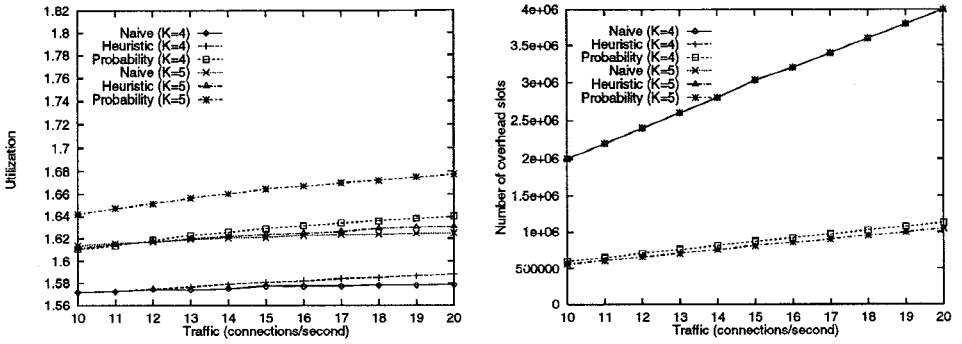
Once the graph is constructed, we can find the optimal arrangement of connections into octets via

```
FOR v_{i,j} = v_{1,K+1} TO v_{K+1,K+1} BY the order of < DO
    num_octet = max(conn(v_{i,j}), octet(v_{i,j}));
    octet_set = {⟨⟩, ...}, a set of num_octet octets;
    Depth_First_Search(v_{i,j}, octet_set);

where Depth_First_Search(v_{x,y}, octet_set) =
/* find out how v_{x,y} share octets with v_{k,x-1} */
1   FOR edge ⟨v_{x,y}, v_{k,x-1}⟩ where k = 1 TO x − 1 DO
      /* calc # of octets shared by v_{x,y} and v_{k,x-1} */
2     shared_o = min(# of octets in octet_set,
                                  octet(⟨v_{x,y}, v_{k,x-1}⟩));
      /* calc # of v_{x,y}'s connections to be carried */
3     carried_c = min(shared_o, conn(⟨v_{x,y}, v_{k,x-1}⟩));
      /* construct a new octet set and put carried_c
                              connections into its octets */
4     IF shared_o > 0
5     THEN Decrease octet(⟨v_{x,y}, v_{k,x-1}⟩) by shared_o;
6          Decrease conn(⟨v_{x,y}, v_{k,x-1}⟩) by carried_c;
7          Move shared_o octets from octet_set to
                                  curr_octet_set;
8          Select carried_c octets from curr_octet_set;
9          FOR each selected octet o DO
10              o = concat(⟨v_{x,y}⟩, o);
11         Depth_First_Search(v_{k,x-1}, curr_octet_set);
12  IF there are still some octets in octet_set
13  THEN FOR each octet o in octet_set DO
14            o = concat(⟨v_{x,y}⟩, o);
15       Print the octets with the connections carried;
16       Generate an octet type consisting of erasure
                      nodes separating the connections;
```

Figure 7. Graph searches

graph searches, which are depicted in Figure 7. The searches are depth-first, starting from the last $K+1$ vertexes, $v_{1,K+1}, \ldots, v_{K+1,K+1}$. For any $v_{i,K+1}$, it has $\text{conn}(v_{i,K+1})$ connections and some of them are arranged into $\text{octet}(v_{i,K+1})$ octets. If $\text{conn}(v_{i,K+1}) > \text{octet}(v_{i,K+1})$, $\text{conn}(v_{i,K+1}) - \text{octet}(v_{i,K+1})$ new octets are needed to carry the unarranged connections. Therefore, the search starting from $v_{i,K+1}$ provides $\max(\text{conn}(v_{i,K+1}), \text{octet}(v_{i,K+1}))$ empty octets. An octet is represented by a tuple $\langle v_{i_1,j_1}, v_{i_2,j_2}, \ldots \rangle$, indicating that it carries connections from segment i_1 to j_1, from segment i_2 to j_2, etc. The recursive *Depth_First_Search* function considers how to arrange the connections of $v_{x,y}$ into the octets in *octet_set*. Some of them may share octets with those of $v_{k,x-1}$, provided that there is an edge $\langle v_{x,y}, v_{k,x-1} \rangle$ whose *octet* field is not zero. In this case, we construct a new octet set, *curr_octet_set*, via moving some octets from *octet_set* into it and putting some of $v_{x,y}$'s connections into the octets. Since these octets will also carry the connections of $v_{k,x-1}$, we recursively call the function and pass to it $v_{k,x-1}$ and *curr_octet_set* for further arrangement. After all edges have been considered, the octets left in *octet_set* can


