DELAY-THROUGHPUT ANALYSIS OF INTER- AND INTRA-MAN VOICE AND DATA INTEGRATED TRAFFIC IN IEEE 802.6 MAN BASED PCN

by

WILLIAM Y. L. WONG

B.Sc.(EE), University of Manitoba, 1992

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF ELECTRICAL ENGINEERING

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

February 1995

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of ELECTRICAL ENGINEERING

The University of British Columbia
Vancouver, Canada

Date Feb 27, 1995

DE-6 (2/88)

Abstract

This thesis focuses on a proposed IEEE 802.6 Metropolitan Area Network (MAN) based distributed communication architecture for supporting the Personal Communication Services (PCS). Future-generation telecommunications systems are intended to combine all kinds of networks (wireline or wireless) and services (data, voice, video, graphics, etc.) into one single universal personal communications system.

An IEEE 802.6 MAN-based distributed Personal Communication Network (PCN) architecture is introduced in this thesis, and network performance analysis based on this architecture is presented. Isochronous (voice) traffic will be transported by high priority queue-arbitrate (QA) slots instead of by using the more conventional pre-arbitrated (PA) slots. Consequently, the highest priority QA slots will be used to transport signalling traffic while the lowest priority QA slots will be used to transport data traffic. Since analytical models are not available, simulation models have been developed to evaluate the transmission delay of voice and data packets for both inter- and intra-MAN traffic. From these simulation models, nodal transmission delay is found to be depended on the physical location of the transmitting node and traffic levels within the network. Inter-MAN traffic nodal transmission delay is additionally dependent on the connection scheme of homogeneous bridges which interconnect different MANs.

With CCITT Q.931 used as the signalling protocol for call setup, the QA access for isochronous traffic (QAAIT) call setup and clearing is simpler and faster than the PA access for isochronous traffic (PAAIT), QA access avoids specific channel allocation and call clearing. Although QAAIT has its advantages in call setup and clearing, its use under light traffic load only is advised. Under heavy traffic loads PAAIT is advised, since the later case provides bounded packet transmission delay at all times by controlling network call access.

Table of Contents

List of Figures

List of Tables

Acknowledgment

I would like to take this opportunity to thank my research supervisor, Dr. R. W. Donaldson, for his guidance, suggestions and support in producing this thesis. I would also like to thank my family and friends for their encouragement and support in my graduate studies. Special thanks to my colleagues and staff in the department for their kindness and help during my studies in UBC. At last, but not least, thanks to the Canadian Institute of Telecommunications Research for continued support of this research.

Chapter 1. Introduction

Section 1.1
Background and Motivation

The B-ISDN (Broadband Integrated Services Digital Network) concept supports telecommunications involving universally available integrated broadband communications networks and services. The B-ISDN is intended to provide an integrated network capable of gathering information from variety sources and delivering it to designated destinations regardless of the service type. The B-ISDN is designed to be capable of handling all kinds of services, including data, voice, video and graphics.

Personal Communication Services (PCS's) are expected to have a continuing growth via B-ISDN in the coming decades. The PCS concept enables people to call other people irrespective of the geographical location of either party, using only callee's personal identification number (PIN). PCS access to the B-ISDN is via both wireline and wireless terminals [1]. The Personal Communication Network (PCN) which supports PCS's must be capable of the following [2]:

- carrying many types of traffic;
- being integrated with current and future wireline communication networks;
- serving a mass market in urban areas;
- operating efficiently in sparsely populated areas;
- operating indoors, outdoors, and in vehicles;
- serving terminals which may move at high speed;

The many distinctive features and potential demands required for the PCN suggests that the centralized communication architectures may not be suitable for implementing the PCN. The

predicted growth in mobile subscribers requires that cell dimensions be reduced substantially from the size currently used in mobile radio system, to improve spectral efficiency through frequency reuses. Reduction in cell radii will increase handoff rates. With the BTS antenna in these microcells situated a few meters above ground, a 20–30 dB drop in signal level can occur at street corners when a mobile terminal (MT) loses line-of-sight with the BTS. As a result of the increased processing load arising from the growing demand for wireless access, together with the necessity of a fast handoff due to the propagation effects discussed above, current centralized network and control architectures are probably unsuitable for future PCN's.

A distributed communication architecture based on the IEEE 802.6 MAN (Metropolitan Area Network) has been proposed for implementing the PCN. The proposed architecture of IEEE 802.6 MAN will be discussed in the next section, and all the analyses in this thesis are based on this architecture. Not surprisingly, MANs are expected to be a significant element in the evolution towards B-ISDN.

Section 1.2
PCN Architecture Based on IEEE 802.6 MAN

The proposed IEEE 802.6 MAN based distributed PCN architecture is shown in Figure 1.

The MAN has two 155 Mbps buses, one for each direction of transmission. Several base stations (BSs), Local Area Networks (LANs), and Private Branch Exchange (PBXs) are connected to a MAN node and are subnetworks of the MAN. Each MAN node functions as an interface to enable subnetworks to access the local MAN. Consequently, information between subnetworks can be exchanged within the local MAN, which then serves as a distributed switch. Interconnection of adjacent MANs enables coverage of an entire Metropolitan Area (MA) as shown in Figure 2. Initially, the relatively few MANs deployed in the MA would be

interconnected via point-to-point homogeneous bridges. As the deployed number of MANs

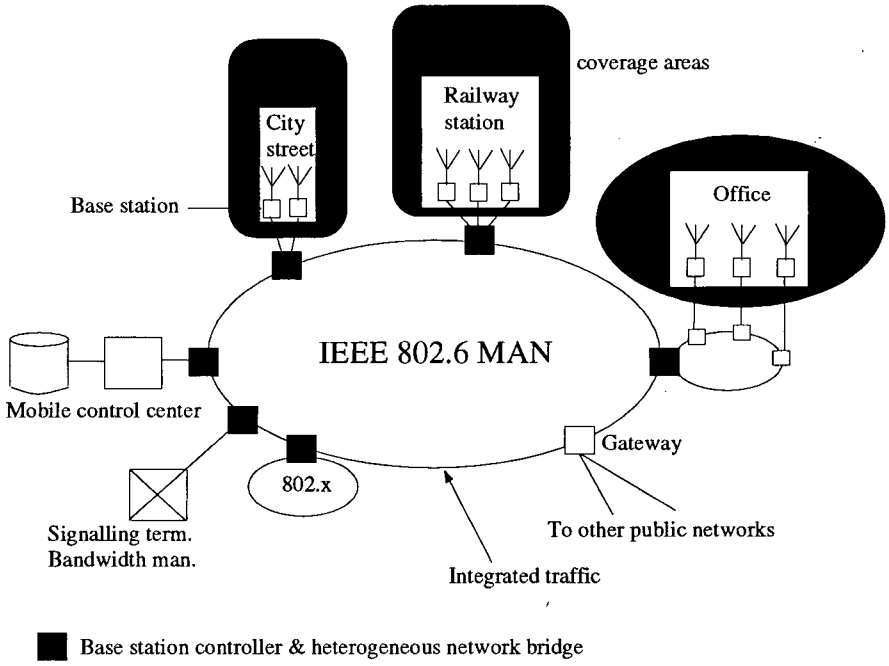


Figure 1. IEEE 802.6 MAN based distributed PCN architecture

rises, network manageability would necessitate the use of a centralized higher-speed multiport bridge — a B-ISDN switch at a Central Office (CO). Since the existing connection-oriented telecommunication networks will continue to exist for many years, an Interworking Unit (IWU) at the CO should have the functionality to support both ATM and Synchronous Transfer Mode (STM) switching fabrics. Additionally, a backbone MAN could interconnect these access MANs using homogeneous bridges to form a cluster (see Figure 2). An IWU would then connect the backbone MAN(s) in a MA. If neighboring access MANs within a cluster are also bridged, call set up and inter-MAN handoff would be simplified, and reliability against bridge failure would be enhanced [1].

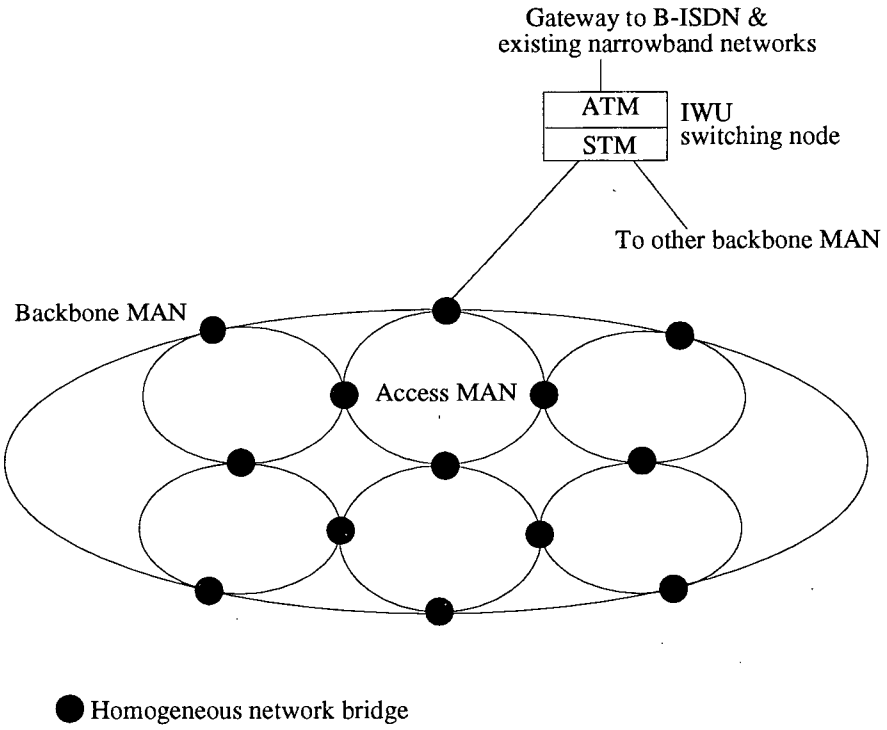


Figure 2. MAN interconnection within a backbone MAN [1]

The IEEE 802.6 MAN provides two modes of access control to the dual bus. These are Queue arbitrated (QA) and Pre-Arbitrated (PA), which use QA and PA slots for access, respectively. QA access is controlled by the Distributed Queueing protocol (details of the protocol are provided in chapter 2) and would be used typically to provide non-isochronous services, such as data or graphics. PA access would be used to provide isochronous services, such as real-time voice or video [3]. Although allocation of PA slots involves some overhead, PA access provides a constant packet transmission delay at any time. On the other hand, QA access provides a variable packet transmission delay with a simpler access method. In this thesis, QA access is used to provide all services in order to achieve full statistical multiplexing.

The main advantage of using the 802.6 MAN as the base-site interconnection network is that not only can it readily interface to the current wireline public network facilities, but it

is also capable of implementing the Asynchronous Transfer Mode (ATM) for interworking between B-ISDNs and MAN-based PCNs. Since the cell format of ATM is similar to 802.6 MAN, the interworking between two networks is simplified and the transition to ATM will be relatively easy and inexpensive. In that case, the use of the 802.6 MAN as a transit network for ATM-based B-ISDN appears not only as a stepping stone in the evolution, but also as a more permanent solution [4].

Section 1.3
Outline of The Thesis

In chapter 2, the IEEE 802.6 MAN with distributed queue dual bus (DQDB) operation is described. The DQDB protocol with QA access will be evaluated along with two slot reuse schemes, *Destination Release and Previous Slot Information.* These reuse schemes are solely for data packet transmission. Different types and levels of user traffic are considered.

Transmission of voice and data integrated traffic using QA access on a single MAN is described and analyzed in chapter 3. Voice and data packet waiting delay is evaluated for different user traffic levels and different speech encoding/decoding rate at voice stations.

In chapter 4, transmission delay of voice and data integrated traffic using QA access for Inter- and Intra-MAN will be evaluated. Various level of inter-MAN traffic, different user traffic levels and different speech encoding/decoding rates at voice stations is also considered in the analysis. For completeness, comparison of call setup delay between QA access isochronous traffic and PA access isochronous traffic is presented.

A summary of the main results is presented in Chapter 5, together with suggestions for further work.

Chapter 2. Performance Analysis of IEEE 802.6 MAN

In order to understand the working of IEEE 802.6 MAN, the MAN architecture and protocol mechanisms are reviewed in this chapter. The general performance of the DQDB protocol using QA access with and without slot reuse schemes, including Destination Release and PSI, are compared using simulation models designed solely for data packet transmission.

Section 2.1
The IEEE 802.6 MAN

2.1.1 The IEEE 802.6 MAN architecture

The DQDB medium access control (MAC) protocol has been standardized by the IEEE 802.6 committee as part of its MAN standard. The DQDB protocol is intended for use with a dual-bus configuration as illustrated in Figure 3.

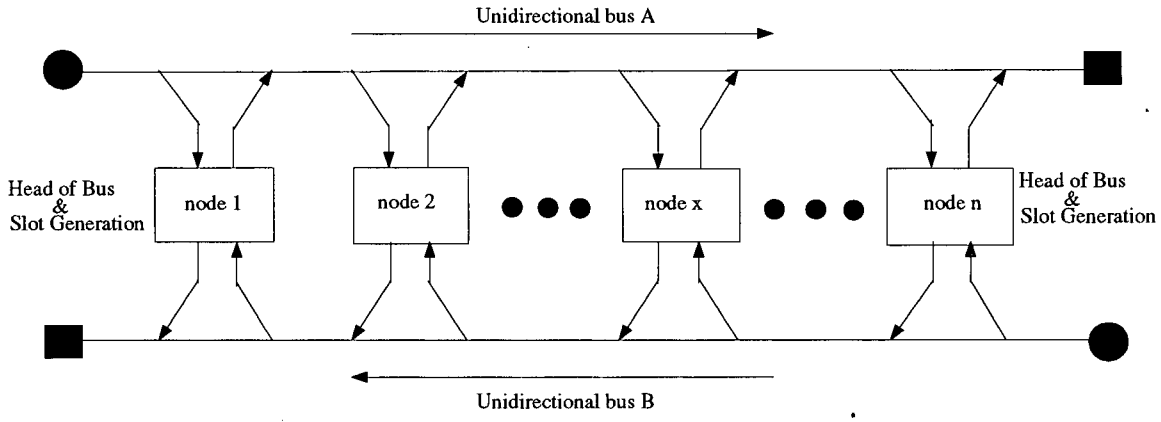


Figure 3. DQDB dual-bus configuration

As shown in Figure 3, each node is connected to two unidirectional buses which transmit in opposite directions. A node accesses the right-moving bus to send packets to nodes on its right and the left-moving bus for nodes on its left. Two head nodes are responsible for generating

slots on two buses. Transmission on each bus consists of a steady-stream of fixed-size slots with a length of 53 octets. The 53 octet length was chosen for compatibility with ATM. Nodes read and copy data from the slots and gain access to the bus by writing to empty slots. Streams of slots are controlled by a 125 μsec-clock. Two head stations generate multiple slots to the shared medium every 125 μsec; the number of slots generated per clock cycle depends on the physical data rate. Figure 4 shows the frame and slot format for the DQDB MAN.

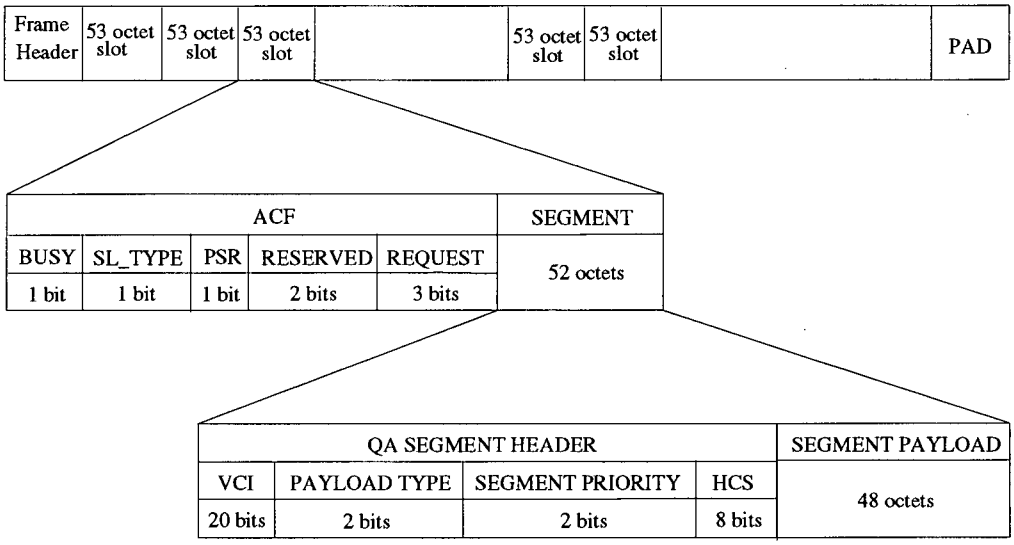


Figure 4. Frame and slot format in DQDB MAN

Each 53–octet slot contains a 1–octet Access Control Field (ACF) and a 52–octet segment. The ACF contains five fields that control access to slots. The BUSY bit indicates whether or not the slot contains information (BUSY=1, NOT BUSY=0). The SL_TYPE bit indicates whether the slot is a QA slot (SL_TYPE=0) or a Pre-Arbitrated (PA) slot (SL_TYPE=1). Table 1 shows the slot state with respect to the combinations of BUSY and SL_TYPE. The PSR bit indicates whether the segment in the previous slot may be cleared (PSR=1) or not (PSR=0). The REQUEST field contains three REQ bits which are used in the operation of the three priority

level distributed queue access mechanism. The remaining 52 octets of the slot are divided into a 4–octet QA segment header and a 48–octet payload. The QA segment header contains four fields. The 20–bit Virtual Channel Identifier (VCI) provides a means to identify the virtual channel to which the QA segment belongs. A single VCI space is shared by all services. The VCI value corresponding to all bits being set to one is the default value of the connectionless MAC service provided by the MAC Convergence Function. All nodes shall use the default connectionless VCI for both transmission and reception of QA segments. The 2–bit Payload_Type field indicates the nature of the data to be transferred. The 2–bit Segment_Priority field is reserved for future use with Multiport Bridging. The 8–bit Header Check Sequence (HCS) field provides for detection of errors and correction of single-bit errors in the QA segment header [3].

BUSY	SL_TYPE	Slot State
0	0	Empty QA slot
0	1	Reserved
1	0	Busy QA slot
1	1	PA slot

Table 1. Slot Access Control Field coding [3]

Normally, PA slots are use to carry isochronous traffic (e.g. voice, video) while QA slots carry data traffic. For PA slots, some arbitration function must be used to allocate a dedicated sequence of slots for each stream of data and this function has not yet been standardized. In this thesis QA access only will be employed for all traffic types.

2.1.2 The distributed queue access protocol

The 802.6 medium access control technique is the distributed-queue dual-bus (DQDB) protocol. The DQDB layer is independent of the physical layer. Therefore, a variety of DQDB

networks can be implemented using the same access layer but operating at different rates over different transmission system [5]. Three common transmission systems for DQDB MAN are as follows:

- ANSI DS3: transmits at 44.736 Mbps.
- ANSI SONET (CCITT SDH): transmits at 155.52 Mbps and above.
- CCITT G.703: transmits at 34.368 Mbps and 139.264 Mbps.

In this thesis, we will consider the ANSI SONET transmission system as the foundation of our DQDB protocol analysis. Distributed queueing is a media access protocol that controls the access of QA slots on the DQDB bus. The DQDB protocol is well suited for handling bursty traffic.

The operation of the DQDB protocol involves use of two control fields: the BUSY bit indicates whether or not a slot is used, and the REQUEST field indicates when a segment has been queued for access. Each node, by counting the number of requests it receives and unused slots that pass, can determine the number of segments queued ahead of it. Therefore, whenever a node has a segment for transmission, the node uses this count to determine its position in the distributed queue. This counting operation establishes a single queue across the subnetwork of segments queued for access to each bus [3]. The behavior of each node, refer to Figure 3, is summarized in Table 2 (in this table, we are referring to A as the transmitting bus and B as the request bus).

(a) Behavior of Node 1:

(i) At time when it is ready to issue the next QA slot on bus A

	No preceeding requests outstanding	One or more preceeding requests outstanding
Node 1 has no data to send	Issue a free QA slot	Issue a free QA slot and reduce preceeding requests by one
Node 1 has a QA segment to send	Insert data into the QA slot; any following requests become preceeding request	Issue a free QA slot and reduce preceeding requests by one

(ii) At time when Node 1 receives the next QA slot on bus B

	Incoming slot contains a request	Incoming slot does not contain a request
Node 1 has no data to send	Add 1 to preceeding requests	—
Node 1 has a segment to send	Add 1 to following requests	—

(b) Behavior of Node x:

(i) At time when it observes a free QA slot on bus A

	No preceeding request outstanding	One or more preceeding request outstanding
Node x has no data to send	Let free slot pass	Let free slot pass and reduce preceeding request by one
Node x has a QA segment to send and has issued a request on bus B	Inset data into the QA slot; any following requests become preceeding requests	Let free slot pass and reduce preceeding request by one

(ii) At time when Node x observes a QA slot on bus B

	Incoming slot contains a request	Incoming slot does not contain a request
Node x does not have an outstanding request	Add 1 to preceeding requests	—
Node x has a QA segment to send and has issued a request for that segment	Add 1 to following requests	—
Node x has a QA segment to send and has not issued a request for that segment	Add 1 to preceeding requests	Insert request into the passing slot

Table 2. Behavior of nodes in Figure 3 [5]

Section 2.2
Slot Reuse Schemes

In order to investigate the efficiency of QA access to the DQDB MAN, two slot reuse schemes, *Destination Release and Previous Slot Information (PSI)*, together with the absence of slot reuse are compared.

2.2.1 Destination Release

In Destination Release slot reuse, every destination station releases slots for further reuse by downstream stations. This scheme offers maximum possible usage of capacity. However, the complexity of the receiver hardware is increased, the access protocol becomes more complicated and the delay at every station increases.

2.2.2 Previous Slot Information (PSI)

PSI, introduced by Sharon and Segall [6], is a more adaptive scheme and less efficient than Destination Release, but will perform slot reuse without delay and without increasing the

hardware complexity. PSI utilizes the fact that a station is always listening to the bus in order to find messages destined to itself, and that every station reads the destination of every passing slot. PSI is based on a comparison of the destination addresses of consecutive slots. The scheme employs a reuse flag R in the slot's ACF (Access Control Field). Flag R is set by a transmitting node in any slot whose destination is beyond the destination of the packet that was received and retransmitted in the previous slot. If a busy slot, B=1, is received, a station α performs reuse if both the destination of the previous received slot is located before α and R=0. The setting R=0 implies that the destination of the current slot is located before the destination of the previous slot. If station α reuses the slot and if the destination of the current slot is beyond the destination of the previous slot, R is set to 1.

A simple example of PSI is given in Table 3. This example indicates the value of the Destination Address (DA) and the R bit of four consecutive slots travelling to stations connected to the bus in the order α, β, γ, δ; station α transmits to station β, and station γ transmits to δ starting with the second slot.