Erasure Nodes
Octets
1
36
46
55
66
100
Octet Types
1190
[1, 5] 1190
<>
340
[1, 1] 340
[2, 5] 340
<36>
306
[1, 2] 306
[3, 5] 306
<46>
315
[1, 3] 315
[4, 5] 315
<55>
59
[1, 1] 59
[2, 3] 59
[4, 5] 59
<36,55>
385
[1, 4] 385
[5, 5] 385
<66>
110
[1, 1] 110
[2, 4] 110
[5, 5] 110
<36,66>
44
[1, 2] 44
[3, 4] 44
[5, 5] 44
<46,66>
55
[1, 1] 55
[2, 2] 55
[3, 4] 55
[5, 5] 55
<36,46,66>
31
[1, 1] 31
[2, 3] 31
[4, 4] 31
[5, 5] 1
<36,55,66>
35
[1, 1] 35
[2, 2] 0
[3, 3] 35
[4, 4] 35
[5, 5] 0
<36,46,55,66>
10
[3, 3] 10
[4, 4] 0
[5, 5] 0
<55,66>
```

**Figure 8. Octet arrangement for the example**

be printed out. The results of the searches on the graph of Figure 6 are depicted in Figure 8. Each line represents a set printed in Step 15 of the function. It shows the number of octets and the connections carried. $\underline{[x,y]n}$ indicates that $n$ connections from segment $x$ to $y$ are carried. Octet types generated in Step 16 are also shown with the lines. By following the edges in the graph, depth-first searches can ensure that the connections are arranged into a minimal number of octets. However, there is still a question of how many octet types can be produced. We answer this question via the following theorem.

**Lemma 3** *The number of edges in the graph is upper bounded by $K^2$ ($K$ is the number of erasure nodes).*

**Proof.** According to the graph construction, an edge can only be generated between $v_{i,j}$ and $v_{j+1,k}$. For any $j$, consider sets $S' = \{v_{1,j}, \ldots, v_{j,j}\}$ and $S'' = \{v_{j+1,j+2}, \ldots, v_{j+1,K+1}\}$. Before an edge $\langle v_{j+1,k}, v_{i,j} \rangle$ is generated in Step 11, either total_o is set to 0 in Step 10 or $\text{octet}(v_{j+1,k})$ is made equal to $\text{conn}(v_{j+1,k})$ in Step 9. In the first case, the loop of Step 3 will be broken and $v_{i,j}$ not be considered to generate edges any more while in the second one, $v_{j+1,k}$ will not. Thus at least one element will be deleted from $S' \cup S''$ for one edge. Since there are $K$ elements, we have at most $K$ edges for any $j$. Since $1 \leq j \leq K$, the number of edges in the graph is bounded by $K^2$. ■

**Lemma 4** *The number of octet sets created by the Depth_First_Search function is no more than the number of edges whose* octet *fields are zero.*

**Proof.** Suppose the lemma is correct before the function is called, we prove that it is also correct after a call. The function can be called either (1) by the main program or (2) recursively by itself. For case (1), the parameters should be some $v_{i,K+1}$ and octet_set $\{\langle\rangle, \ldots\}$, which is not created by the function. Let $s_{v_{i,K+1}}$ be the sum of *octet* fields of the edges connected to $v_{i,K+1}$. Since $\text{octet}(v_{i,K+1})$ is increased with *octet* fields of its edges in the graph construction, we have $\text{octet}(v_{i,K+1}) = s_{v_{i,K+1}}$. On the other hand, since $\text{num\_octet} = \max(\text{octet}(v_{i,K+1}), \text{conn}(v_{i,K+1}))$, we have no fewer than $s_{v_{i,K+1}}$ octets in *octet_set*. To generate a new octet set in the call, an edge must be checked first. Since
```



(a) Utilization

(b) Number of overhead slots

Figure 9. 1500 octets limitation

$|\text{octet\_set}| \geq s_{v_{i,K+1}}$, *shared_o* should always be equal to the edge's *octet* field. Thus before the new set is generated, the edge's *octet* field has been set to zero in Step 5. The lemma should be correct after the call. Now consider case (2). Let the parameters be $v_{x,y}$ and set *os*, which was created by a previous call of the function. Let $s_{v_{x,y}}$ be the sum of *octet* fields of the edges connected to $v_{x,y}$. Consider two cases: (a) $|os| \geq s_{v_{x,y}}$ or (b) $|os| < s_{v_{x,y}}$. In case (a), just like case (1), a new octet set is created after an edge's *octet* field is set to zero. However, in case (b), when the last new set is created, *shared_o* is smaller than the *octet* field of the edge just being checked. Thus this edge's *octet* field will not be made zero. But the number of octet sets created by the function still remains unchanged because *os* becomes empty and is deleted after all its octets are moved to the new set in Step 7. The lemma is also correct in this case. ■