Slot	0		1		2		3	
Stns	DA	R bit	DA	R bit	DA	R bit	DA	R bit
α	NULL	0	β	1	β	0	β	0
β	NULL	0	β	1	β	0	β	0
γ	NULL	0	β	1	δ	1	δ	0
δ	NULL	0	β	1	δ	1	δ	0

Table 3. Example of PSI [6]

Since slot 0 remains empty, its DA is represented as NULL and its R bit remains 0. Station α sets the R bit of slot 1 since the NULL address of slot 0 is considered upstream to all stations, and α transmits in slot 1 to β. Since the R bit is set, slot 1 will never be reused. When α

transmits in slot 2, it does not set the R bit because it does not transmit to a station beyond the destination of slot 1. Station γ knows that slot 1 is destined to β which is before itself, and so observing that the R bit of slot 2 is not set, it can reuse slot 2 to transmit data to δ. But it sets the R bit because it transmits to station δ which is beyond β. Thus slot 2 can never be reused again. Finally, both stations α and γ transmit in slot 3 and neither transmits to a station beyond their last transmitted packet destinations, β and δ respectively, so R bit is not set.

Section 2.3
Modelling and Simulation of the DQDB Protocol

2.3.1 DQDB protocol model

The DQDB protocol serves a distributed collection of queues; usual service order in each queue is first-in, first-out (FIFO). Since the access control mechanisms of two buses are identical we discuss the access control of bus A only. Consider that node α has a data packet to send to node β which is downstream of node α. Node α sends a request on Bus B to the head node of Bus A to obtain an empty slot on Bus A for packet transmission.

At each node, a queue of each priority level is formed and two counters, ReQuest Counter (RQC) and CountDown Counter (CDC) for each queue are employed to track the status of the distributed queue of Bus A. The RQC is used to track the number of slots requested by the downstream stations while the CDC is used for transmission queueing. For example, when a station sees the request bit of a slot set on Bus B, then station increments its RQC by one. After a station sends its request, the value of its RQC is transferred to its CDC and its RQC is reset to zero. The CDC is now decremented for each passing idle slot on Bus A until it reaches zero, when the station is then allowed to transmit in the next idle slot. While awaiting transmission on Bus A, the RQC continues to increment for any new request made by downstream stations

on Bus B. Evidently, each station will only have one and only one packet of information in the system queue at any time, because a new request is not allowed until the current data packet has been transmitted.

2.3.2 Simulation models

In this preliminary analysis, we only consider the transmission of one-way single-priority level traffic (i.e. data traffic) on a DQDB MAN. Three models are developed for performance comparison: (1) standard DQDB MAN, (2) DQDB MAN with PSI slot reuse, and (3) DQDB MAN with Destination Release slot reuse. All simulation models are written in **SIMSCRIPT II.5** [7].

The simulation model for standard DQDB MAN is built exactly as described in the last section. However, modifications are needed for PSI-DQDB MAN and Destination Release-DQDB MAN models.

In addition to the standard DQDB MAN model, the PSI-DQDB MAN model introduces a R bit and a reuse bit in the ACF in each slot, and two memory units to keep track of the previous slot address received (PSAR) and the previous slot address transmitted (PSAT) in each node. The two memories, PSAR and PSAT, are used to control the setting of R bit in the current slot and determine whether or not the current slot can be reused. Each node continuously updates the two memories as each slot passes that node. If a node has not transmitted in the previous slot, its PSAR will equal its PSAT. PSAR may not be equal PSAT if a node has transmitted in an empty slot or has reused a slot. The R bit may be set only if a node transmits either in an empty slot or by reusing a slot, as described earlier in section 2.2.2.

Implementing the Destination Release DQDB MAN model is simpler. The BUSY bit in the ACF is reset to 0 once a node copies the information from a QA slot which is addressed to that

node. Apart from these differences, the following assumptions are common to all three models:

i. The data rate on each bus is 155 Mbps
ii. Slot length is 53 octets (424 bits)
iii. Slot transmission time is 2.73 μs
iv. Nodes are evenly distributed along the bus
v. Successive nodes are separated by 3 slots
vi. The number of buffers available to each node is 30
vii. All message are 1 slot in duration
viii. Inter-arrival time of data packets at each node is exponentially distributed
ix. Each node has the same Poisson arrival rate
x. The number of nodes, excluding two head stations, is 100 (50 nodes per bus)
xi. Simulation time equals to 50,000 slots

Since PSI performance is highly dependent on the traffic destination distribution, two bursty data traffic models have been simulated for completeness. The first one is the uniform destination traffic distribution; the second is the geometric destination traffic distribution. The probability function, P(x), of a geometric distribution is as follows:

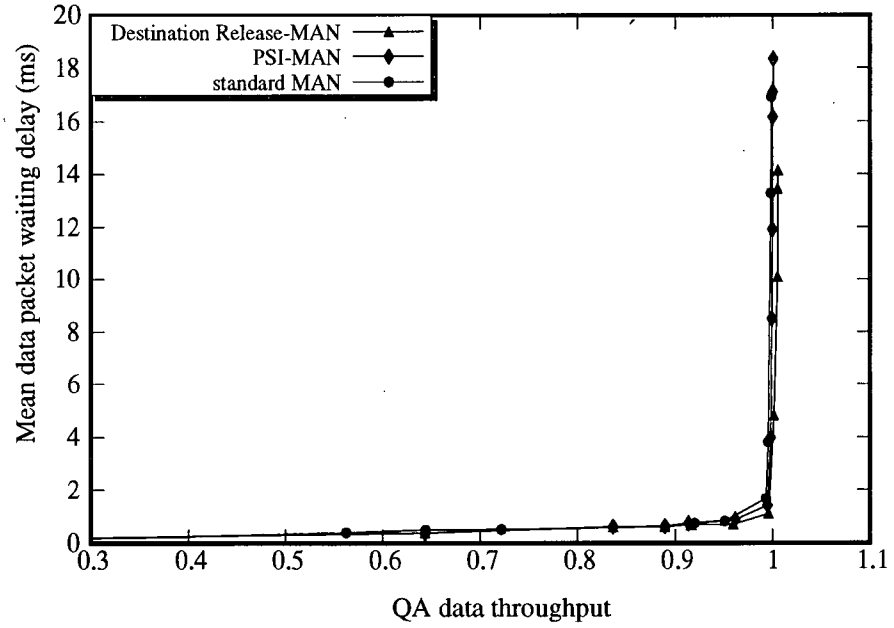
$$P(x) = \theta(1-\theta)^{x-1} \qquad x = 1, 2, 3, ...$$

where θ is the degree of traffic locality [8]. We selected θ to be 0.9, to simulate traffic with a high degree of locality.

2.3.3 Simulation results

Simulation results plotted in Figures 5 and 6 show mean packet delay versus throughput. Mean packet delay is defined as the elapsed time from when a packet enters the station buffer until that packet is successfully transmitted. Figure 5 shows the uniform destination distribution results while Figure 6 shows the geometric destination distribution results.

(a) Overview for low and high traffic levels



(b) Zoom in for high traffic level

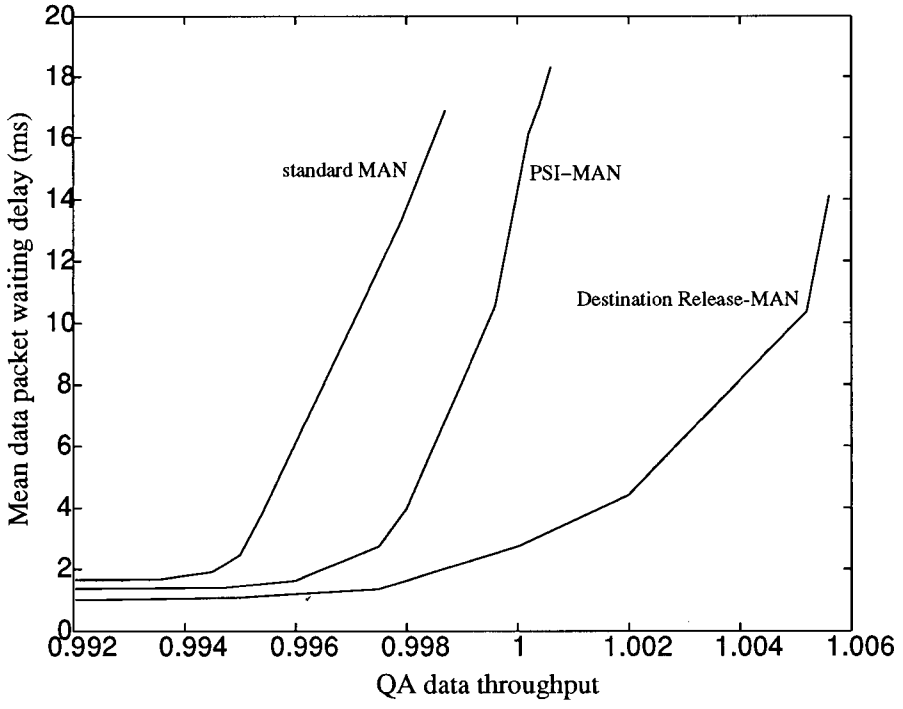
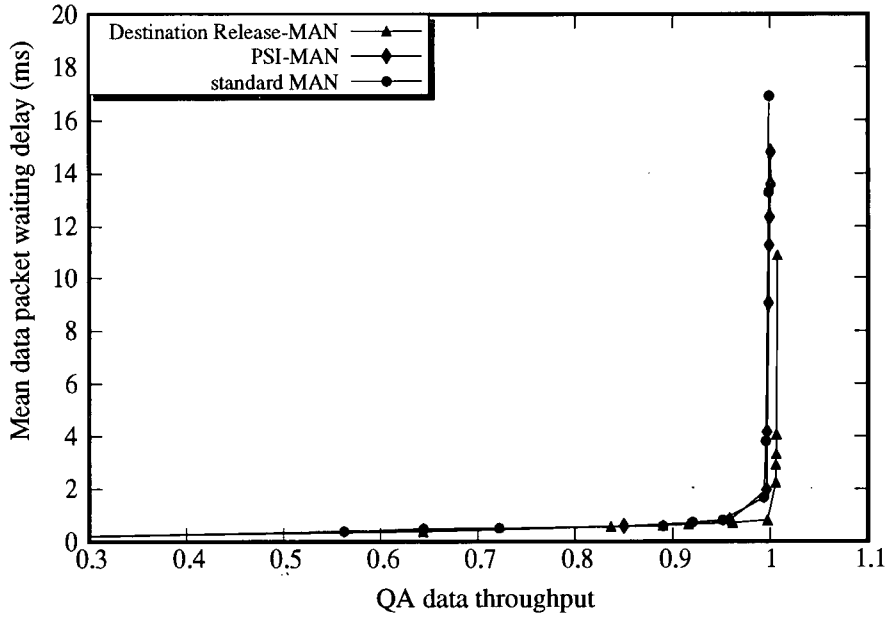


Figure 5. Uniform destination distribution for data packets in a DQDB MAN

(a) Overview for low and high traffic levels



(b) Zoom in for high traffic level

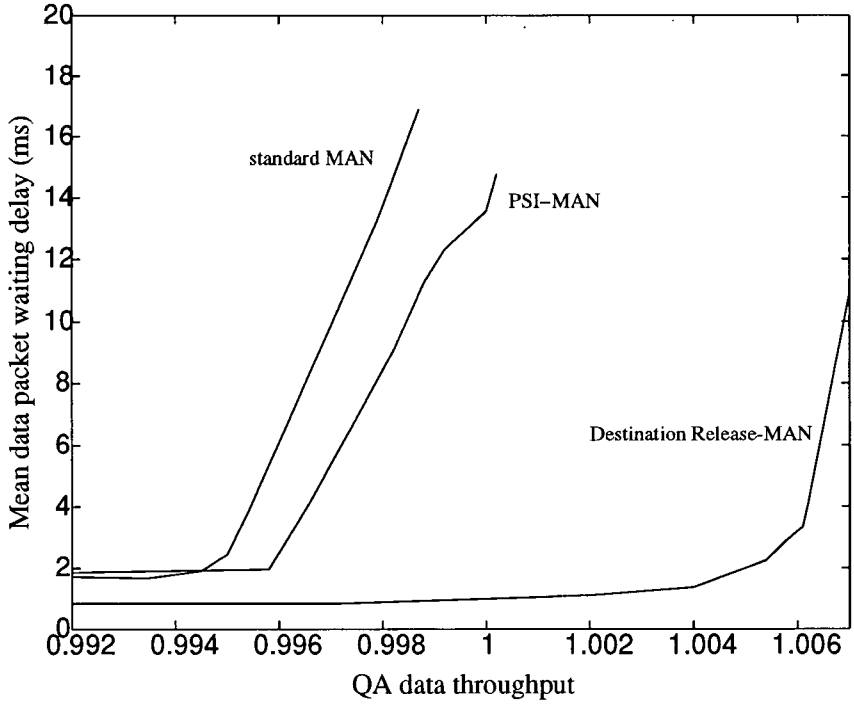


Figure 6. Geometric destination distribution for data packets in a DQDB MAN

From Figures 5a and 6a, one observes that all three simulation models have approximately the same mean packet delay at traffic levels below 0.95, for both uniform and geometric traffic. However, under heavy traffic (above 0.95), Figures 5b and 6b show that the PSI slot reuse scheme shows some improvement in the packet waiting delay with respect to the QA throughput. The Destination Release slot reuse scheme shows even a better improvement in the packet waiting delay with respect to the QA data throughput than the PSI.

At a given delay value, the two slot reuse schemes do not result in much improvement in throughput (i.e. less than 10 percent). The PSI slot reuse scheme shows less improvement in geometric destination traffic than in uniform destination traffic , once a station transmitted its current packet, say in slot 1, slot 2 is either occupied or idle while passing the station. When the address memories are updated prior to arriving at slot 2, the updated addresses have a very high probability at being less than the station's address because the upstream stations' packet addresses are highly localized. Therefore the probability of having a set reuse bit, R, in the slot where the station transmitted its next packet is increased considerably for geometrically distributed destination traffic. This result occurs because a station is allowed only one packet in the DQDB system queue at any time. If a station were not limited to having at most one packet in the system queue, the performance of PSI in geometrically distributed destination traffic would be better than in uniformly distributed destination traffic.

The results in Figures 5 and 6 suggest that it is not worthwhile to implement either slot reuse schemes DQDB MAN, since either one complicates the implementation and could introduce processing delay.

Chapter 3. Delay Analysis of QA Access for Integrated Intra-MAN Voice and Data Traffic

Voice packets have different transmission requirements from data packets. To preserve the integrity of a conversation, voice packets must be delivered within a maximum delay time comparable to the period between generation of consecutive voice packets. Some loss of voice packet is tolerable. On the other hand, data packets can undergo longer and variable delay but no loss [9]. Base on these properties of voice and data packet transmission requirements, a system model is built for voice and data integrated traffic in a IEEE 802.6 MAN. The model allows for use of different values of key parameters such as speech digitizing rate, voice stations activation time and bandwidth available for QA access.

Section 3.1
Voice-Data Integration

It is assumed that a fixed number of voice calls is activated at each node for each simulation, and these calls will remain active throughout the simulation. Each voice call independently alternates between a talkspurt period and a silence period; these are independent exponentially distributed random variables with mean $1/\alpha$=1.5 s and $1/\mu$=2.25 s, respectively. Therefore, the probability, P(t->s), that a voice call which changes from talkspurt to silence in f seconds is

$$P(t \rightarrowtail s) = 1 - e^{-\alpha f}$$

and from silence to talkspurt is

$$P(s \rightarrowtail t) = 1 - e^{-\mu f}$$

Figure 7 shows the transition probabilities of the associated two-state Markovian model.

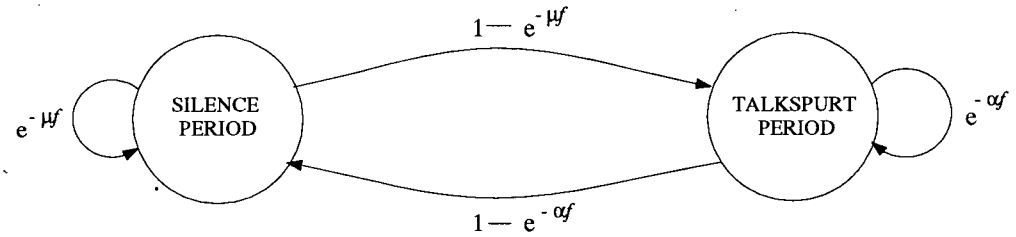


Figure 7. Active voice station model [9]

One sees that the transition probabilities of the above two-state Markovian model are highly depended on f. Figure 7 shows that the transitional probabilities increase exponentially as the time variable f increases and conversely for the non-transitional probabilities. The longer the time a call remains in one state, the greater the tendency to change. Since voice packets can tolerate losses, we assume that if a voice packet cannot be transmitted within a constant time period t, the voice packet will be discarded. Our choice of t=5.5 ms is to ensure compatibility with the air interface which has 5.5 ms frame length.

3.1.1 Voice clipping

Voice clipping occurs only if the number of ready-to-transmit voice stations within a frame period exceeds the number of slots/frame . For each active voice call in the model, we assume that talkspurt and silence periods alternate independently with exponential probability density functions $f(\tau)$ and $g(\tau)$ respectively, where α and μ are constants and

$$f(\tau) = \alpha e^{-\alpha\tau}$$

$$g(\tau) = \mu e^{-\mu\tau}$$

From these two probability density functions, we can determine the probability, P(t), that an active voice station is in talkspurt mode at any arbitrary time instant. With the help of P(t), we can then determine the voice clipping probability for a specific transmission system.

Since functions f and g are independent, the joint probability density $h(\tau_1, \tau_2) = f(\tau_1)g(\tau_2)$ and

$$\begin{aligned} P(t) = P(\tau_1 > \tau_2) &= \int_0^\infty \int_{\tau_2}^\infty h(\tau_1, \tau_2) d\tau_1 d\tau_2 \\ &= \int_0^\infty \int_{\tau_2}^\infty \alpha\mu e^{-(\alpha\tau_1 + \mu\tau_2)} d\tau_1 d\tau_2 \\ &= \alpha\mu \int_0^\infty e^{-\mu\tau_2} \left[\int_{\tau_2}^\infty e^{-\alpha\tau_1} d\tau_1\right] d\tau_2 \\ &= \mu \int_0^\infty e^{-(\mu+\alpha)\tau_2} d\tau_2 \\ &= \frac{\mu}{\alpha + \mu} \end{aligned}$$

Voice clipping occurs whenever more stations are in talkspurt mode than there are QA slots available. As a result, the voice clipping probability, P_c, for N active voice stations and k, slots/frame, available is as follows:

$$P_c = \begin{cases} \sum_{T=k+1}^{N} \binom{N}{T} P(t)^T (1 - P(t))^{N-T} & , N > k \\ 0 & , N \leq k \end{cases}$$

where $\binom{N}{T} P(t)^T (1 - P(t))^{N-T}$ is the probability of having T out of N number of voice stations in talkspurt mode.

Figure 8 shows the voice clipping probability versus the number of ready-to-transmit voice stations. For the values of constants listed earlier, P(t)=0.4, in which case k=2×2010 slots/frame/2 buses for a 5.5 ms frame length of 424 bit per slot at 155 Mbps transmission rate.

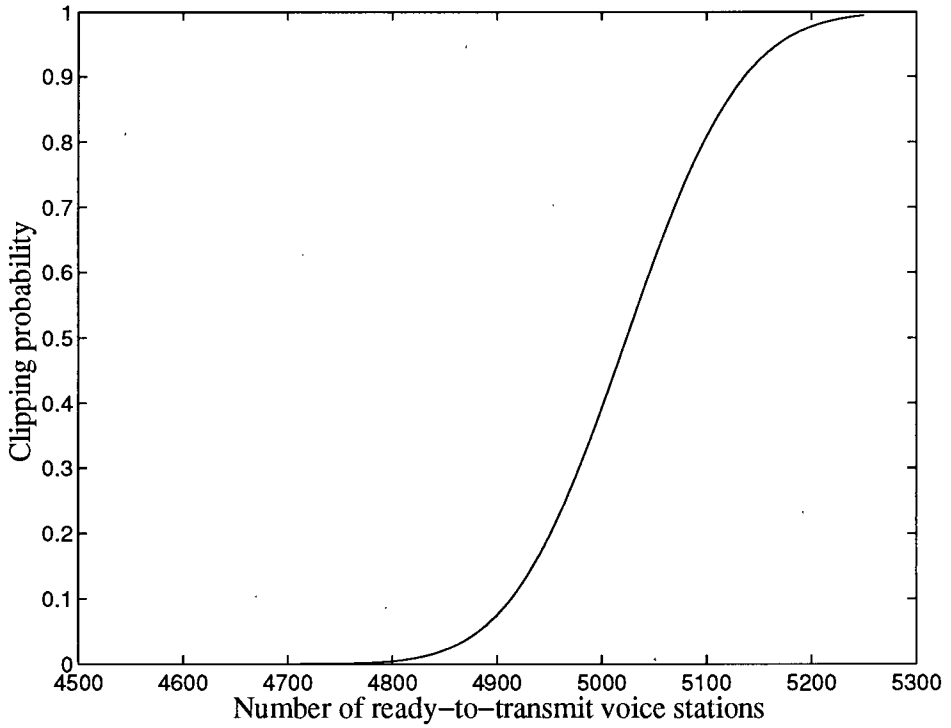


Figure 8. Voice clipping probability vs. number of ready-to-transmit voice stations

Section 3.2
Simulation Models

Voice and data traffic are assigned access levels priority 1 and 0, respectively. Priority 2, the highest level, is reserved for control signalling. Three priority levels are established by operating three queues, one for each level. Within each level, the performance characteristics described above are maintained. Waiting packets acquire access as soon as capacity becomes available; however, access priority is always given to packets in higher level queues.

Since voice packets are discarded if not be transmitted within 5.5 ms, the implementation of a DQDB counter mechanism will be sightly more complicated than for data only networks. According to the counters mechanism discussed in section 2.3.1, each node tracks the number of preceding and following requests for its downstream stations in each priority level. Therefore, if a voice packet at one particular node is discarded, its priority 1 countdown counter and request transmission for the next packet cannot just simply be reset. For example, let us consider two nodes, α and β, with β located downstream from α. Assume that there is only one voice packet in node α and several voice packets in node β awaiting transmission. Consider node α at the instant when no following request (RQC=0) is received for voice traffic from downstream stations and the priority level 1 CDC is reset because its last voice packet is discarded; then its next transmission request will be the request for the data packet which located at the head of the data packet queue, since node α does not have any voice packet awaiting transmission. Possibly, node α may transmit its requested data packet while node β still has voice packets waiting for transmission since the priority 1 level CDC is reset and RQC is equal to zero in node α. Consequently, the priority accessing scheme is violated. In order to preserve priority accessing, preceding requests, if any, will be added to the following requests (i.e. the value of CDC is added to RQC) for each discarded packet in each priority level before the next packet makes request for transmission. Hence, the problem of the above example will be solved.

In our model, all voice call destinations are predetermined and are uniformly distributed among destination nodes. Since voice packet delay is our primary concern, all voice call connections are assumed to be established at the beginning of any simulation. Moreover, each voice call will remain active throughout each simulation run. Except for the differences in parameter values, the following assumptions apply to all simulation models. These models are

developed for two way traffic for more realistic analysis; packets are sent on bus A or B, as appropriate:

i. The transmission rate on each bus is 155 Mbps
ii. Slot length is 53 octets (424 bits)
iii. Slot transmission time is 2.73 μs
iv. Nodes are evenly distributed along the bus
v. Successive nodes are separated by 3 slots
vi. Twenty buffers are available at each node for data packets
vii. All packet lengths, including voice and data, equal 1 slot in duration
viii. Inter-arrival time of data packet at each node is exponentially distributed
ix. Each node has the same Poisson arrival rate
x. The number of nodes including two head stations equals 50
xi. Each node has the same number of active voice calls
xii. Priority is given to voice packets
xiii. Any voice packet that cannot be transmitted within 5.5 ms will be discarded

Simulation models are written in **SIMSCRIPT II.5** [7]. Statistics are compiled after an initial period of 10,000 slots. Analysis is performed for various values of the following parameters: (1) speech digitizing rate of voice stations, (2) percentage of system bandwidth available for QA access, and (3) activation time for voice calls.

Section 3.3
Simulation Results

3.3.1 Simultaneous voice calls arrival without silence deletion

The simulation model used here is based on the assumption that all voice calls arrive simultaneously at the beginning of the frame and remain in the talkspurt period throughout the simulation (without silence deletion). The voice coding rate is 64 kbps; thus each voice station generates a voice packet every 6.0 ms.

Figure 9 shows the mean waiting delay[1] for a voice packet versus the QA voice throughput[2]. Since all voice packets enter MAN nodes at the same time, an increase in M, the number of voice packets, transmitted results initially in a proportional increase in the mean waiting delay. As M increases further, the mean voice packet waiting delay remains at 3 ms because further increase in the number of voice calls (voice packets) will only increase the number of discarded voice packets; these are packets cannot be transmitted within the 5.5 ms time limit as shown in Figure 10. There will not be any significant change in either the waiting delay or throughput as M increases beyond some limit. Figure 10 shows that the number of voice calls that the network can accommodate is about 3600 within an acceptable value of 2% discarded voice packets under the assumption of simultaneous voice calls arrival without silence deletion.

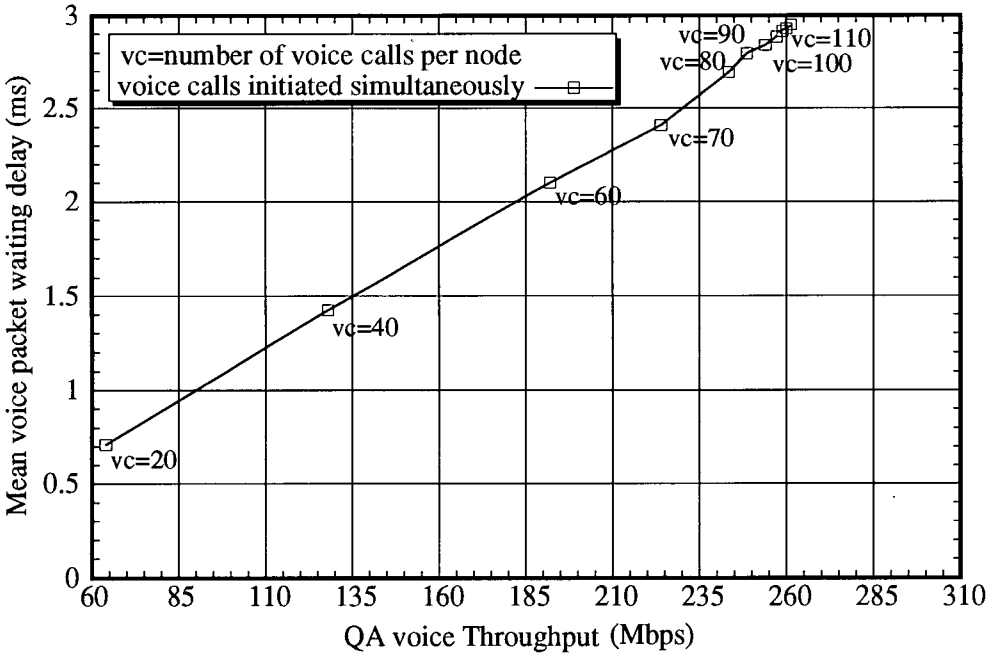


Figure 9. Mean voice packet waiting delay vs. QA voice Throughput

---

1 waiting delay is calculated from the time a packet enter a node until it is transmitted.

2 the voice throughput is calculated only for the segment payload excluding the overheads.

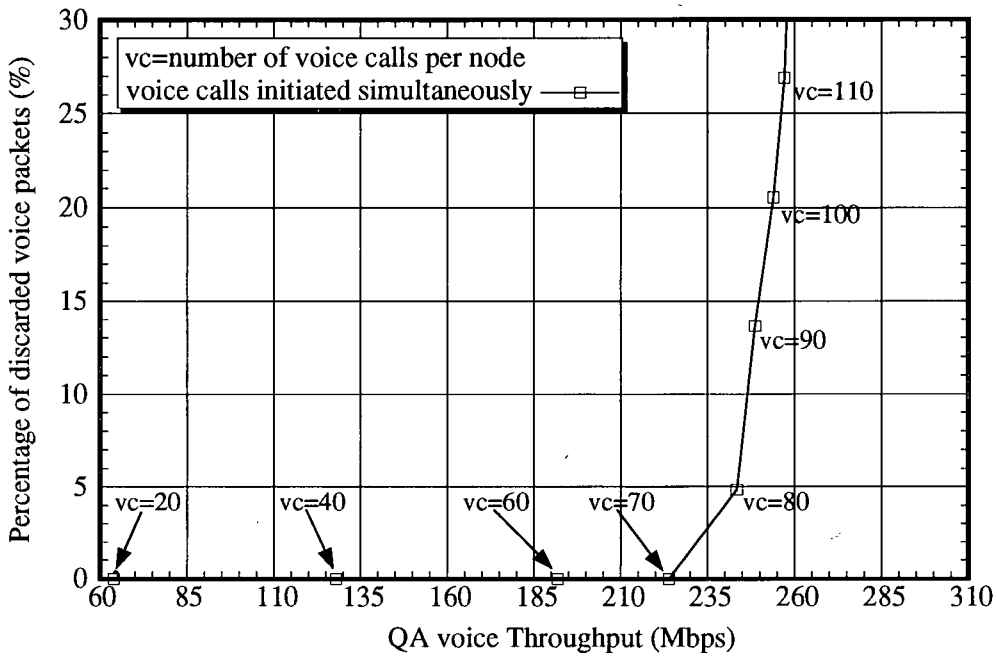


Figure 10. Percentage of voice packets discarded vs. QA voice Throughput

Figure 11 shows the mean data packet waiting delay versus the QA data throughput for different voice throughput levels. Voice calls are initiated simultaneously at the beginning of the simulation. As expected, data packet waiting delay increases as the network's voice throughput increases for the same data throughput.

Figure 12 shows a typical example of the mean waiting delay of data and voice packets with respect to the nodes' locations. Node 1 and 50 are the head of bus (HB). One sees that both voice and data packet mean waiting delay varies among the nodes. Nodes located near HBs have less delay than nodes located near the buses midpoints.

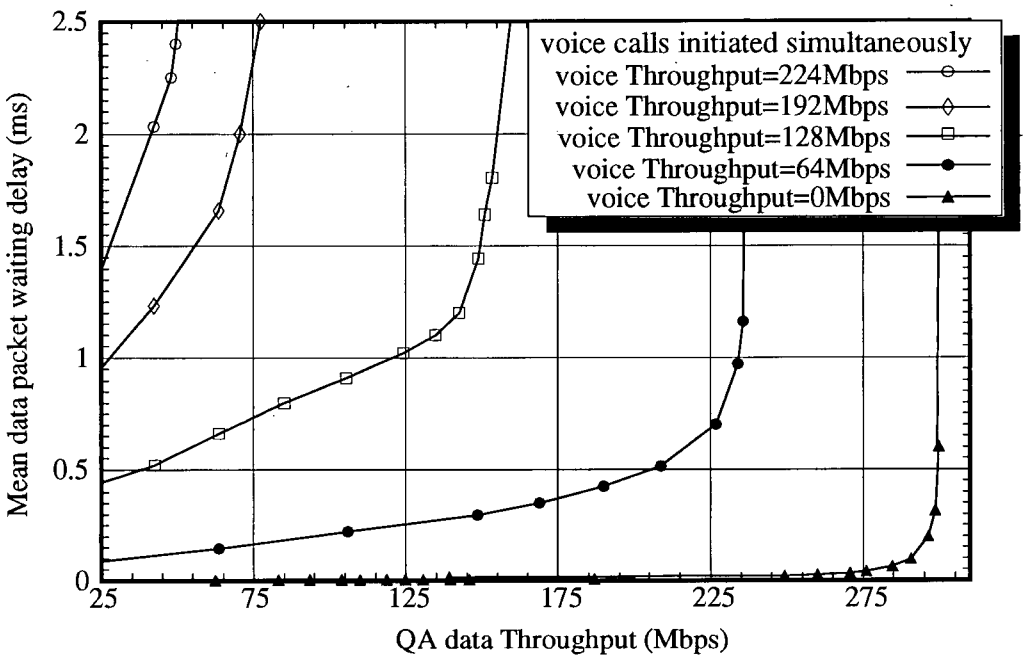


Figure 11. Mean data packet waiting delay vs. QA data Throughput for various voice throughput

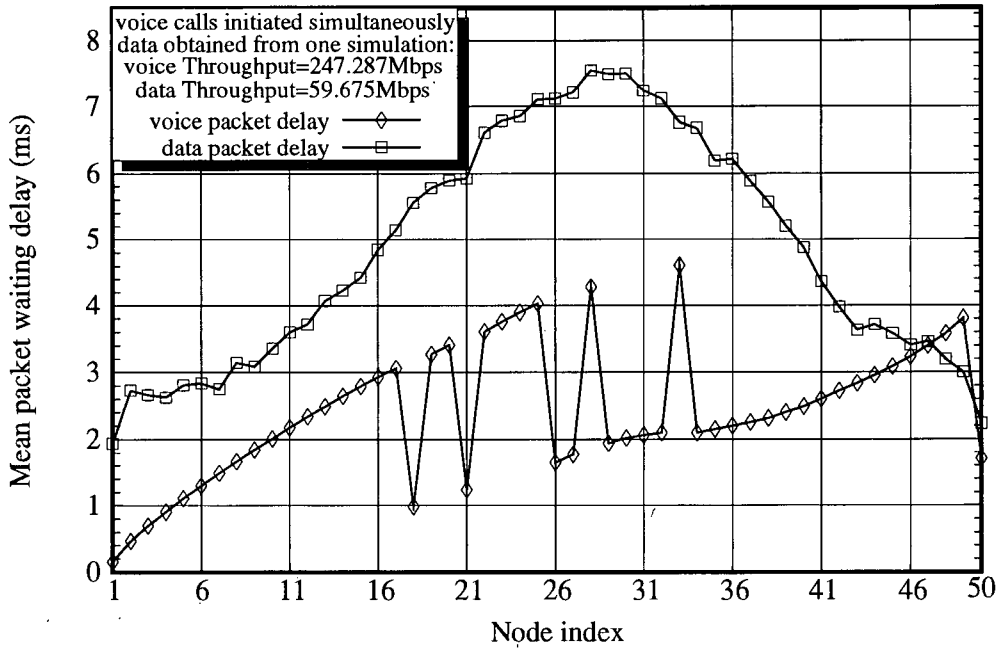


Figure 12. Mean packets waiting delay vs. nodes locations

Slots are generated from HBs travel along the buses unidirectionally; nodes located near HBs (upstream nodes) therefore will have a better chance for bus access than will nodes located at the bus midpoint.

3.3.2 Exponentially distributed voice call arrivals without silence deletion

The simulation model used here is based on the same assumptions as in the previous section except that the voice call arrival times are Poisson. Time separation occurs between successive voice calls, and improvements in both voice and data waiting delay are expected as a result.

Figure 13 shows the mean voice packet waiting delay versus the QA voice throughput for both simultaneous and exponential voice call arrival models. The Poisson arrivals result in large improvements in mean waiting delay relative to the delay for simultaneous arrivals. Also, the QA voice throughput achieved under Poisson arrivals is higher than the simultaneous voice call arrival model because more voice packets are transmitted within the 5.5 ms time limit as the number of voice calls increases. The mean voice packet waiting delay limits at 5.5 ms under Poisson arrivals, as expected.

Figure 14 shows the percent discarded voice packets versus throughput. One sees that the number of voice calls that the network can accommodate is approximately 4200 within an acceptable 2% discarded rate. Under Poisson arrivals, the network can accommodate approximately 600 more voice calls than with the simultaneous call arrival model.

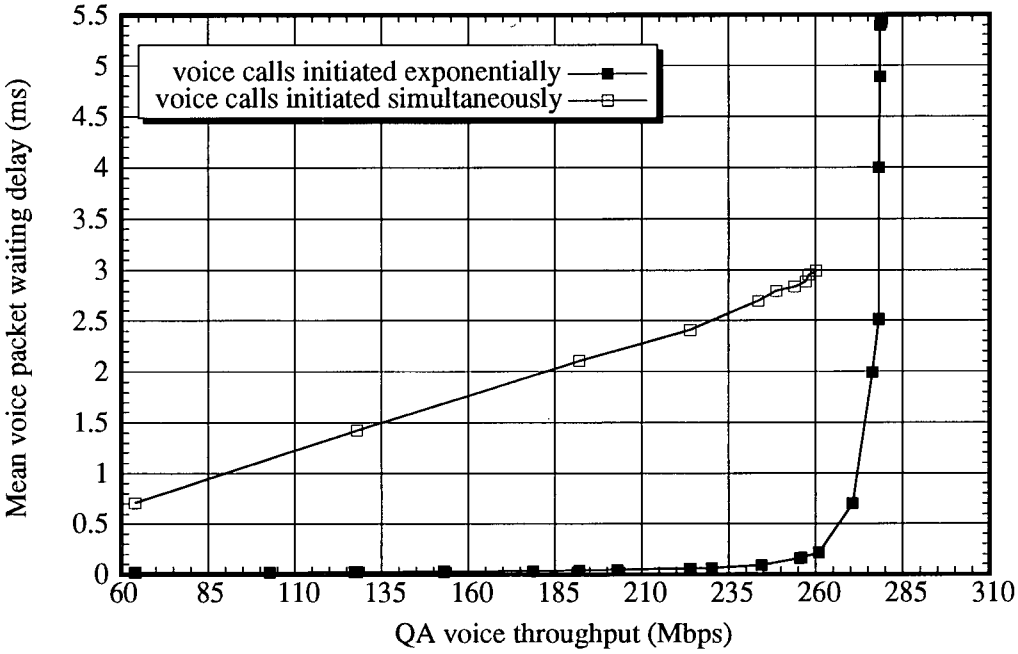


Figure 13. Mean voice packet waiting delay vs. QA voice Throughput

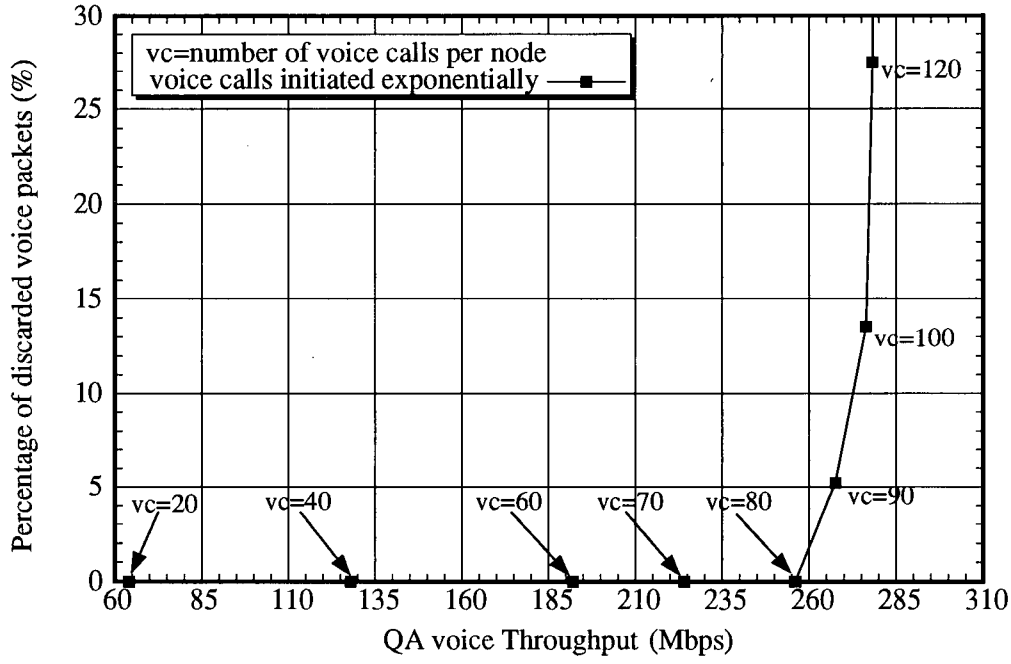


Figure 14. Percentage of discarded voice packet vs. QA voice Throughput

Figure 15 shows the mean data packet waiting delay versus data throughput for different voice throughput levels for both simultaneous and exponential voice call arrival models. The mean waiting delay is lower for the exponentially distributed voice call arrivals. However, both models result in the same (asymtotic) maximum data throughput levels.

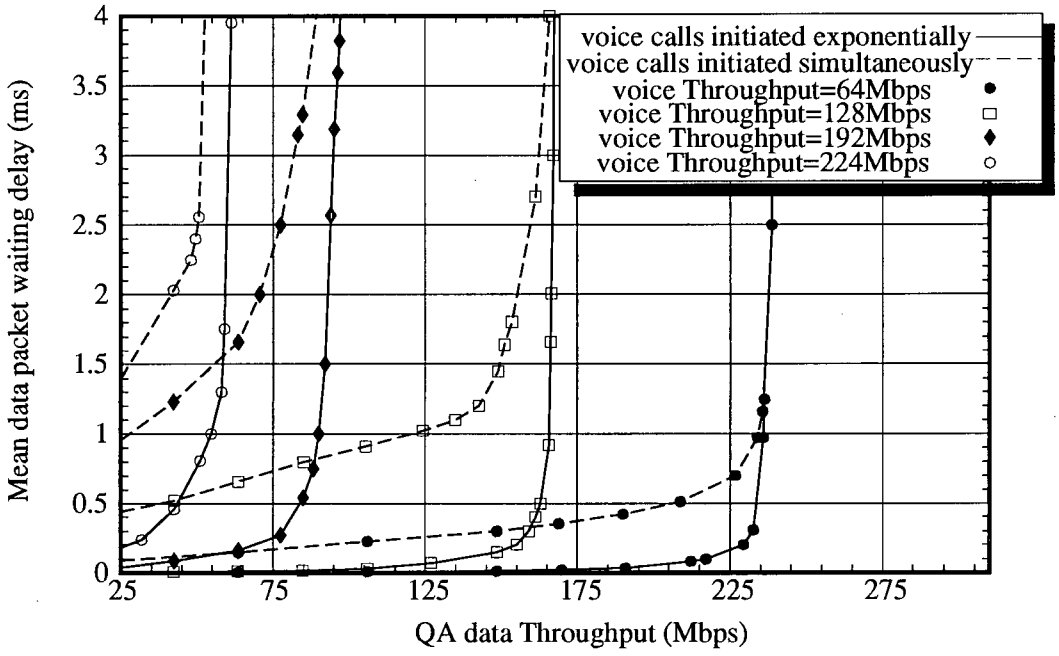


Figure 15. Mean data packet waiting delay vs. QA data Throughput for various voice Throughput

Figure 16 shows that the mean packet waiting delay varies among nodes. Upstream nodes have more rapid access to the bus than do nodes located in the middle of the bus.

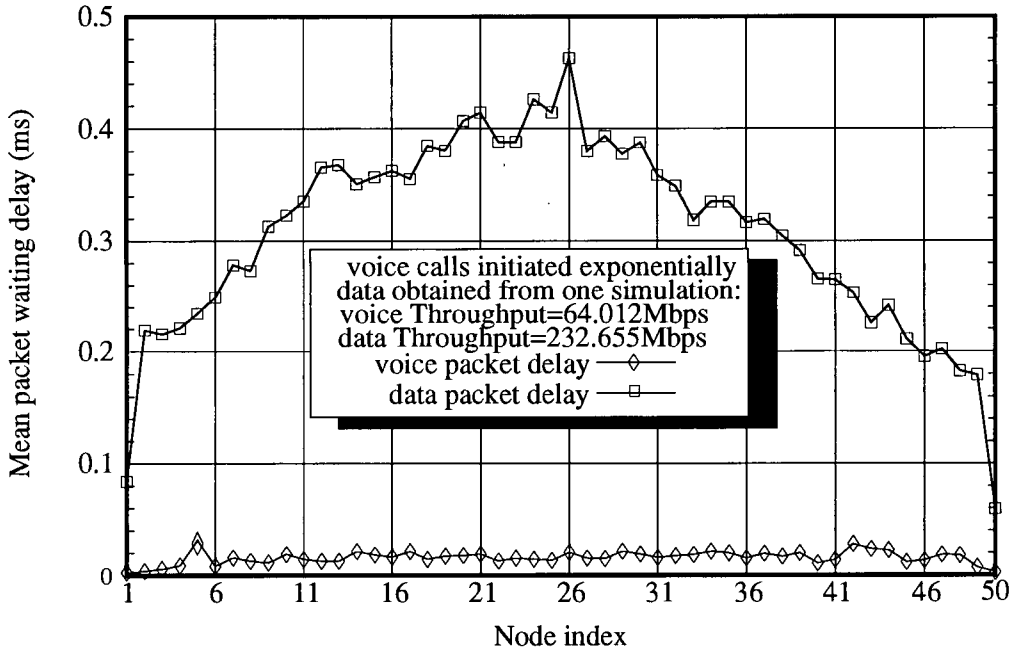


Figure 16. Mean packet waiting delay vs. node location

The results of the simultaneous and exponential voice call arrivals without silence deletion are summarized in Table 4 below.

Number of voice calls in the network	QA voice throughput achieved (Mbps)		Mean voice packet waiting delay (ms)		Percentage of discarded voice packets (%)	
	exponential voice calls arrival	simultaneous voice calls arrival	exponential voice calls arrival	simultaneous voice calls arrival	exponential voice calls arrival	simultaneous voice calls arrival
1000	64.012	64.012	0.018	0.711	0	0
2000	127.996	127.996	0.022	1.424	0	0
3000	192	192	0.041	2.102	0	0
3500	223.986	224.014	0.057	2.409	0	0
4000	255.992	243.527	0.163	2.694	0	4.86
4500	267.688	248.608	0.546	2.798	5.22	13.64
5000	276.375	254.087	1.992	2.886	13.55	20.55

Table 4. Results summary of simultaneous and exponential voice calls arrival without silence detection

The results of this section shows clearly the need for simulations which include the effects of random call arrivals.