Theorem 2 *The selection algorithm generates no more than* $K^2 + 1$ *octet types.*

Proof. An octet type is generated for each set printed in Step 15 of Figure 7. These sets are either created by the Depth_First_Search function or passed from the main program. According to Lemma 4, the number of sets created by the function is not larger than that of edges in the graph, which is in turn upper bounded by K^2, as specified in Lemma 3. For any printed set not created by this function but passed from the main program, it originally consists of octets carrying no connection. Before it is printed out, at most one connection is added to each of its octets. Therefore, its octets belong to octet type $\langle\rangle$. Since at most K^2 octet types can categorize octets in sets created by the function and another one, $\langle\rangle$, can specify all those in sets passed from the main program, no more than $K^2 + 1$ types are enough for all octets. ■

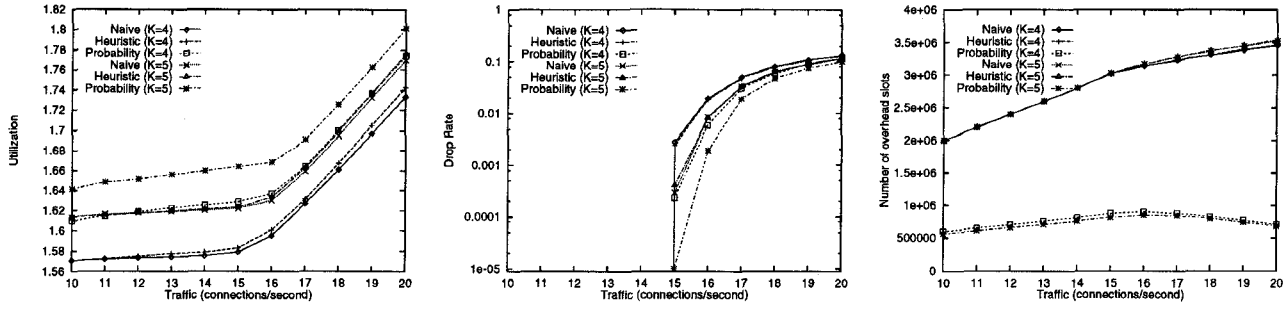
4.3 Traffic Oriented Octet Assignment

We can use the selection algorithm to generate octet types for use in the scheme proposed in Section 3. According to Theorem 2, the scheme's complexity can be lowered to $O(K^2 + Kn) = O(Kn)$. Furthermore, we may refine the scheme to take advantage of the optimal arrangement produced by the algorithm. If $c_{i,j}$ connections from segment i to j are carried by octets of OT_k in the optimal arrangement, the probability of a connection from segment i to j being assigned to an octet of OT_k, $P_{i,j,k}$, can be calculated as $c_{i,j}/\text{conn}(v_{i,j})$. Otherwise, $P_{i,j,k} = 0$. For example, from Figure 8, we can calculate $P_{1,2,3} = 306/350$, $P_{1,2,8} = 44/350$ and $P_{1,2,k} = 0$ for any other k. Based on these probabilities, we can modify Steps 5 and 8 of Figure 4. Let z denote the number of octet types generated by the algorithm. Suppose there are w octets of $OT_{k_1}, \ldots, OT_{k_w}$ in S_o. We modify Step 6 to "Pick from S_o an octet of OT_u with probability $P_{ss,ds,u} / \sum_{i=1}^{w} P_{ss,ds,k_i}$ for the connection". Similarly, Step 8 can be changed to "Associate the octet with OT_v with probability $P_{ss,ds,v} / \sum_{i=1}^{z} P_{ss,ds,i}$". After these modifications, the scheme becomes traffic-oriented because octets are chosen or associated with octet types according to the optimal arrangement of traffic pattern. This will improve the scheme's performance, as will be seen in Section 5.

5 Performance

We have performed some simulations to evaluate the new scheme. In this section, we first describe the simulation environment and then present the results.