3.3.3 Exponentially distributed voice call arrivals with silence deletion

In this section, results using a simulation model with Poisson voice call arrivals with silence deletion are presented, to show how silence deletion affects networking performance. In the previous two models, all voice calls remained active and in talkspurt mode throughout the simulations. Now we assume that all voice calls are remain active throughout the simulation but that each voice call alternates independently between a talkspurt and a silence period, with exponentially distributed random durations of means 1.5 s and 2.25 s, respectively. Voice packets are generated every 6.0 ms (voice coding rate=64 kbps) from those voice calls in a talkspurt period. As shown previously in this chapter, the probability that an active voice call is in talkspurt mode at an arbitrary time instant equals 0.4; therefore, we assume that 40% of all voice calls are in talkspurt mode and that the remaining 60% are in silence mode at the beginning of each simulation. Each voice call alternates between silence and talkspurt according to time intervals randomly chosen from two independent exponentially distributed random distributions.

Silence deletion prevents a voice call from generating a packet for a specific period of time and will not have any effect on the mean packet delay for any given QA voice (talkspurt) throughput value. Consequently, the mean voice and data packet waiting delays are identical to those obtained earlier without silence deletion. Table 5 shows some examples of the number of voice calls needed to achieve listed QA voice throughputs with their respective mean voice packet waiting delays for exponentially distributed voice call arrivals with and without silence detection.

QA voice throughput (Mbps)	Number of voice calls needed to achieve the throughput		Mean voice packet waiting delay (ms)	
	with silence deletion	without silence deletion	with silence deletion	without silence deletion
127	5000	2000	0.022	0.022
192	7500	3000	0.039	0.041
256	10000	4000	0.159	0.163

Table 5. Comparison between models with and without silence deletion

It is almost impossible to achieve an exact match on the QA voice throughputs for a given number of voice calls for the two models used. The matching between the QA voice throughput and corresponding number of voice calls in Table 5 is a good approximation. The mean waiting delays for two cases are almost equal. Therefore, we can conclude that silence deletion will not affect the mean voice and data packet waiting delay with respect to QA voice throughput.

On the contrary, the number of voice calls that the network can accommodate with silence deletion must be larger than the one without silence deletion. The network can accommodate approximately 10500 voice calls with an acceptable 2% discarded voice packets with silence deletion, as shown in Figure 17. This is more than double in comparison to the case without silence deletion.

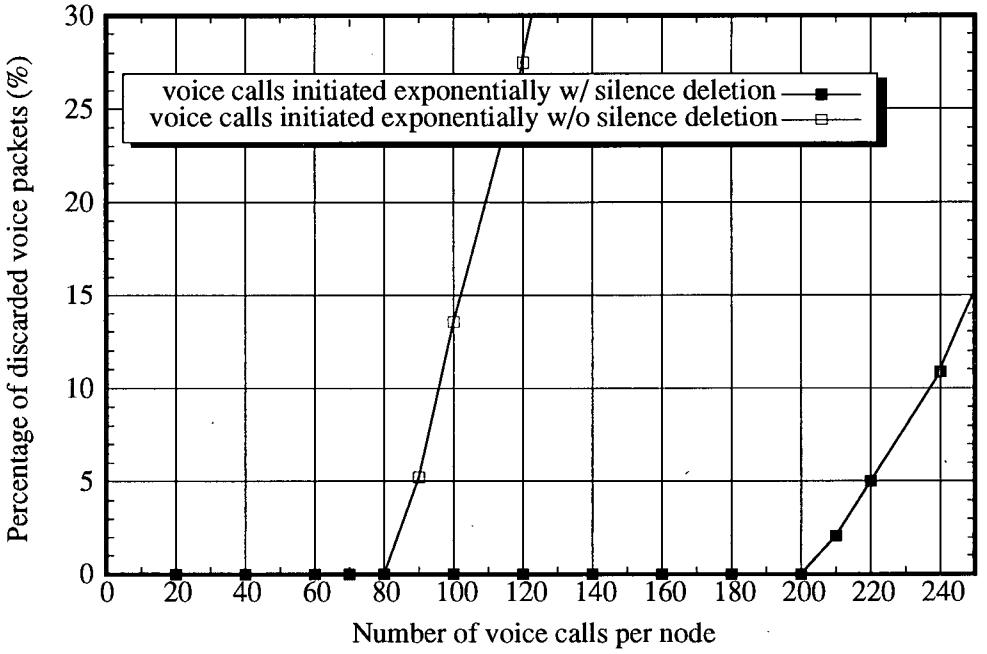


Figure 17. Percentage of discarded voice packets vs. number of voice calls per node

3.3.4 Exponentially distributed voice calls arrival for different voice coding rate with silence deletion

In this section, we consider the effect of voice coding rate on voice and data packet delay in the network. The simulation models are basically the same as the one in previous section. Each node has the same number of voice calls with Poisson arrivals. All voice calls are assume to be active throughout the simulations and they alternate between talkspurt and silence period independently with exponentially distributed random durations of means 1.5 s and 2.25 s, respectively. As before, 40% of the randomly chosen voice calls are in talkspurt and the other 60% in silence at the beginning of each simulation.

However, the time between two successively generated voice packets at a voice station is not fixed at 6.0 ms (64 kbps) because we are varying the voice coding rate of the voice station. Therefore, the time between two successive voice packets in a voice station is 6.0 ms for 64

kbps voice coding rate, 12.0 ms for 32 kbps voice coding rate, 24.0 ms for 16 kbps voice coding rate and so on. As before, voice packets not transmitted within 5.5 ms will be discarded.

Figures 18 to 21 show the mean data packet waiting delay versus the QA data throughput for different numbers of active voice calls and different voice coding rates. These curves show clearly that a reduction in the voice coding rate leaves more capacity for data packet transmission.

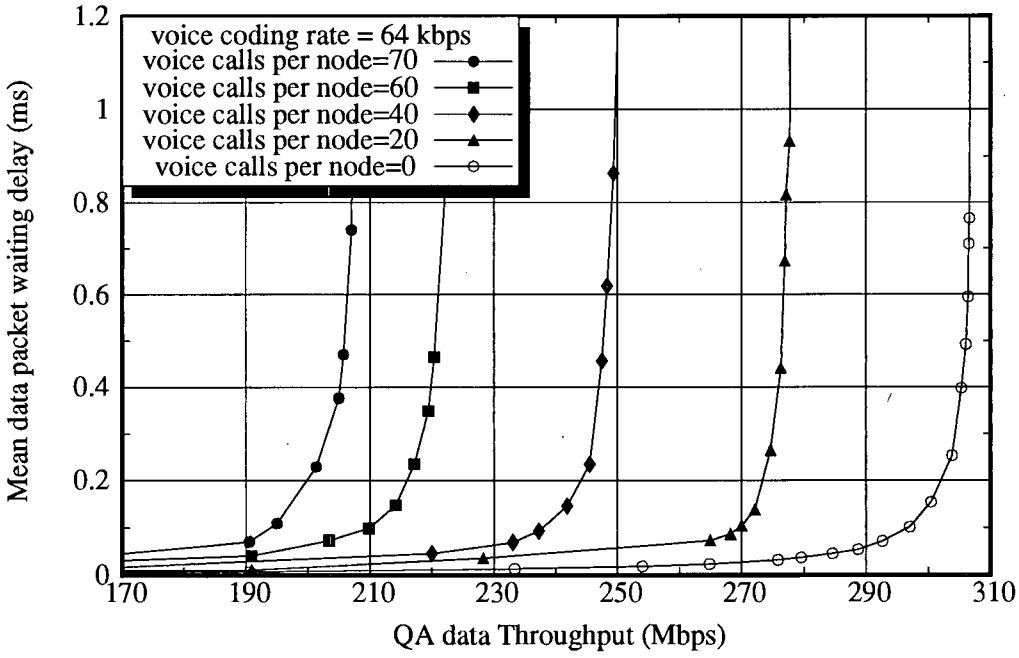


Figure 18. Mean data packet waiting delay vs. QA data Throughput for various number of active voice calls with voice coding rate of 64 kbps

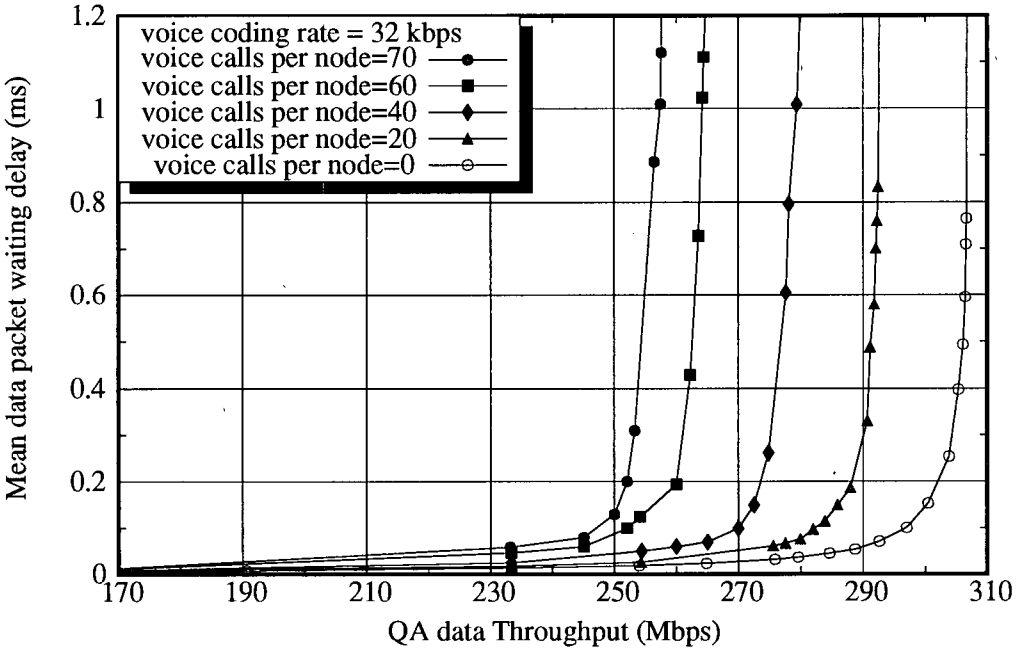


Figure 19. Mean data packet waiting delay vs. QA data Throughput for various number of active voice calls with voice coding rate of 32 kbps

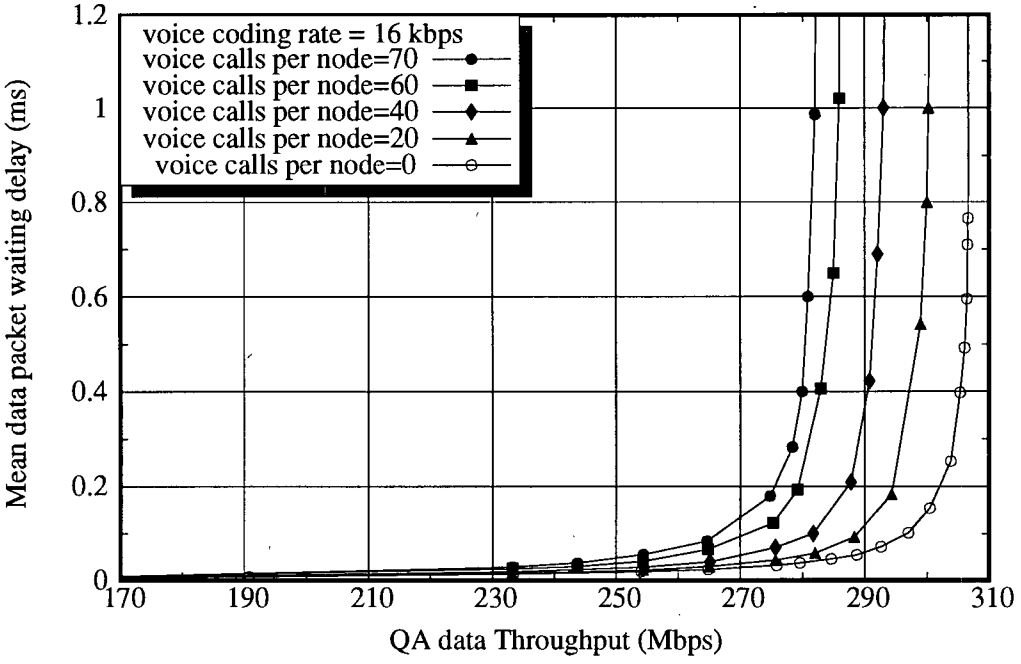


Figure 20. Mean data packet waiting delay vs. QA data Throughput for various number of active voice calls with voice coding rate of 16 kbps

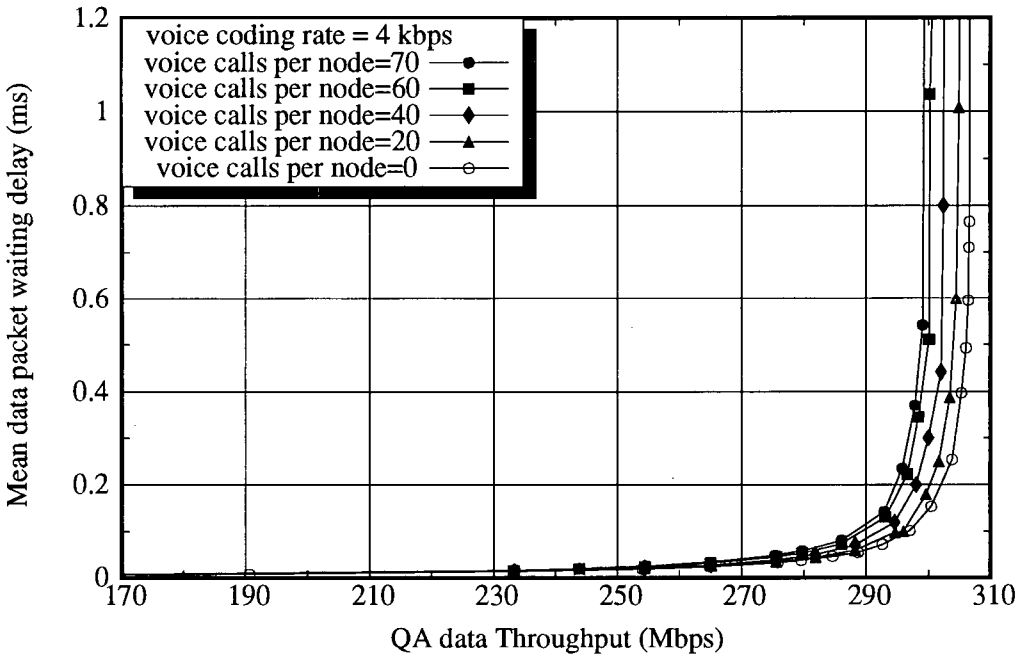
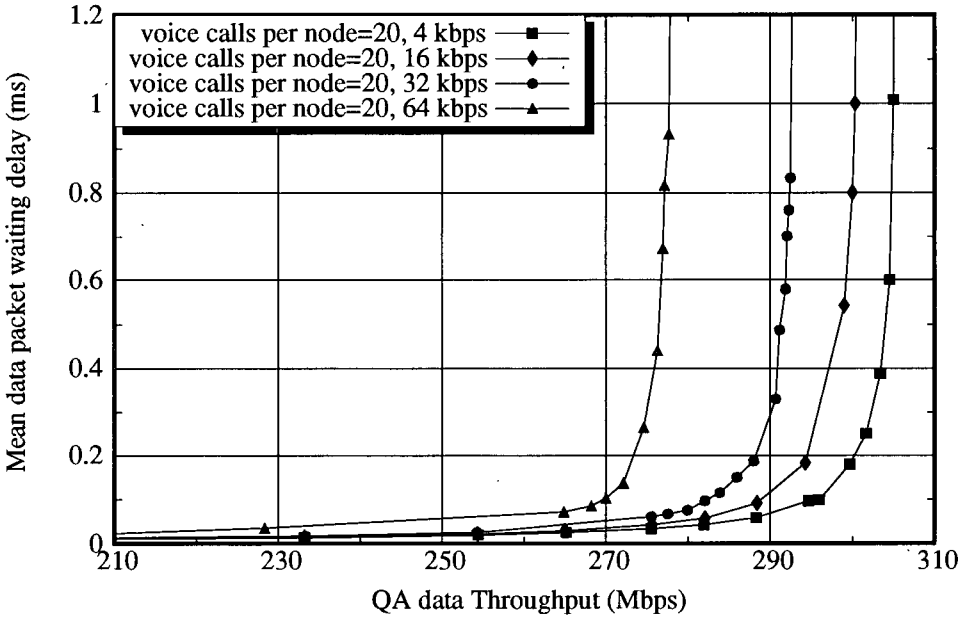


Figure 21. Mean data packet waiting delay vs. QA data Throughput for various number of active voice calls with voice coding rate of 4 kbps

Figure 22 shows the mean data packet waiting delay versus the QA data throughput for different voice coding rates with 20 and 70 active voice calls activated per node. Reduction of the voice coding rate from 64 to 4 kbps increases the maximum QA data throughput from 280 to 305 Mbps. With 70 active voice calls per node, the QA data throughput increases from 200 to 300 Mbps as the voice coding rate is reduced from 64 to 4 kbps.

(a) 20 voice calls per node



(b) 70 voice calls per node

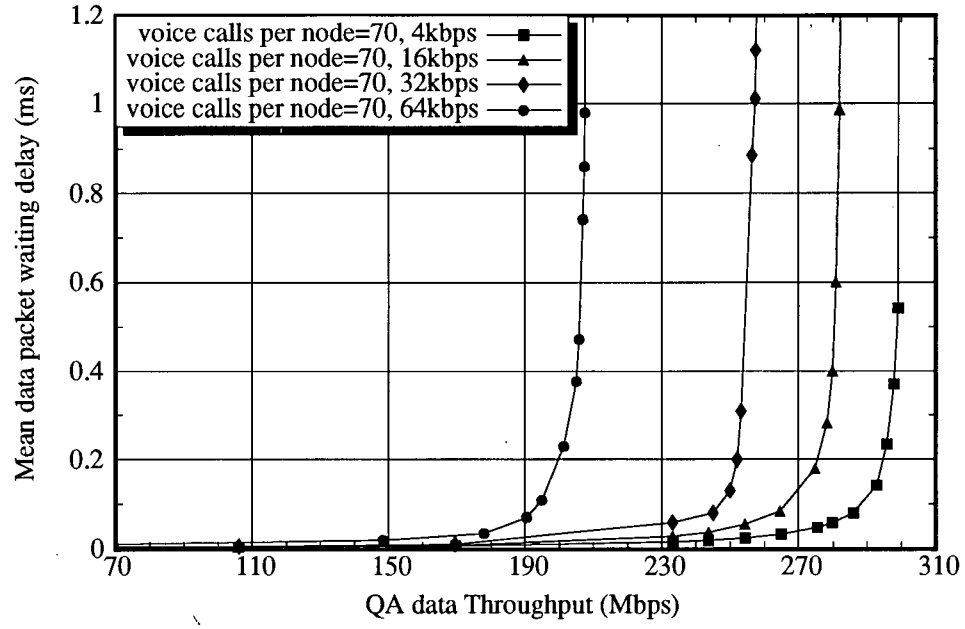


Figure 22. Mean data packet waiting delay vs. QA data Throughput with different numbers of active voice calls per node for various voice coding rates

There is a simple relationship between the voice coding rate and the number of active voice calls, as follows, where R_i denotes the voice coding rate, N_i denotes the number of active voice calls and C is a constant:

$$R_1 N_1 = R_2 N_2 = C$$

From this equation, one sees that the mean data packet waiting delay curve for 64 kbps voice and 20 voice calls per node is the same as the mean data packet waiting delay curve for 32 kbps voice and 40 voice calls per node. Both configurations give the same QA voice throughput in the network and the corresponding curves of mean data packet delay versus QA data throughput overlay each other, as shown in Figure 23. Similar comments apply for other sets of curves. Simulation results of the QA voice throughput achieved for different voice coding rates and number of voice calls activated are summarized in Table 6.

number of voice calls	QA voice Throughput achieved (Mbps)			
	64 kbps	32 kbps	16 kbps	4 kbps
1000	25.970	12.771	6.345	1.657
2000	51.659	25.769	12.830	3.229
3000	78.246	38.524	19.625	4.913
3500	89.617	44.536	22.376	5.784

Table 6. QA voice Throughput achieved for various voice coding rates and number of active voice calls

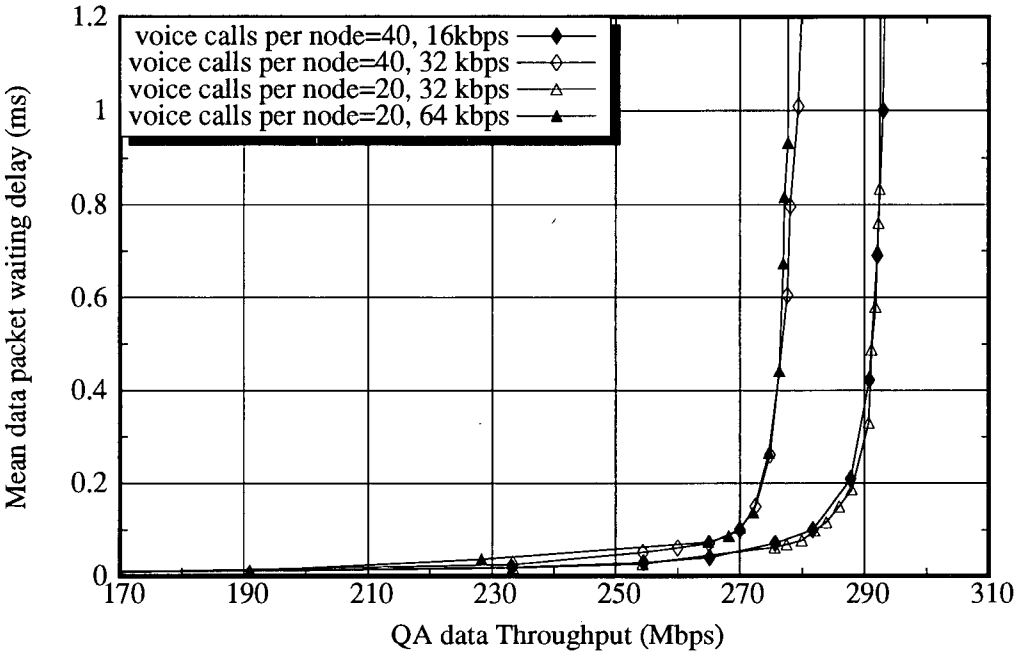


Figure 23. Relation between voice coding rate and number of active voice calls activated

Because the voice coding rate at the network stations primarily affects the QA voice throughput achieved in the network, the maximum number of voice calls a network can accommodate within an acceptable 2% of discarded voice packets range may vary with voice coding rate. The network can hold approximate 21000 voice calls for 32 kbps voice coding rate, 42000 voice calls for 16 kbps voice coding rate and 168000 voice calls for 4 kbps voice coding rate. These numbers are based on the assumptions of our simulation models.

Since the voice coding rate limits the QA voice throughput achieved in the network for a fixed number of active voice calls, the mean voice packet waiting delay versus the QA voice throughput remains the same as before and is as shown in Figure 13. However, for a fixed number of voice calls activated in the network, the time from when a voice station encodes a voice cell to the time of decoding at the destination may vary substantially with voice coding

rate. Table 7 shows two typical examples of 1000 active voice calls (20 voice calls per node for 50 nodes) and 3500 active voice calls in the network.

number of voice calls in network	voice coding rate (kbps)	voice cell encoding time (ms)	mean voice packet waiting delay (ms)	propaga-tion delay (ms)	voice cell decoding time (ms)	total elapse time (ms)
1000	64	6.0	0.012	t	6.0	12.012+t
	32	12.0	0.010	t	12.0	24.010+t
	16	24.0	0.010	t	24.0	48.010+t
	4	96.0	0.009	t	96.0	192.009+t
3500	64	6.0	0.016	t	6.0	12.016+t
	32	12.0	0.013	t	12.0	24.013+t
	16	24.0	0.011	t	24.0	48.011+t
	4	96.0	0.009	t	96.0	192.009+t

Table 7. Delay comparison for different voice coding rate voice stations

As shown in Table 7, the voice coding rate of a voice station plays a very important role in the total elapse time for transmitting a voice cell since the propagation delay, t, is almost constant for all cases in uniform destination distribution and the variation of mean voice waiting delay is very small compared to the encoding/decoding time.

Chapter 4. Delay Analysis of QA Access for Inter- and Intra-MAN Voice and Data Integrated Traffic within a Backbone MAN Network

In this chapter, voice and data packet transmission delays will be analyzed for both inter- and intra-MAN traffic. Inter-MAN traffic is that flowing on two or more DQDB MANs within a backbone MAN network; intra-MAN traffic is confined to one access DQDB MAN only. The number of homogeneous bridges connecting each access MAN depends on its physical location within the backbone MAN network, as illustrated in Figure 24.

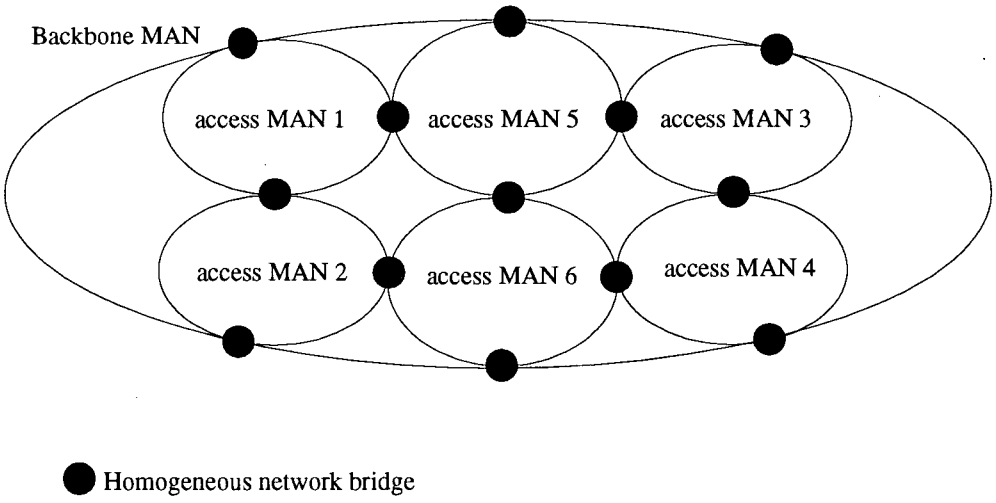


Figure 24. Geographical location of access MANs within a backbone MAN

In Figure 24, six access MAN are connected to adjacent MANs and to a backbone MAN. Four homogeneous bridges connect both MAN 5 and MAN 6 which are located in the middle of the MAN cluster. Each of the other four access MANs has three homogeneous connecting bridges. According to this connection scheme, any inter-MAN traffic need only go through two homogeneous bridges, at most, in order to reach the destinated MAN within a backbone MAN network. Details of the homogeneous bridges connection scheme are provided next.

Section 4.1
Homogeneous Bridges Connection Scheme

From the previous chapter we know that the upstream nodes have a transmission advantage over the downstream nodes; therefore, bridge connections are arranged according to this property in order to reduce the overall delay, including bridge delay.

For MANs 1 to 4, two of the three bridges connect to the two neighborhood MANs while the third bridge is used to direct packets to the remaining three MANs via the backbone MAN. The bridge connected to the backbone MAN will have a heavier traffic load than the other two bridges provided that the inter-MAN traffic is uniformly distributed among the six access MAN. Therefore, we connected the bridge with the heaviest traffic load as the head node of the access MAN, as shown in Figure 25.

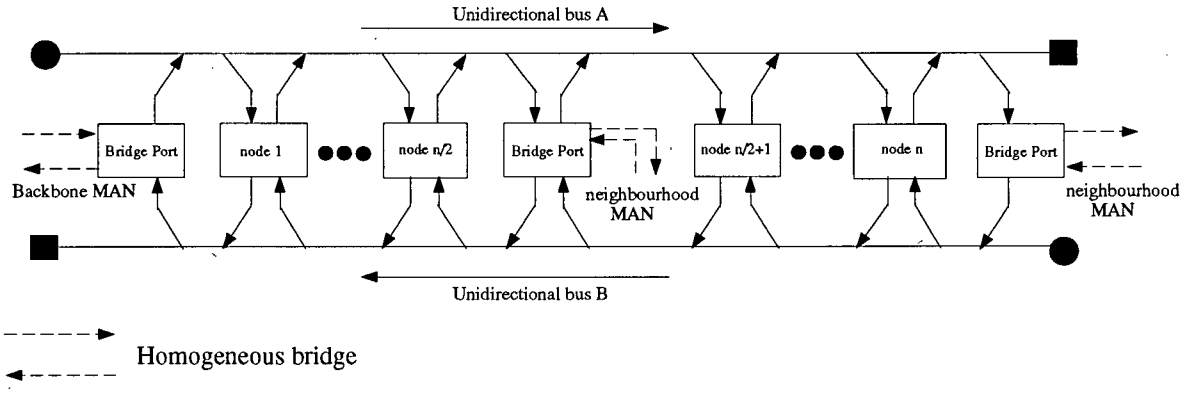


Figure 25. Illustration of three homogeneous bridges connection

For MAN 5 and MAN 6, three of the four bridges are connect to the three neighborhood MANs and the fourth bridge is used to deliver packets to the other two MANs via the backbone MAN. With four bridges, we can actually put one more bridge port into the access MAN or simply connect the fourth bridge to either one of the other three bridge ports. In our connection scheme, we implemented the latter scheme, for simplicity. Since upstream nodes provide a transmission

advantage, the fourth bridge is connected to the head bridge port for the neighboring MAN. As a result, two head bridge ports will be responsible for delivering inter-MAN traffic destinated to four access MANs (each bridge port for two access MANs), as illustrated in Figure 26.

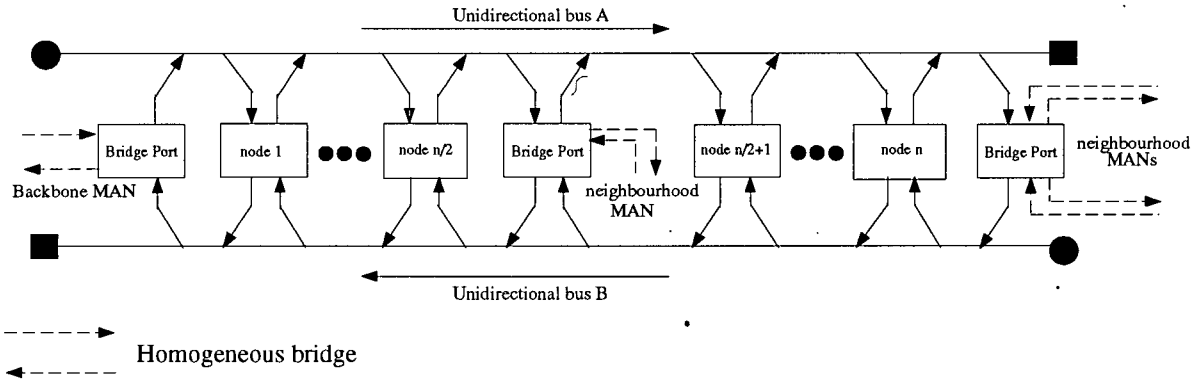


Figure 26. Illustration of four homogeneous bridges connection

Each bridge port functions as a node, with the same nodal media access mechanism. Twenty buffers were used for each priority level in each bridge port in our model. In our simulations, the processing delay and the propagation delay of the bridges are neglected. Each of these delays would typically be less than 0.25 ms. Figure 27 shows the physical arrangement and bridge connections of six access MANs within a backbone MAN network in our model.

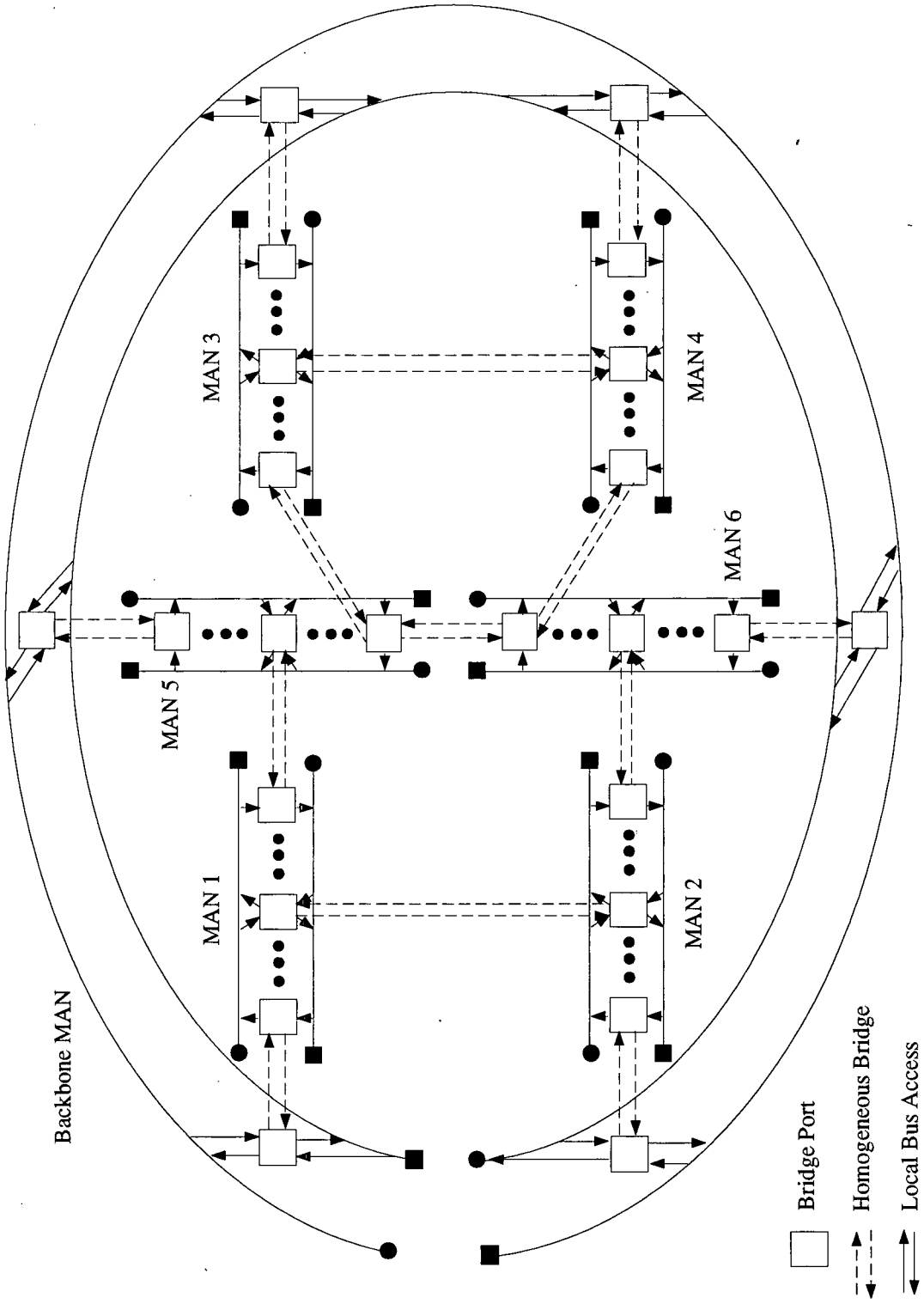


Figure 27. Physical representation of a backbone MAN network

Section 4.2
Simulation Model

Each node of each access MAN within the backbone MAN network is basically the same as the previous models except that now a predetermined level of voice calls and data packets in each node will be inter-MAN traffic. Inter-MAN traffic occurs at random. The destination node and the destination MAN for all inter-MAN traffic of each access MAN is uniformly distributed among the two hundred and fifty nodes in the other five access MANs. Since the DQDB QA protocol allows only three priority access levels and since the highest priority level is reserved for control signalling, inter-MAN and intra-MAN traffic priorities are equal. All assumptions from the previous simulation models apply for each access MAN within the backbone MAN network. Inter-MAN voice call destination are paired. For example, a voice call initiated in MAN 1, node 26 with destination at MAN 5, node 4 must be paired with a voice call from MAN 5, node 4 to MAN 1, node 26. All voice calls are processed to have silence deletion with a steady-state talkspurt probability of 0.4.

Each of the six backbone MAN bridge port functions as an access MAN node. Sixty time slots separate successive bridge ports (nodes) in the backbone MAN. Twenty buffers serve each priority level at each backbone MAN node. An extremely high level of inter-MAN traffic will cause some lost packets, because of limited buffer queue capacity.

The source code of the backbone MAN network simulation model is provided in Appendix A. The model is written in **SIMSCRIPT II.5** [7].

Section 4.3
Preliminary Simulation Results

In a preliminary analysis, the packet mean delay of inter-MAN and intra-MAN packets was

investigated for a fixed percentage of inter-MAN voice and data traffic. Voice coding rates of 32 and 4 kbps were used, with 50 voice calls activated in each node. A total of 300 × 50 = 15000 voice calls were active at any time in the six-MAN network.

Figures 28 and 29 show the mean inter- and intra-MAN data packet transmission delay[3] versus the total QA data traffic[4] for 32 kbps voice coding. Figures 30 and 31 show the mean inter- and intra-MAN data packet transmission delay versus the total QA data traffic for 4 kbps voice coding.

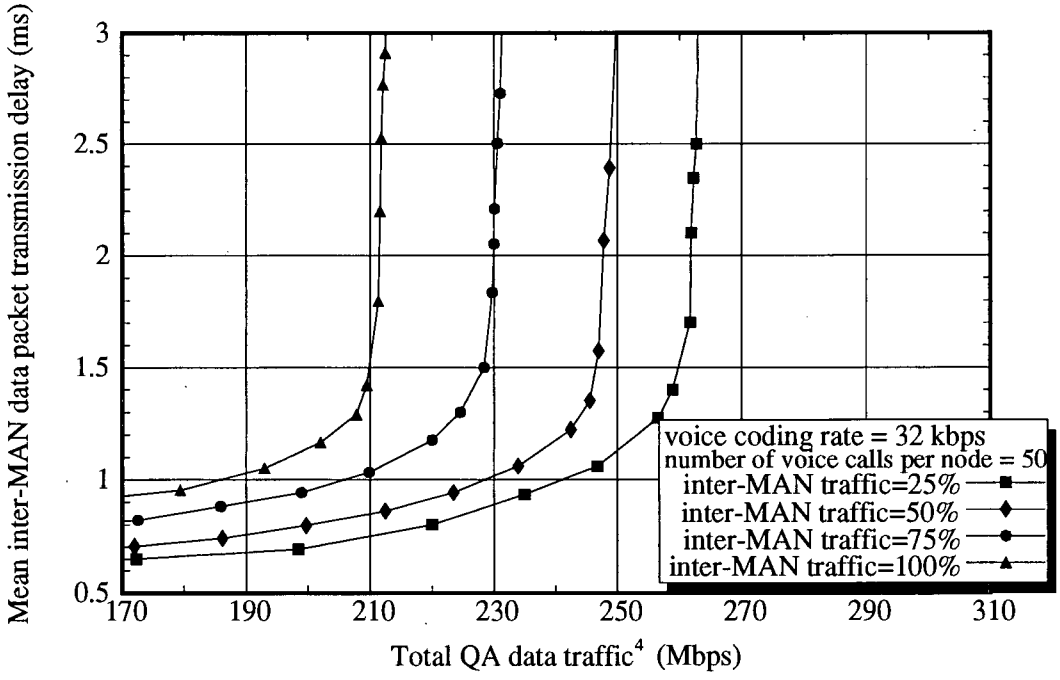


Figure 28. Mean inter-MAN data packet transmission delay vs. total QA data traffic for 32 kbps voice coding rate

---

3 Mean transmission delay is calculated from the time a packet enters the buffer of a node until it reaches the destinated node.

4 Total QA data traffic is the sum of inter- and intra-MAN data traffic per single access MAN.

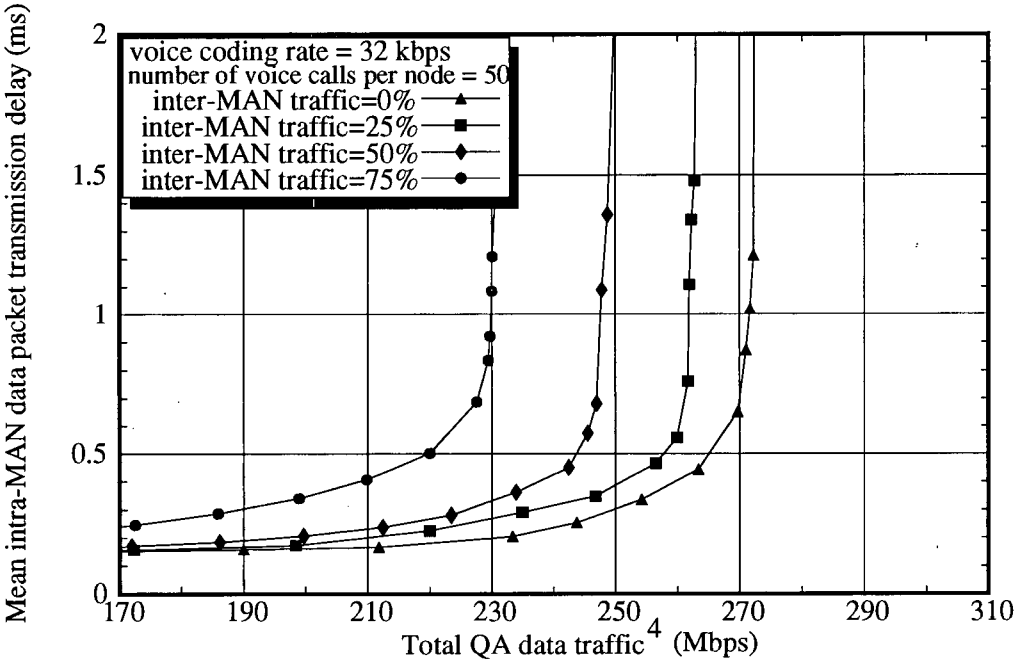


Figure 29. Mean intra-MAN data packet transmission delay vs. total QA data traffic for 32 kbps voice coding rate

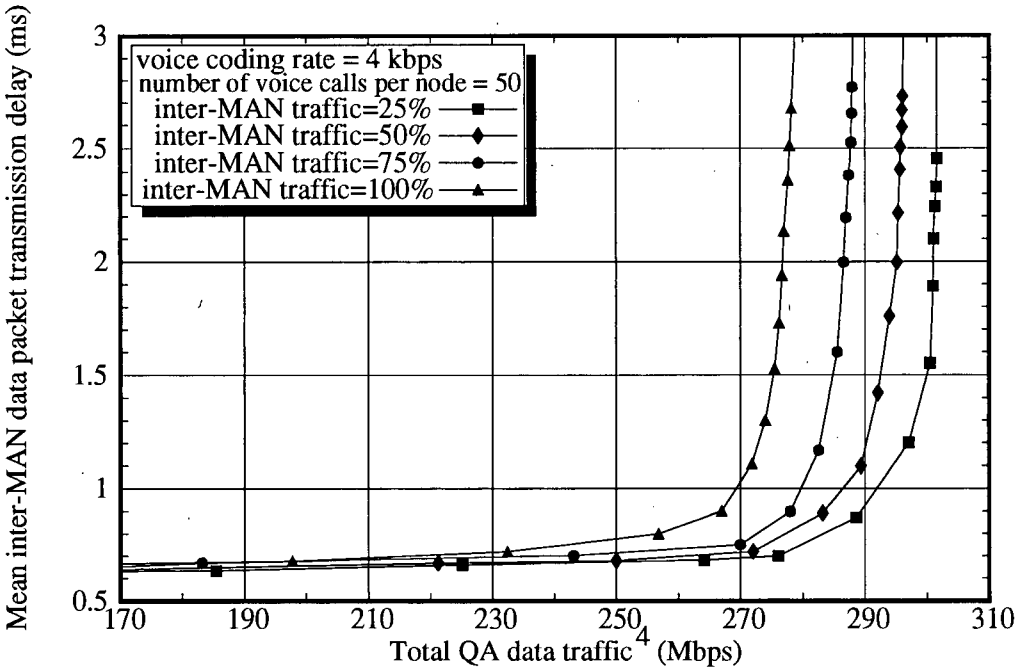


Figure 30. Mean inter-MAN data packet transmission delay vs. total QA data traffic for 4 kbps voice coding rate

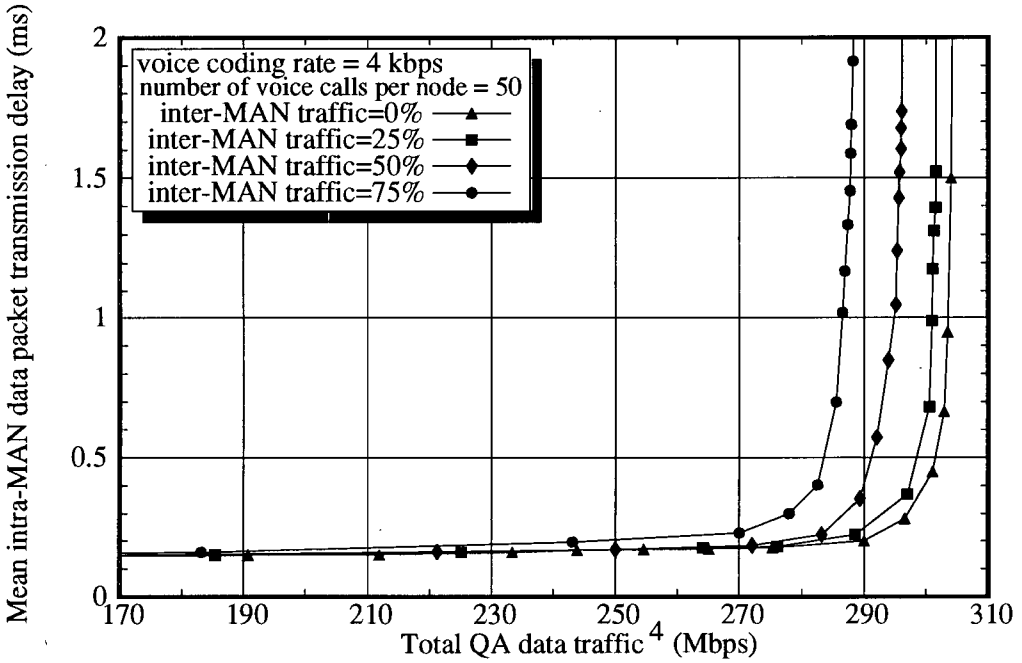


Figure 31. Mean intra-MAN data packet transmission delay vs. total QA data traffic for 4 kbps voice coding rate

The mean inter- and intra-MAN voice packet transmission delay[5] with different percentages of inter-MAN traffic and the two voice coding rates for 50 voice calls per node are summarized in Table 8. Since there is a limited buffer space in each bridge port, the voice throughput achieved may be less than the percent inter-MAN traffic might suggest. Moreover, the voice throughput achieved is calculated excluding any overheads.