We simulated three schemes: *naive*, *heuristic* and the new scheme, also called *probability* scheme. In the naive scheme, when a connection requests an octet, the BMVS first tries to find the smallest active octet which can carry it. If not found, activate an octet. This scheme's complexity is $O(Kn)$, where n is the number of active octets. The heuristic scheme is based on the ICRA algorithm [3]. Its performance depends on how many originating connections are considered together. We assume the BMVS considers connections at 1 second interval, which much favors the scheme. Its complexity is $O(m^3)$ [3], where m is the number of connections. Due to reuse, m is larger than n. The new scheme's complexity is $O(K^2 + Kn)$, which is in the same level as the naive scheme but lower than the heuristic one. To evaluate their performance, we consider three metrics: (1) utilization, which is the ratio of connections carried to octets used. (2) drop rate, which is the probability of an



(a) Utilization (b) Drop rate (c) Number of overhead slots

Figure 10. 1000 octets limitation

originating connection being dropped due to lack of octet. (3) number of overhead slots, which are used by the BMVS to send erasure notifications to the erasure nodes.

We simulate a 100-station network. There may be $K = 4$ or $K = 5$ erasure nodes. They are located in the optimal positions as calculated in [1]. The arrival rate ranges from 10 to 20 connections per second. The sources and destinations are uniformly distributed. The duration of a connection is exponentially distributed with 100 seconds mean. To study the schemes' behaviors under different bandwidth limitations, we consider two situations where at most 1000 or 1500 octets can be used to carry connections.

Simulation results are depicted in Figures 9 and 10. Every data point is the average of five runs, each of which simulates the BMVS's activities for 10^5 seconds. As seen from the figures, the new scheme has the highest utilization among the three. With 4 erasure nodes, it can achieve the same utilization as the other two do with 5 erasure nodes. The heuristic scheme has little advantage over the naive one. When the number of octets is limited to 1000, the schemes' utilizations increase sharply, as depicted in Figure 10(a). This can be explained by connection drops, as depicted in Figure 10(b). In order to maintain an acceptable drop rate, *e.g.*, 0.01, we are not able to achieve the highest utilization. Still, the new scheme has the lowest drop rate. When limited to 1500 octets, there is no drop for all three schemes. So their utilizations improve smoothly, as depicted in Figure 9(a). As regards overhead slots, the new scheme always generates the fewest, as shown in Figures 9(b) and 10(c). Compared with the other two schemes, it reduces the overhead by 70%.

6 Conclusions

In this paper, we have studied the issue of PA slot reuse in DQDB networks when there are only a limited number of erasure nodes and erasure has to be performed by octet instead of slot. We propose a PA slot reuse scheme which will take the traffic pattern into consideration. It is based on the idea that octets are categorized by various octet types and connections are arranged into octets according to their types.

The number of different octet types is exponential to the number of erasure nodes in the network. If all were used, the new scheme's complexity would be too high. It is desirable to select a portion of these types to achieve good performance. To this aim, we have developed a selection algorithm. Given a traffic pattern specified by a set of connections, it can arrange the connections into a minimal number of octets and select a polynomial number of octet types to categorize the octets. Based on the algorithm's results, the new scheme is made to be of low complexity and traffic-oriented. Via simulations, the scheme has been shown to be quite effective in improving bandwidth utilization.

Since DQDB networks are recently proposed to be backbones for wireless networks, further research should be done on how to adapt the new scheme into a mobile environment. In this direction, some works are under way and initial results are encouraging.

References

[1] M. W. Garrett and S.-Q. Li. A study of slot reuse in dual bus multiple access networks. *IEEE Journal on Selected Areas in Communications*, 9(2):248–256, February 1991.

[2] D. J. Goodman. Cellular packet communications. *IEEE Transactions on Communications*, 38(8):1272–1280, August 1990.

[3] N.-F. Huang, H.-I. Liu, and G.-K. Ma. A study of isochronous channel reuse in DQDB metropolitan area networks. In *Proc. INFOCOM*, pages 1319–1325, 1994.

[4] IEEE Standard 802.6. *Distributed Queue Dual Bus (DQDB) Subnetwork for a Metropolitan Area Network (MAN)*. 1990.

[5] G. C. Kessler and D. A. Train. *Metropolitan Area Networks*. McGraw-Hill, 1992.

[6] V. Leung, N. Qian, A. D. Malyan, and R. W. Donaldson. Call control and traffic transport for connection-oriented high speed wireless personal communications over metropolitan area networks. *IEEE Journal on Selected Areas in Communications*, 12(8):1376–1388, October 1994.

[7] A. D. Malyan, L. J. Ng, V. Leung, and R. W. Donaldson. Network architecture and signaling for wireless personal communications. *IEEE Journal on Selected Areas in Communications*, 11(6):830–840, August 1993.

[8] M. A. Rodrigues. Erasure node: Performance improvements for the IEEE 802.6 MAN. In *Proc. INFOCOM*, pages 636–643, 1990.