As shown in Table 8, the inter- intra-MAN voice packet transmission delays depend not only on their respective traffics, but also on the total traffic achieved in the network (i.e. the summation of the inter- and intra-MAN voice traffics). For example, the fifth entry of the mean intra-MAN voice packet transmission delay in Table 8 is equal to 0.153 ms with a 32.209 Mbps intra-MAN voice traffic; on the other hand, the second entry of the mean intra-MAN voice

[5] The value of the voice packet transmission delay is obtained close to the network's capacity.

packet transmission delay is equal to 0.159 ms with a 7.983 Mbps intra-MAN voice traffic. The latter entry has a longer mean transmission delay than the fifth entry with its smaller intra-MAN voice traffic. This result can be explained by the fact that the total network voice traffic for the second entry is more than 1.5 times that for the fifth entry. Therefore, the mean intra-MAN voice packet transmission delay in the second entry is larger than for the fifth entry.

voice coding rate (kbps)	% inter-MAN traffic	inter-MAN voice thraffic (Mbps)	intra-MAN voice traffic (Mbps)	total voice traffic (Mbps)	total source-destination voice throughput (Mbps)	mean inter-MAN voice packet tx. delay (ms)	mean intra-MAN voice packet tx. delay (ms)
32	100	62.025	0	62.025	29.841	1.224	0.0
	75	47.176	7.983	55.159	31.010	1.024	0.159
	50	31.047	16.167	47.214	31.047	0.735	0.151
	25	16.043	24.134	40.177	32.132	0.680	0.151
	0	0	32.209	32.209	32.209	0.0	0.153
4	100	7.843	0	7.843	3.772	0.990	0.0
	75	5.875	1.070	6.945	3.835	0.899	0.160
	50	4.163	1.986	6.149	4.028	0.733	0.152
	25	2.248	2.802	5.050	4.066	0.687	0.151
	0	0	4.071	4.071	4.071	0.0	0.152

Table 8. Mean inter- and intra-MAN voice packet transmission delay for 50 voice calls per node

Section 4.4
Statistical Results for The Six-MAN Network

The following statistics are obtained from one typical simulation result with 50 active voice calls per node for 32 kbps voice coding; The mean data packet Poisson arrival rate per node is

8500 packets/sec, with 25% inter-MAN traffic. Table 9 summarizes relevant delay and throughput results.

inter-MAN voice throughput (Mbps)	8.006
inter-MAN voice packet tx. delay (ms)	0.677
intra-MAN voice throughput (Mbps)	24.017
intra-MAN voice packet tx.delay (ms)	0.147
total voice throughput (Mbps)	39.711
inter-MAN data throughput (Mbps)	45.043
inter-MAN data packet tx. delay (ms)	0.681
intra-MAN data throughput (Mbps)	135.129
intra-MAN data packet tx. delay (ms)	0.171
total data throughput (Mbps)	225.091
total network throughput (Mbps)	264.802

Table 9. A typical simulation results for the six-MAN network

Figure 32 shows the mean inter-MAN voice and data packet waiting delay over the homogeneous bridges for each access MAN. Bridge 1 connects to the backbone MAN and is located at the head of one bus, bridge 2 is located in the middle of each access MAN, while bridge 3 is another bridge located at the head of another MAN bus.

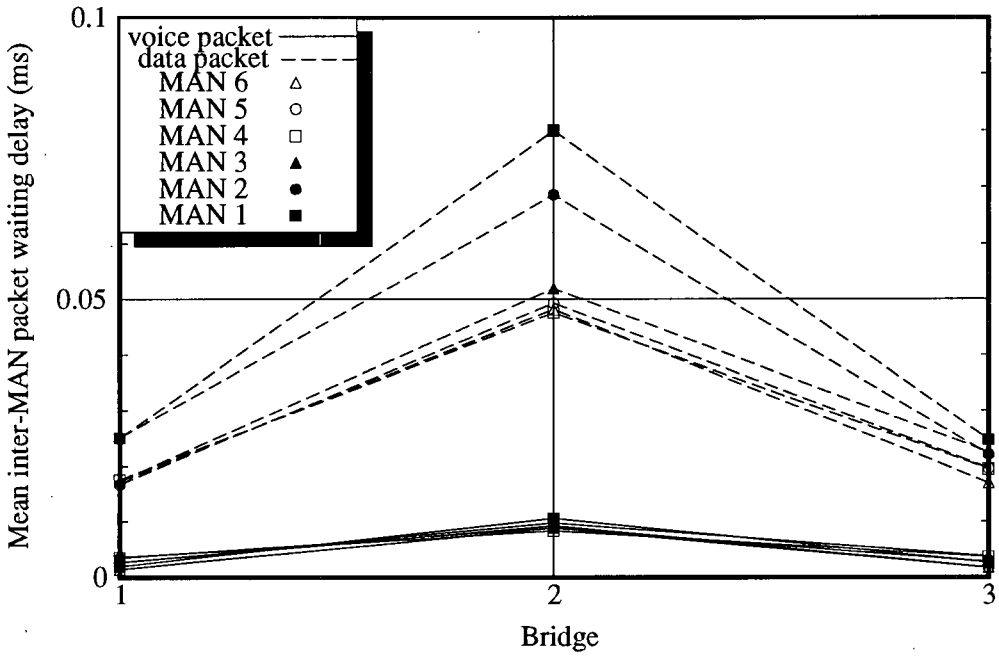


Figure 32. Mean inter-MAN packets waiting delay over homogeneous bridges

As shown in Figure 32, the mean packet waiting delay in bridge 2 (the bridge located in the middle of each access MAN) is higher than at the bridges located at the head of two buses (bridges 1 and 3). Therefore, it is useful to route much of the inter-MAN traffic through the homogeneous bridges located at the head of bus.

Figure 33 shows the mean inter-MAN voice and data packet transmission delay via different bridges. The locations of bridges 1, 2 and 3 are as before. Figure 32 shows the packet waiting delay of different bridges, while Figure 33 shows the packet transmission delay that via the same bridges as in Figure 32.

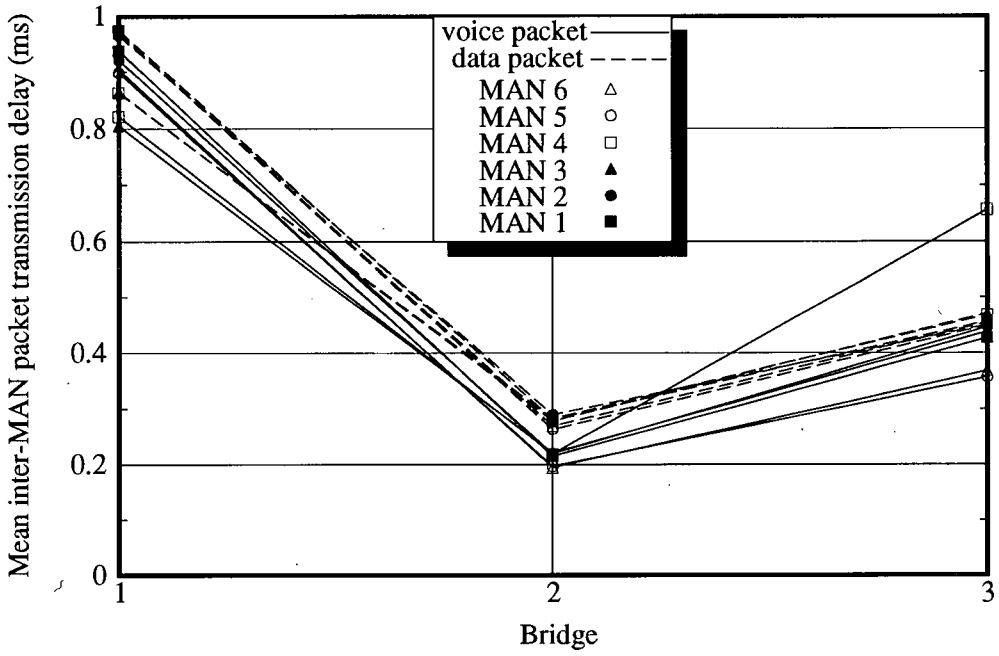


Figure 33. Mean inter-MAN packet transmission delay via different bridges

From figure 33, the mean inter-MAN voice and data packet transmission delay via bridge 2 is less than the packet transmission delay via the other two bridges because of the physical location of bridge 2, and because the packet waiting delay is negligible compared to the propagation delay. From the data obtained for Figure 33, the mean time a packet spends in the backbone MAN is approximately 0.5 ms.

It is shown in the previous chapter that the upstream nodes have a transmission advantage over the nodes located in the middle of two buses in that the waiting delay of a packet at upstream nodes is smaller than for the middle nodes. However, Figure 33 together with Figure 32 shows that the propagation delay is relatively large compared to the waiting delay of a packet; therefore, the transmission delay of the uniformly distributed destination intra-MAN voice and data packets with respect to the nodes' locations will have a shape as shown in Figure 34. All

six access MANs have the same shape of intra-MAN packets transmission delay, we only show the data for one access MAN, access MAN 1.

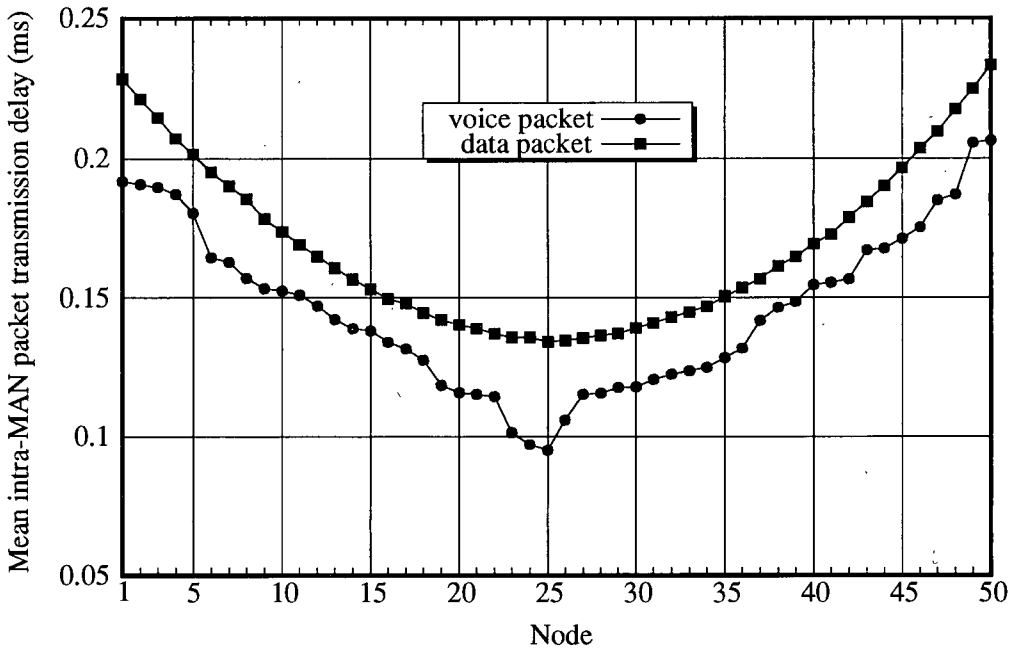


Figure 34. Mean intra-MAN packets transmission delay vs. nodes locations

As expected, the transmission delays for both intra-MAN voice and data packets are larger at the upstream nodes than for the nodes located in the middle of the buses. The reason why the voice packet transmission delay curve is not as smooth as the data packet transmission delay curve in Figure 34 is because the number of voice calls activated in each node is not large enough to uniformly spread the voice call destinations of a node among the remaining 49 MAN nodes.

The inter-MAN voice and data packet transmission delay will depend not only on the nodes' locations, but also on the connection scheme for the homogeneous bridges that connect to each access MAN. Figure 35 shows the inter-MAN data packet transmission delay with respect to the nodes locations. Access MANs 1 to 4 have exactly the same homogeneous bridges connection scheme, and access MANs 5 and 6 have connection schemes which are identical to each other.

Therefore, we plot only one curve for access MANs 1 to 4 and one curve for access MANs 5 and 6.

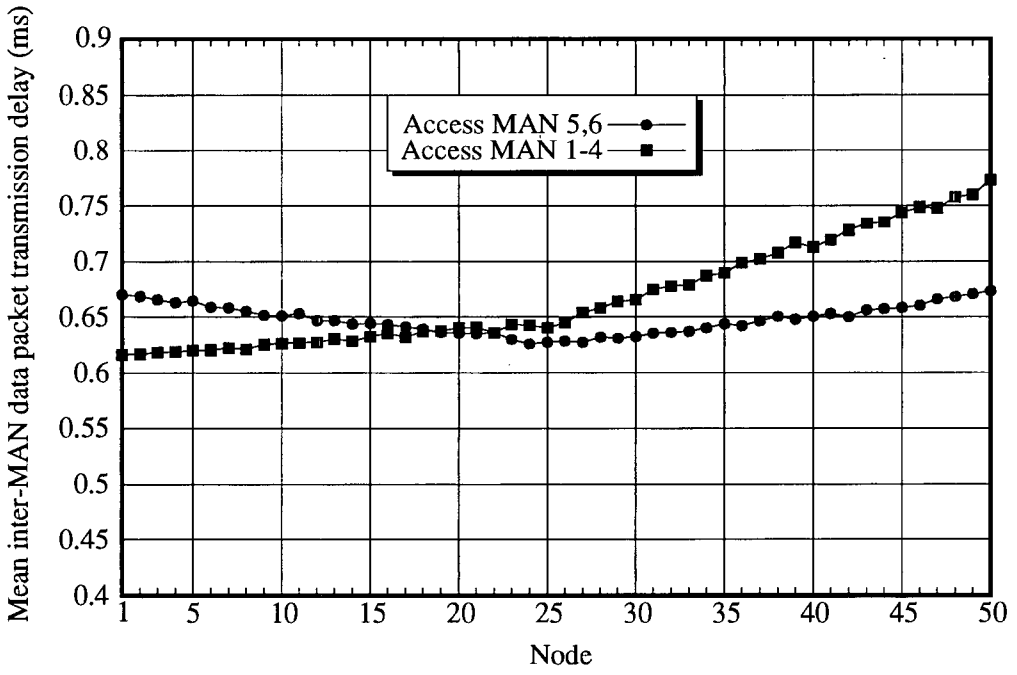


Figure 35. Mean inter-MAN data packet transmission delay vs. nodes locations

In Figure 35, node 1 is closest to the homogeneous bridge connected to the backbone MAN, while node 50 is the farthest node from the backbone MAN. For access MANs 1 to 4, three out of the five other MANs require inter-MAN traffic to be route through the backbone MAN. Since the destination MAN traffic is uniformly distributed, the nodes further away from the backbone MAN will have larger transmission delays than nodes closer to the backbone MAN, as shown in Figure 35. For access MANs 5 and 6, the homogeneous bridges connection scheme divides the uniformly distributed destination MAN traffic relatively even among the four bridges, and the fluctuation in nodal delay is not as large as access MAN 1 to 4. Moreover, the nodal inter-MAN packet transmission delay for access MANs 5 and 6 also preserved the shape as the intra-MAN packet transmission delay.

Figure 36 shows the inter-MAN voice packet transmission delay with respect to the node locations; this figure reflects the same general characteristics as in Figure 35.

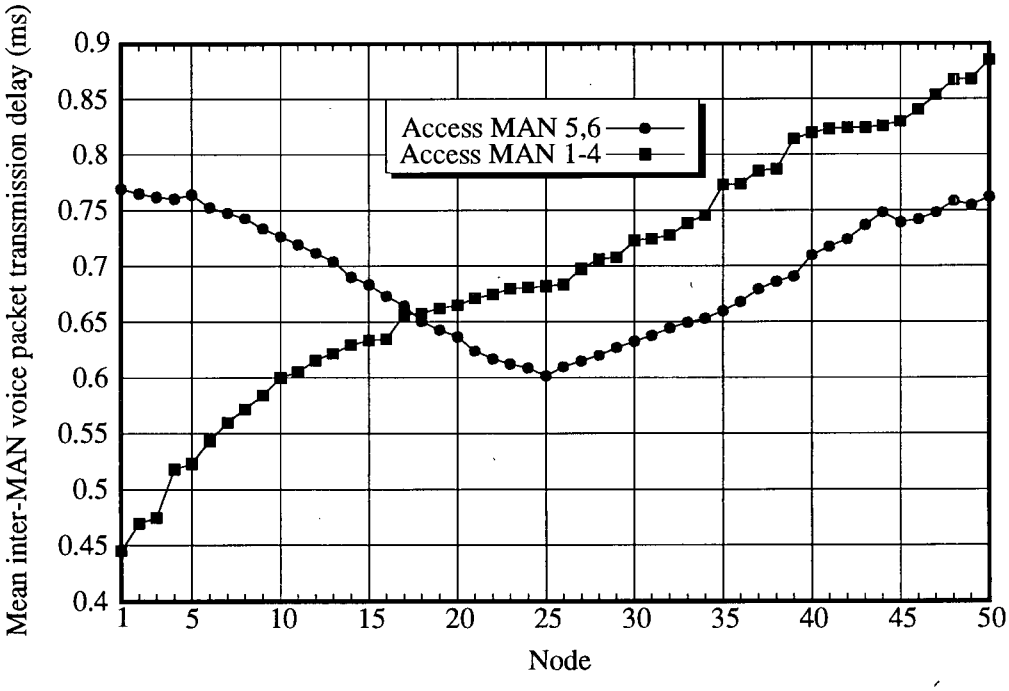


Figure 36. Mean inter-MAN voice packet transmission delay vs. nodes locations

For this typical simulation in this chapter, the percentage of discarded voice packets in the bridge ports is equal to 0.0014%, whereas the percentage of discarded data packets is 0.165%. Recall that voice packets are not retransmitted, whereas data packets are retransmitted.

Section 4.5
General Results of The Inter- and Intra-MAN Traffic within a Backbone MAN Network

In this section, the inter- and intra-MAN voice and data packet transmission delay for different throughput and inter-MAN traffic levels are presented. In order to achieve the desired voice throughput, the assumption of 50 voice calls per node in the previous section is waived.

Instead, additional active voice calls per node are simulated for higher voice throughput. Figure 37 to 39 show the mean inter- and intra-MAN data packet transmission delay versus the total QA data traffic for different voice throughputs and different inter-MAN traffic levels.

Figures 40 and 41 show the mean inter- and intra-MAN voice packet transmission delay versus the total QA voice traffic[6] for different inter-MAN traffic levels.

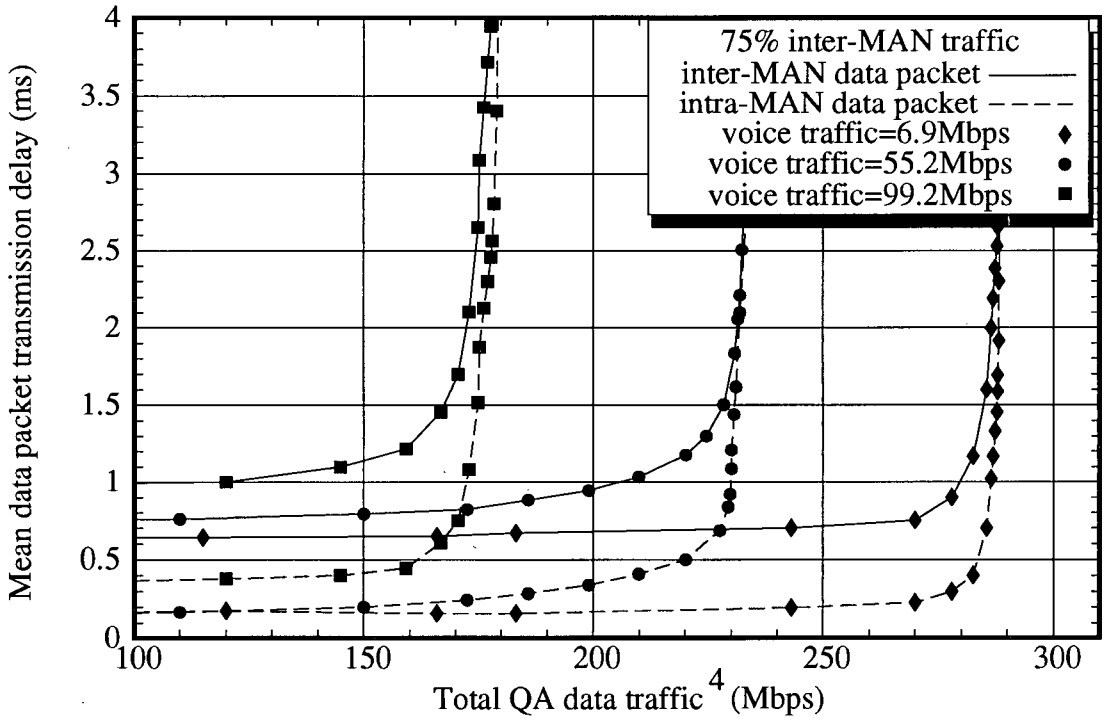


Figure 37. Mean inter- and intra-MAN data packet transmission delay vs. total data traffic for 75% inter-MAN traffic

[6] Total QA voice traffic is the sum of inter- and intra-MAN voice traffic per single access MAN.

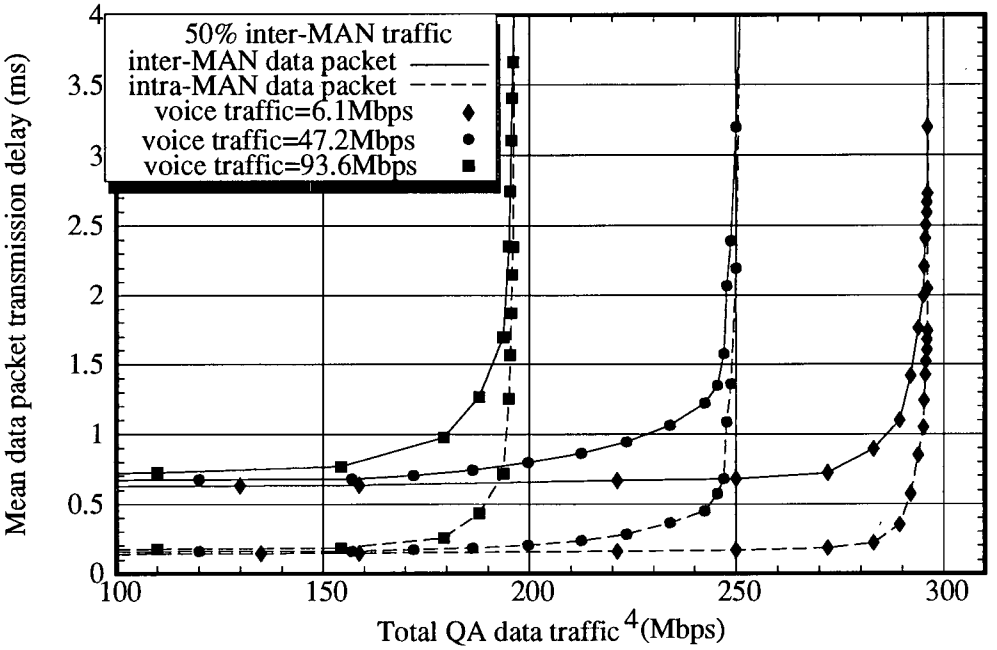


Figure 38. Mean inter- and intra-MAN data packet transmission delay vs. total data traffic for 50% inter-MAN traffic

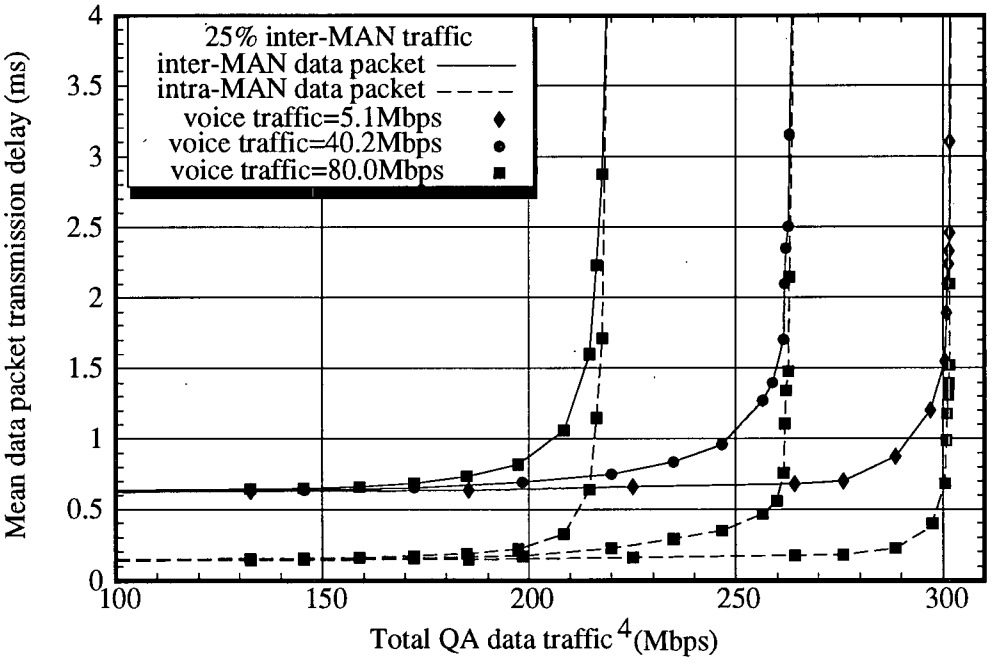


Figure 39. Mean inter- and intra-MAN data packet transmission delay vs. total data traffic for 25% inter-MAN traffic

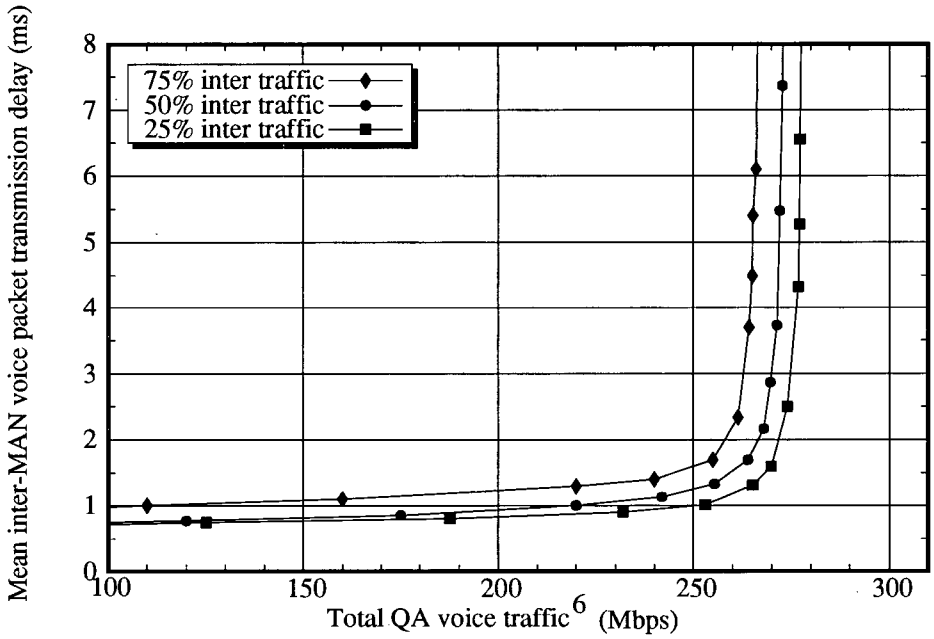


Figure 40. Mean inter-MAN voice packet transmission delay vs. total QA voice traffic for various levels of inter-MAN traffic

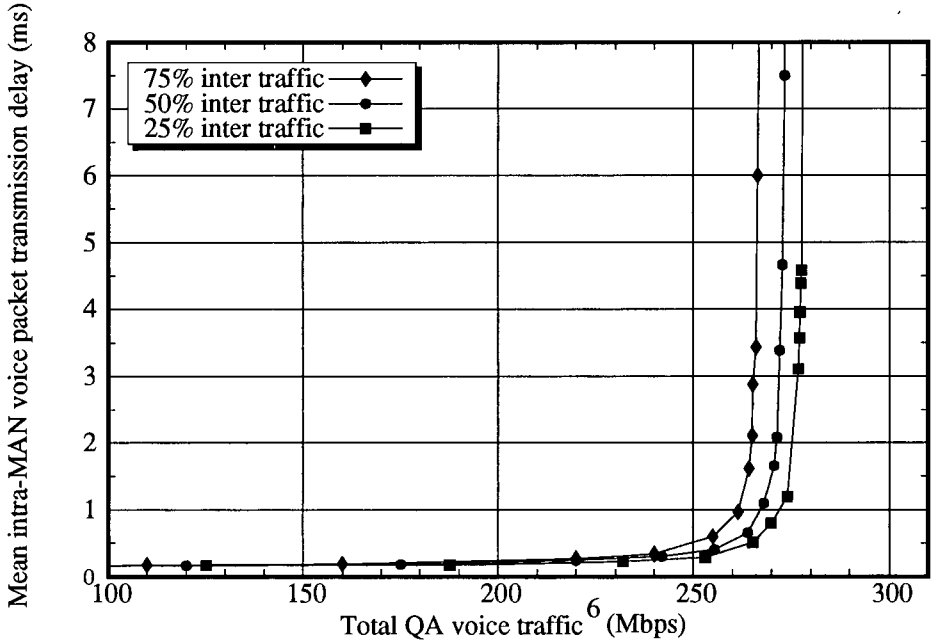


Figure 41. Mean intra-MAN voice packet transmission delay vs. total QA voice traffic for various levels of inter-MAN traffic

Table 10 summarizes the maximum data throughput achieved and its respective data packet transmission delay for various voice throughput levels. Note that the total throughput (voice and data) falls as the voice throughput increases since voice packet overheads throughput also increases as the voice throughput increases. Packet overheads are excluded for the entries of voice throughput in Table 10.

Voice throughput (Mbps)	Maximum data throughput achieved (Mbps)	Total throughput (Mbps)	Mean intra-MAN data packet tx. delay (ms)	Mean inter-MAN data packet tx. delay (ms)
102.87	195.08	297.95	2.27	5.04
127.32	167.77	295.09	2.55	5.60
153.04	139.84	292.88	3.23	6.96
178.64	112.01	290.65	4.14	8.78
202.96	85.47	288.43	5.70	11.89
230.16	55.65	285.81	8.90	18.30
255.54	27.71	283.25	19.81	40.11
270.03	9.25	279.28	36.94	74.38

Table 10. Data packet transmission delay and throughput achieved for different voice throughput levels

With the assumptions and restrictions of our simulation model, the backbone MAN network can accommodate approximately 185 active voice calls per node for 64 kbps voice coding rate with an acceptable 2% discarded voice packets for a 25% inter-MAN traffic level network. This loading represents 9250 active voice calls per MAN and 55500 active voice calls per six-MAN network. For a 50% inter-MAN traffic level network, the system can accommodate approximately 175 active voice calls per node, 8750 per MAN and 52500 per six-MAN network. For a 75% inter-MAN traffic, the system can accommodate approximately 160 active voice calls per node,

8000 per MAN and 48000 per six-MAN network. All delays show in Figures 37 to 41, inclusive, are less than 1.0 ms provided that the QA throughput is limited below some maximum value.

Section 4.6
Call Setup Delay

In this section, we compare the call setup delay of isochronous QA and PA traffic. The setup delay for QA access is expected to be smaller than for PA since QA avoids the complexity and the delay for the Bandwidth Manager (BWM) and VCI to allocate valid PA slots. Moreover, QA access for isochronous traffic eliminates the call clearing process required to release PA bandwidth reserved for voice calls.

4.6.1 Signalling protocol

Below, we explore the possibility of using CCITT Q.931 as the call setup protocol for the MAN-based PCN network. The choice of CCITT Q.931 as the signalling protocol is based on its adoption for the ISDN User-Network Interface (UNI). Because this signalling protocol and call control procedures are designed for wireline networks with centralized switching to enable exchange of control signals between fixed points, it is suitable for implementation in the connection-oriented MAN. In CCITT Q.931, call establishment is initiated when the caller sends a SETUP message across the UNI. Upon receipt of the SETUP message, the network returns a SETUP ACKNOWLEDGE message to the caller. The caller then sends any remaining call information in an INFORMATION message. After receiving this message, the network sends a CALL PROCEEDING message to the caller. Upon receiving ALERTING indication initiated at the called address, the network sends an ALERTING message to the caller. The network then sends a CONNECT message to the caller upon receiving a CONNECT indication from the

called party. Call setup is completed after the caller sends a CONNECT ACKNOWLEDGE message to the network [10].

If PA slots are used to transfer isochronous traffic, the network's Signalling Termination (ST), Bandwidth Manager & VCI Serve (NM) and HOB start to exchange information for allocating an isochronous channel for the caller after the ST receives the SETUP message from the caller. The ST will then send the SETUP ACKNOWLEDGE message back to the caller if and only if an isochronous channel is successfully allocated for that caller. The allocation of an isochronous channel for the called party is similar to that for the caller, except that the request for an isochronous bandwidth is initiated by the ST only after a channel has been allocated for the caller. The ST and NM could be located anywhere on the bus [10]. Figure 42 shows the call setup procedures between the caller and callee.

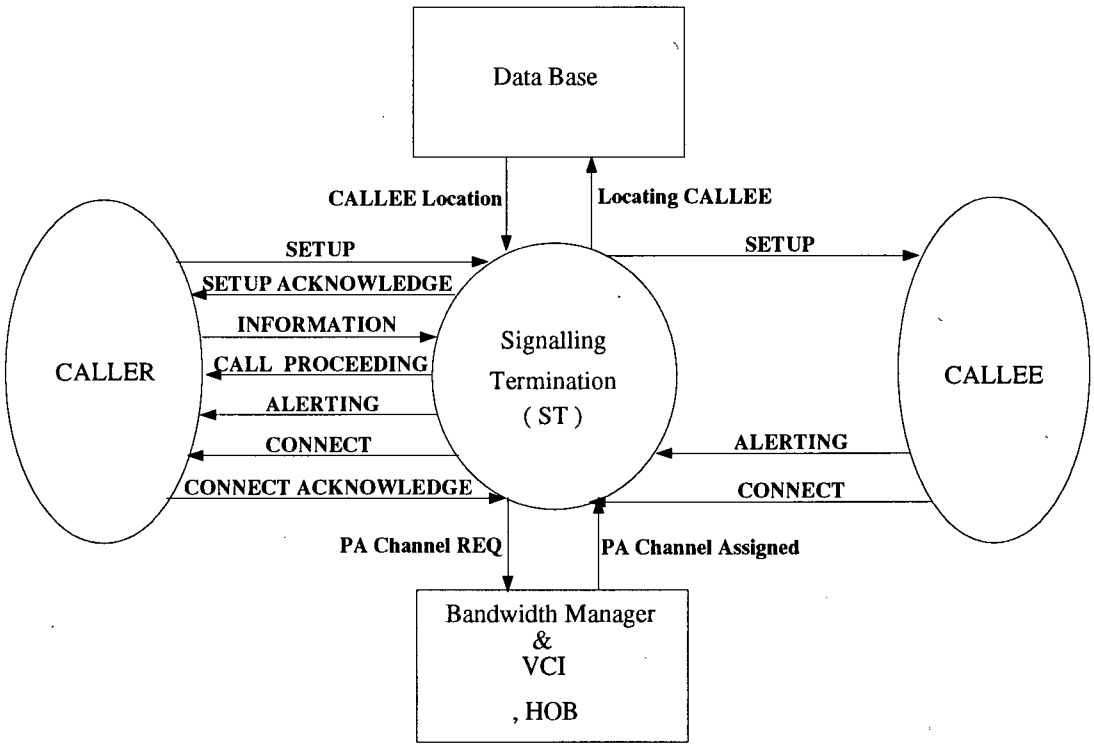


Figure 42. Call setup procedure of Q.931 protocol for PA access. For QA access, signalling involving the Bandwidth Manager and VCI is eliminated.

If QA slots are used to transfer isochronous traffic, the delay and complexity of PA channel allocations for the caller and callee during call setup can be eliminated.

4.6.2 Call setup procedure

From [2], we know that an overlapping call setup procedure provides a lower mean call setup delay than a sequential call setup procedure. The overlapping call setup procedure reduces the setup time by processing isochronous channel setup and the second information phase at the same time (see [2] for more details). Therefore, we expect that reduction in call setup delay for QA access for isochronous traffic over PA access will be greater with sequential call setup than in an overlapping setup procedure.

4.6.3 Performance analysis

As before, the network configuration consists of 50 nodes uniformly distributed in each access MAN, with two successive nodes is separated by 3 slots transmission time. Mean nodal processing delay is 100 μs for each signalling message. The length of the SETUP message is assumed to be 5 slots and that of other signalling messages is 1 slot. Signalling messages are transferred in the highest QA access priority (i.e. priority 2) for the network.

Results in Table 11 and 12 are obtained from the simulation results of the best signalling architecture (which collocates the HOB and NM) in [2] for mean call setup delay[7] versus the background PA throughput. To show the effects of limited QA capacity for signalling traffic, we have assumed the significant bandwidth (i.e. more than 200 Mbps) is unavailable for QA access, but is instead used for PA access. Otherwise, all signalling traffic delay for QA access is constant and, in all cases, less than 0.2 ms.

---

[7] Call setup delay represents the duration from the time the caller initiates the call to the time the SETUP message reaches the callee. If PA access is used, allocation of PA channel for caller is included in the delay.

Background PA throughput (Mbps)	Mean call setup delay (ms)		
	PA access isochronous traffic		QA access isochronous traffic
	Sequential	Overlapping	
204.0	7.72	4.95	4.03
240.8	8.62	5.52	4.48
256.6	8.88	5.68	4.61
273.8	9.52	6.00	4.83

Table 11. Mean inter-MAN call setup delay vs. background PA throughput over 2 transitional MANs

Background PA throughput (Mbps)	Mean call setup delay (ms)		
	PA access isochronous traffic		QA access isochronous traffic
	Sequential	Overlapping	
204.0	3.19	2.88	2.27
240.8	3.53	3.18	2.49
256.6	3.62	3.27	2.56
273.8	3.84	3.45	2.67

Table 12. Mean intra-MAN call setup delay vs. background PA throughput

As expected, the mean QA access call setup delay for isochronous traffic is smaller than the mean PA access setup delay regardless of the call setup procedures used. Since the mean delay for QA access of isochronous packets increases as the traffic throughput increases, it is wise to implement QA access for isochronous traffic under light traffic load and to implement PA access for isochronous traffic under heavy traffic. PA access guarantees bounded delay by denying admission to any packets for which isochronous slots are not available.

Chapter 5. Summary and Conclusions

To handle the increasing communication traffic loads which result from the rapid growth in demand for Personal Communications Services, a distributed switching network architecture based on the IEEE 802.6 MAN with a distributed queue dual bus (DQDB) protocol is proposed. This distributed network has several potential advantages relative to current centralized network architectures for supporting PCS. Performance analysis of the proposed network is needed. Accordingly, network delay versus throughput performance for mixed data and real-time voice traffic is determined using computer simulations.

Section 5.1
Results Summary

The efficiency of the DQDB protocol is analyzed first by comparing its delay-vs-throughput performance with and without slot reuse. Two slot reuse schemes, *PSI and Destination Release*, are employed for the comparison. Simulation results indicate that performance with slot reuse is not much improved relative to non-reuse when implemented under light traffic load. Slot reuse schemes provide some reduction in packet delay under heavy traffic load, but do not provide any significant improvement in throughput. The added implementation complexity associated with slot reuse does not seem worthwhile when DQDB is employed.

Packet transmission delay for inter- and intra-MAN integrated voice and data traffic was evaluated in detail, using Queue-Arbitrated (QA) access only. QA slots with priority 1 access (the middle level) are used for transporting voice traffic in our simulation models. QA slots with the lowest priority access (level zero) are used to transport data traffic and QA slots with the highest priority (level two) would be used to transport control signalling. Simulation results

prove that the waiting delay of a packet is negligible compared to the propagation delay of the packet. Since QA access is used for packets transfer, packet transmission delay for each priority level will depend not only on its own priority level traffic, but also on higher priority level traffic.

The homogeneous bridge schemes employed for MAN interconnection affect inter-MAN packet transmission delay. The four–bridge connection scheme proposed preserves the same packet transmission delay characteristics for inter-MAN traffic as for intra-MAN traffic. As voice throughput increases from 103 to 270 Mbps, the total throughput (voice plus data) varies from 298 to 279 Mbps with a two percent bound on dropped voice packets. Mean inter- and intra-MAN delays vary from 5.0 and 2.3 ms, respectively, at 103 Mbps voice throughput to 74.4 and 36.9 ms at 270 Mbps voice throughput.

Call setup and clearing delays are compared for Queue-Arbitrated Access Isochronous Traffic (QAAIT) and Pre-Arbitrated Access Isochronous Traffic (PAAIT). QAAIT call setup excludes the complex and time consuming process of isochronous channel allocation, and is simpler and faster than for PAAIT. Call clearing involves release of the isochronous channel after call completion, and can be omitted for QAAIT. Although QAAIT has faster call setup and clearing than PAAIT, use of QA access is most appropriate under light traffic loading. Use of PA access under heavy traffic loading is recommended, to provide a bounded packet transmission delay by limiting channel access. QA access results in delays which may become unbounded as traffic levels increases.

Section 5.2
Suggestions for Future Work

There is much future work which would be of interest. Of particular interest are the following issues:

• Performance Analysis for Different Homogeneous Bridges Connection Schemes:

In order to optimize the performance of the Backbone MAN network architecture, it may be useful to investigate further the Backbone MAN network performance by relocating the homogeneous bridges for the interconnected MANs or by altering the physical location of the interconnected MANs within the backbone MAN cluster.

• Performance Analysis for Interwoking Between IEEE 802.6 MAN and Heterogeneous Networks:

Since a DQDB MAN network acts as an information exchange media for its subnetworks, analysis in interworking between a DQDB MAN network and its subnetworks is essential. For example, it would be useful to investigate the possibility of using ATM network as an interconnection network between interconnected DQDB MANs or using DQDB MAN network as an interconnection network for interconnected ATM networks. It would be useful as well to examine the performance of using heterogeneous bridges for connecting DQDB MANs with different subnetworks (e.g. LAN, PBX).

• Performance and Feasibility of Coexisting PA and QA Traffic:

It would be of interest to examine the feasibility and performance advantages of using QA access for all traffic types under light loading, while employing PA access for real-time traffic

under heavy traffic load. One of the difficulties is the delay which occurs in packing QA or PA slots for low-bit rate voice coding; this delay is 96 ms for 48–byte PA slots at 4 kbps coding rate. One approach to overcoming the delay is to pack more than one call per cell, which could then involve different parts of a cell having different destinations. The complexity increases considerably, and requires thoughtful analysis of all of the relevant issues.

Appendix A Source Code

Backbone MAN network Simulation Model Source Code is written in **SIMSCRIPT II.5**

```
********************************************************************************
*                                                                              *
* The PREAMBLE routine gives the description of each modelling element like    *
*  variable type, temporary element, permanent element, process, etc.         *
*                                                                              *
********************************************************************************
preamble

normally, mode is undefined

processes include STATION.VD.BUSA, STATION.VD.BUSB, PACKET.ARR
processes include VOICE.CALL.ARR, VOICE.PACKET.UPDATE, STOP.SIM
processes include BRIDGEA.VD.BUSA, BRIDGEA.VD.BUSB, BRIDGEB.VD.BUSA
processes include BRIDGEB.VD.BUSB, BRIDGEC.VD.BUSA, BRIDGEC.VD.BUSB
processes include BACKBONE.MAN, BACKBONE.HEADA.VD.BUSA, BACKBONE.HEADA.VD.BUSB,
                  BACKBONE.HEADB.VD.BUSA, BACKBONE.HEADB.VD.BUSB,
                  BACKBONE.NODE.VD.BUSA, BACKBONE.NODE.VD.BUSB

permanent entities
        every STATN own a vsbufferq and a dsbufferq
        every BRIDGE own a vbbufferq and a dbbufferq
        every BBNODE own a vbbbufferq and a dbbbufferq

temporary entities
        every vsbuffer has a Vtemp.add, a Vdelay.temp, a Vsdest.man
              and belong to the vsbufferq
        every dsbuffer has a temp.add, a delay.temp, a sdest.man
              and belong to the dsbufferq
        every vbbuffer has a Vadd.bridge, a Vdelay.bridge, a Vdest.MAN,
              a Vini.MAN, a bvdelay, a Vini.add and belong to the vbbufferq
        every dbbuffer has a add.bridge, a delay.bridge, a dest.MAN,
              a ini.man, a bddelay, a ini.add and belong to the dbbufferq
        every vbbbuffer has a Vadd.bb, a Vdelay.bb, a Vdest.MAN.bb,
              a Vini.MAN.bb, a bbvdelay,
              a Vini.add.bb and belong to the vbbbufferq
        every dbbbuffer has a add.bb, a delay.bb, a dest.MAN.bb,
              a ini.MAN.bb, a bbddelay,
              a ini.add.bb and belong to the dbbbufferq


define II, LL, I, spf, man.noAA, Vman as integer variables  '' global index
define man.noAB, man.noBA, man.noBB, man.noCA, man.noCB as integer variables
define man.noSA, man.noSB, man.noPS, man.noVS as integer variables
define N.stn as integer variable '' number of stations
define SD as real variable '' distance between two sucessive stations, slots
define num.slot as integer variable  '' number of slots/bus for simulation
define num.buffer as integer variable  '' number of buffers/station
define Poisson.arr as real variable  '' poisson arrival rate of stations
define slot.add.A, slot.add.B as 2-dimensional, integer array
define B.A, B.B as 2-dimensional, integer array  '' busy bits of slots
define tx.counter.A, Vtx.counter.A as 2-dimensional, integer array
define tx.counter.B, Vtx.counter.B as 2-dimensional, integer array
define req.counter.A, Vreq.counter.A as 2-dimensional, integer array
define req.counter.B, Vreq.counter.B as 2-dimensional, integer array
define tx.counter.BA, Vtx.counter.BA as 2-dimensional, integer array
define tx.counter.BB, Vtx.counter.BB as 2-dimensional, integer array
define req.counter.BA, Vreq.counter.BA as 2-dimensional, integer array
define req.counter.BB, Vreq.counter.BB as 2-dimensional, integer array
define Vnum, vnum.b, dnum.b, discard.vpacket as 2-dimensional, integer array
define DR.A, DR.B as 2-dimensional, integer array '' request bits of data slots
define VR.A, VR.B as 2-dimensional, integer array ''request bits of voice slots
define MAN1.A, MAN1.B, MAN2.A, MAN2.B as 2-dimensional, integer array
define inter.A, inter.B as 2-dimensional, integer array
define voice.A, voice.B as 2-dimensional, integer array
define delay.A, delay.B as 2-dimensional, real array
define vdelay, ddelay as 2-dimensional, real array
define num.re as 4-dimensional, integer array
define delay as 4-dimensional, real array
define delay.temp, Vdelay.temp as real variables
define bvdelay, bddelay as real variables
define temp.add, Vtemp.add as integer variables
define sdest.man, Vsdest.man as integer variables
define Vdelay.bridge, delay.bridge as real variables
define add.bridge, Vadd.bridge as integer variables
define ini.man, Vini.man as integer variables
define dest.man, Vdest.man as integer variables
```



```
define Vini.add, ini.add, Vini.add.bb, ini.add.bb as integer variables
define dsbufferlimit, vsbufferlimit as integer variables
define dbbufferlimit, vbbufferlimit as integer variables
define vsbufsize, dsbufsize as 2-dimensional, integer array
define vbbufsize, dbbufsize as 2-dimensional, integer array
define Vtx.add, Vtx.man, tx.add, tx.man as 2-dimensional, integer array
define Vbtx.add, Vbtx.man, btx.add, btx.man as 2-dimensional, integer array
define voice.add.dest, voice.man.dest as 3-dimensional, integer array
define END.TIME as real variable
define node, voice.call.num, voice.tx, data.tx, pac.arr as integer variables
define slot.dur as real variable
define d, inter, man.count as 1-dimensional, integer array
define bvpac.enter, bvpac.total as integer variables
define bdpac.enter, bdpac.total as integer variables


define BBman as integer variable
define BB.slot.add.A, BB.slot.add.B as 1-dimensional, integer array
define BB.ini.node.A, BB.ini.node.B as 1-dimensional, integer array
define BB.B.A, BB.B.B as 1-dimensional, integer array
define BB.DR.A, BB.DR.B as 1-dimensional, integer array
define BB.VR.A, BB.VR.B as 1-dimensional, integer array
define BB.MAN1.A, BB.MAN1.B as 1-dimensional, integer array
define BB.MAN2.A, BB.MAN2.B as 1-dimensional, integer array
define BB.inter.A, BB.inter.B as 1-dimensional, integer array
define BB.voice.A, BB.voice.B as 1-dimensional, integer array
define BB.delay.A, BB.delay.B as 1-dimensional, real array
define BB.tx.counter.A, BB.Vtx.counter.A as 1-dimensional, integer array
define BB.tx.counter.B, BB.Vtx.counter.B as 1-dimensional, integer array
define BB.req.counter.A, BB.Vreq.counter.A as 1-dimensional, integer array
define BB.req.counter.B, BB.Vreq.counter.B as 1-dimensional, integer array
define Vbbtx.add, bbtx.add as 1-dimensional, integer array
define vnum.bb, dnum.bb as 1-dimensional, integer array
define BB.voice.tx, BB.data.tx as integer variables
define Vadd.bb, Vdest.MAN.bb, Vini.MAN.bb as integer variables
define add.bb, dest.MAN.bb, ini.MAN.bb as integer variables
define bbddelay, bbvdelay, Vdelay.bb, delay.bb as real variables
define vbbbufsize, dbbbufsize as 1-dimensional, integer array
define BB.vdelay, BB.ddelay as 1-dimensional, real array


define bridge.A, bridge.B as 2-dimensional, integer array
define ini.node.A, ini.node.B as 2-dimensional, integer array
define delay.via.bridge as 3-dimensional, real array
define num.via.bridge as 3-dimensional, integer array


define Bv to mean 384 '' num of bits in voice packet, 48 octets
define Rv to mean 32000 '' voice digitized rate, bps
define BP to mean 424 '' num of bits in a slot
define RN to mean 155000000 '' network tx rate/bus, 155Mbps
define .secs to mean DAYS
define .ms to mean HOURS
define .us to mean MINUTES


define percent.inter.traffic, percent.inter.traffic.v as integer variables
define stn.act as integer variable
define Vstn.act as integer variable


end   '' preamble

********************************************************************************
*                                                                              *
*  The function of MAIN routine is to input variables parameters into the      *
*  system, intialized, set up the configuration of the system and schedule     *
*  the beginning time of each process.                                         *
*                                                                              *
********************************************************************************
main

define m, n, k, s as integer variables  '' indexs

let HOURS.V=1000
let MINUTES.V=1000
```

```
let I=1
let II=1
let LL=1
let man.noAA=1
let man.noAB=1
let man.noBA=1
let man.noBB=1
let man.noCA=1
let man.noCB=1
let man.noSA=1
let man.noSB=1
let man.noPS=1
let man.noVS=1
let stn.act=0
let TIME.V=0.0
let voice.tx=0
let data.tx=0
let pac.arr=0


print 1 line thus
Enter the number of nodes (exclude head & end stations):
read N.stn

print 1 line thus
Enter the separation distance 2 successive nodes in slots:
read SD

print 1 line thus
Enter the Poisson arrival rate of node, packets/sec:
read Poisson.arr

print 1 line thus
Enter the number of buffers per node:
read num.buffer

print 1 line thus
Enter the number of voice stations activated per node:
read Vstn.act

print 1 line thus
Enter the time for simulations, in secs:
read END.TIME

print 1 line thus
Enter the percentage of inter-MAN traffic for data traffic, in %:
read data.percent.inter.traffic

print 1 line thus
Enter the percentage of inter-MAN traffic for voice traffic, in %:
read voice.percent.inter.traffic


let slot.dur=BP/RN
let num.slot=trunc.f(END.TIME/slot.dur)
let spf=(N.stn+3)*SD+5
let N.STATN=6*(N.stn+2)
let N.BRIDGE=6*3
let N.BBNODE=6
let dsbufferlimit=num.buffer
let vsbufferlimit=Vstn.act
let dbbufferlimit=num.buffer
let vbbufferlimit=num.buffer

create every STATN
create every BRIDGE
create every BBNODE

reserve slot.add.A(*,*), slot.add.B(*,*), B.A(*,*), B.B(*,*), MAN1.A(*),
        MAN1.B(*), DR.A(*,*), DR.B(*,*), VR.A(*,*), VR.B(*,*), MAN2.A(*,*),
        MAN2.B(*,*), inter.A(*,*), inter.B(*,*), voice.A(*,*), voice.B(*,*),
        delay.A(*,*), delay.B(*,*), bridge.A(*,*), bridge.B(*,*),
        ini.node.A(*,*), ini.node.B(*,*) as 6 by spf
reserve tx.counter.A(*,*), req.counter.A(*,*), Vtx.counter.A(*,*),
        Vreq.counter.A(*,*), tx.counter.B(*,*), req.counter.B(*,*),
        Vtx.counter.B(*,*), Vreq.counter.B(*,*) as 6 by N.stn+2
```



```
reserve Vnum(*,*), discard.vpacket(*,*) as 6 by N.stn+2
reserve tx.counter.BA(*,*), req.counter.BA(*,*), Vtx.counter.BA(*,*),
        Vreq.counter.BA(*,*), tx.counter.BB(*,*), req.counter.BB(*,*),
        Vtx.counter.BB(*,*), Vreq.counter.BB(*,*) as 6 by 3
reserve voice.add.dest(*,*,*), voice.man.dest(*,*,*) as 6 by N.stn+2 by Vstn.act
reserve delay(*,*,*,*), num.re(*,*,*,*) as 6 by N.stn+2 by 2 by 2
reserve d(*) as Vstn.act*0.4

if percent.inter.traffic.v>5
  reserve inter(*) as trunc.f(Vstn.act*percent.inter.traffic.v/100)
always

reserve vnum.b(*,*), dnum.b(*,*), vdelay(*,*), ddelay(*,*) as 6 by 3
reserve vsbufsize(*,*), dsbufsize(*,*), Vtx.add(*,*), Vtx.man(*,*),
        tx.man(*,*), tx.add(*,*) as 6 by N.stn+2
reserve vbbufsize(*,*), dbbufsize(*,*), Vbtx.add(*,*), Vbtx.man(*,*),
        btx.man(*,*), btx.add(*,*) as 6 by 3

reserve delay.via.bridge(*,*,*), num.via.bridge(*,*,*) as 6 by 3 by 2
reserve man.count(*) as 6


reserve BB.slot.add.A(*), BB.slot.add.B(*), BB.B.A(*), BB.B.B(*), BB.DR.A(*),
        BB.DR.B(*), BB.VR.A(*), BB.VR.B(*), BB.MAN1.A(*), BB.MAN1.B(*),
        BB.MAN2.A(*), BB.MAN2.B(*), BB.inter.A(*), BB.inter.B(*),
        BB.voice.A(*), BB.voice.B(*), BB.delay.A(*), BB.delay.B(*),
        BB.ini.node.A(*), BB.ini.node.B(*) as spf*2
reserve BB.tx.counter.A(*), BB.Vtx.counter.A(*), BB.tx.counter.B(*),
        BB.Vtx.counter.B(*), BB.req.counter.A(*), BB.Vreq.counter.A(*),
        BB.req.counter.B(*), BB.Vreq.counter.B(*), Vbbtx.add(*), bbtx.add(*),
        vnum.bb(*), dnum.bb(*), vbbbufsize(*), dbbbufsize(*), BB.vdelay(*),
        BB.ddelay(*) as 6


for m= 1 to 6
do
  for n = 1 to N.stn+2
  do
   let tx.counter.A(m,n)=-1
   let req.counter.A(m,n)=0
   let tx.counter.B(m,n)=-1
   let req.counter.B(m,n)=0
   let Vtx.counter.A(m,n)=-1
   let Vreq.counter.A(m,n)=0
   let Vtx.counter.B(m,n)=-1
   let Vreq.counter.B(m,n)=0
   let Vnum(m,n)=0
   let discard.vpacket(m,n)=0
   let vsbufsize(m,n)=0
   let dsbufsize(m,n)=0
   let Vtx.add(m,n)=0
   let Vtx.man(m,n)=0
   let tx.add(m,n)=0
   let tx.man(m,n)=0
  loop
loop

for m=1 to 6
do
  for n = 1 to 3
  do
   let tx.counter.BA(m,n)=-1
   let req.counter.BA(m,n)=0
   let tx.counter.BB(m,n)=-1
   let req.counter.BB(m,n)=0
   let Vtx.counter.BA(m,n)=-1
   let Vreq.counter.BA(m,n)=0
   let Vtx.counter.BB(m,n)=-1
   let Vreq.counter.BB(m,n)=0
   let Vnum.b(m,n)=0
   let dnum.b(m,n)=0
   let vdelay(m,n)=0.0
   let ddelay(m,n)=0.0
   let vbbufsize(m,n)=0
   let dbbufsize(m,n)=0
```

```
   let Vbtx.add(m,n)=0
   let Vbtx.man(m,n)=0
   let btx.add(m,n)=0
   let btx.man(m,n)=0
  loop
loop

for k = 1 to 6
do
  for m = 1 to N.stn+2
  do
   for n = 1 to Vstn.act
   do
        let voice.add.dest(k,m,n)=0
        let voice.man.dest(k,m,n)=0
   loop
  loop
loop

for k = 1 to 6
do
 for s = 1 to N.stn+2
 do
  for m = 1 to 2
  do
   for n = 1 to 2
   do
        let delay(k,s,m,n)=0.0
        let num.re(k,s,m,n)=0
   loop
  loop
 loop
loop

for m=1 to 6
do
   for n= 1 to 3
   do
     for k= 1 to 2
     do
        let delay.via.bridge(m,n,k)=0
        let num.via.bridge(m,n,k)=0
     loop
   loop
loop

for m=1 to Vstn.act*0.4          ''initialized array for talkspurt VCs
do
   let d(m)=0
loop

if percent.inter.traffic.v>5
for m=1 to trunc.f(Vstn.act*percent.inter.traffic.v/100)''ini.array inter.v.c.
do
        let inter(m)=0
loop
always

call voice.call.predet
''
''
'' ACTIVATE PROCESSES
''
for m=1 to 6
do
   activate a BRIDGEA.VD.BUSA in (m-1)/15*slot.dur  .secs
loop

for m=1 to 6
do
   activate a BRIDGEB.VD.BUSB in (0.5+(m-1)/15)*slot.dur  .secs
loop

for n= 1 to (N.stn+2)/2
do
   for m=1 to 6
```



```
   do
        activate a PACKET.ARR in (n*SD+(m-2)/15)*slot.dur  .secs
        activate a VOICE.CALL.ARR in (n*SD+(m-2)/15)*slot.dur  .secs
        activate a STATION.VD.BUSA in (n*SD+(m-1)/15)*slot.dur  .secs
   loop
   for m=1 to 6
   do
        activate a STATION.VD.BUSB in (n*SD+0.5+(m-1)/15)*slot.dur  .secs
   loop
loop

for m=1 to 6
do
activate a BRIDGEC.VD.BUSA in ((N.stn+3)*SD/2+(m-1)/15)*slot.dur  .secs
loop

for m=1 to 6
do
activate a BRIDGEC.VD.BUSB in ((N.stn+3)*SD/2+0.5+(m-1)/15)*slot.dur  .secs
loop

for n= (N.stn+2)/2+1 to (N.stn+2)
do
   for m=1 to 6
   do
        activate a PACKET.ARR in (n*SD+(m-2)/15)*slot.dur  .secs
        activate a VOICE.CALL.ARR in (n*SD+(m-2)/15)*slot.dur  .secs
        activate a STATION.VD.BUSA in (n*SD+(m-1)/15)*slot.dur  .secs
   loop
   for m=1 to 6
   do
        activate a STATION.VD.BUSB in (n*SD+0.5+(m-1)/15)*slot.dur  .secs
   loop
loop

for m=1 to 6
do
   activate a BRIDGEB.VD.BUSA in ((N.stn+3)*SD+(m-1)/15)*slot.dur  .secs
loop

for m=1 to 6
do
activate a BRIDGEA.VD.BUSB in ((N.stn+3)*SD+0.5+(m-1)/15)*slot.dur  .secs
loop

activate a BACKBONE.MAN in ((N.stn+3)*SD+0.5+6/15)*slot.dur  .secs

activate a STOP.SIM in END.TIME+((N.stn+3)*SD+1)*slot.dur  .secs

start simulation

STOP
end  '' main

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                              +
+  The VOICE.CALL.ARR routine is use to determine the arrival ( exponentially  +
+  distributed ) time of each voice call in each MAN's node.                   +
+                                                                              +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
process VOICE.CALL.ARR

define stn.no, J, MAN as integer variables

let stn.no=II
let MAN=man.noVS
add 1 to man.noVS

for J = 1 to Vstn.act
do
        let node=stn.no
        let Vman=MAN
        let voice.call.num=J
        activate a VOICE.PACKET.UPDATE now

   wait exponential.f(0.0055/Vstn.act,4)  .secs
```

```
loop

end '' VOICE.CALL.ARR


*****************************************************************************
*                                                                           *
*  The VOICE.PACKET.UPDATE routine is use to generate voice packet according   *
*  to the voice coding rate of the system during a call which is in talkspurt. *
*  This routine also alternate the voice calls between talkspurt and silence   *
*  for an exponentially period of time.                                     *
*                                                                           *
*****************************************************************************

process VOICE.PACKET.UPDATE


define stn.no, MAN, num.call as integer variables
define J as integer variables
define p, END.TIME.VPU as real variable

let stn.no=node
let MAN=Vman
let num.call=voice.call.num
let END.TIME.VPU=END.TIME+stn.no*SD*slot.dur


for J=1 to Vstn.act*0.4
   with num.call=d(J)
find the first case
if none
   go to 'SILENT'
always


while TIME.V < END.TIME.VPU
do
   let p=TIME.V+exponential.f(1.5,4)
   while (TIME.V<=p and TIME.V<END.TIME.VPU)
   do
        if vsbufsize(MAN,stn.no) < vsbufferlimit
           add 1 to vsbufsize(MAN,stn.no)
           create a vsbuffer
           let Vtemp.add(vsbuffer)=voice.add.dest(MAN,stn.no,num.call)
           let Vdelay.temp(vsbuffer)=TIME.V
           let Vsdest.man(vsbuffer)=voice.man.dest(MAN,stn.no,num.call)
           if vsbufsize(MAN,stn.no)=1
                let Vtx.add(MAN,stn.no)=Vtemp.add(vsbuffer)
                let Vtx.man(MAN,stn.no)=Vsdest.man(vsbuffer)
           always
           file this vsbuffer in vsbufferq((MAN-1)*50+stn.no)
           add 1 to Vnum(MAN,stn.no)
        always
        if (TIME.V+Bv/Rv)<END.TIME.VPU
           wait Bv/Rv  .secs
        else
           go to 'FINISH'
        always
   loop

'SILENT'   let p=TIME.V+exponential.f(2.25,4)
           if p < END.TIME.VPU
             wait p-TIME.V  .secs
           else
             go to 'FINISH'
           always
loop

'FINISH' end  ''VOICE.PACKET.UPDATE


*****************************************************************************
*                                                                           *
*  The PACKET.ARR routine is responsible to generate a data packet enters to   *
*  each node according to a Poisson arrival rate. This routine also responsible*
*  for determining the destination MAN and node of the packet according to an  *
*  uniform distribution and inserting the generated data packet to the nodal   *
*  queue.                                                                   *
*                                                                           *
*****************************************************************************

process PACKET.ARR


define stn.no, s, MAN as integer variables
define p, END.TIME.PACKET as real variable

let stn.no=II
let MAN=man.noPS
add 1 to man.noPS
let END.TIME.PACKET=END.TIME+II*SD*slot.dur


while TIME.V<END.TIME.PACKET
do
   if dsbufsize(MAN,stn.no) < dsbufferlimit
        add 1 to dsbufsize(MAN,stn.no)
        create a dsbuffer
        let delay.temp(dsbuffer)=TIME.V
   if percent.inter.traffic<5
        go to 'INTRA'
   always
        let s=randi.f(1,100,5)
        if s<=percent.inter.traffic
           let s=randi.f(1,6,7)
           let temp.add(dsbuffer)=randi.f(1,N.stn+2,4)
           while MAN=s
           do
                let s=randi.f(1,6,4)
           loop
           let sdest.man(dsbuffer)=s
        else
'INTRA'    let sdest.man(dsbuffer)=MAN
           let s=randi.f(1,N.stn+2,4)
           while stn.no=s
           do
                let s=randi.f(1,N.stn+2,4)
           loop
           let temp.add(dsbuffer)=s
        always
        add 1 to man.count(sdest.man(dsbuffer))
        if dsbufsize(MAN,stn.no)=1
           let tx.add(MAN,stn.no)=temp.add(dsbuffer)
           let tx.man(MAN,stn.no)=sdest.man(dsbuffer)
        always
        file this dsbuffer in dsbufferq((MAN-1)*50+stn.no)
        add 1 to pac.arr
   always

   let p=TIME.V+exponential.f(1/Poisson.arr,4)
   if p < END.TIME.PACKET
        wait p-TIME.V  .secs
   else
        go to 'FINISH'
   always

loop

'FINISH' end  '' packet.arr


*****************************************************************************
*                                                                           *
*  The following routines are used to update the transmission address of each  *
*  packet in each node. One of the routines is used to predetermine the voice  *
*  calls addresses.                                                         *
*                                                                           *
*****************************************************************************

routine vstx.add.update given MAN, stn.no

define MAN, stn.no as integer variables
```

```
remove first vsbuffer from vsbufferq((MAN-1)*50+stn.no)
let Vtx.add(MAN,stn.no)=Vtemp.add(vsbuffer)
let Vtx.man(MAN,stn.no)=Vsdest.man(vsbuffer)
file vsbuffer first in vsbufferq((MAN-1)*50+stn.no)

end  '' vstx.add.updata
''
''
''
''
routine stx.add.update given MAN, stn.no

define MAN, stn.no as integer variables

remove first dsbuffer from dsbufferq((MAN-1)*50+stn.no)
let tx.add(MAN,stn.no)=temp.add(dsbuffer)
let tx.man(MAN,stn.no)=sdest.man(dsbuffer)
file dsbuffer first in dsbufferq((MAN-1)*50+stn.no)

end  '' stx.add.updata
''
''
''
''
routine vbtx.add.update given MAN, bridge.no

define MAN, bridge.no as integer variables

remove first vbbuffer from vbbufferq((MAN-1)*3+bridge.no)
let Vbtx.add(MAN,bridge.no)=Vadd.bridge(vbbuffer)
let Vbtx.man(MAN,bridge.no)=Vdest.MAN(vbbuffer)
file vbbuffer first in vbbufferq((MAN-1)*3+bridge.no)

end  '' vbtx.add.updata
''
''
''
''
routine btx.add.update given MAN, bridge.no

define MAN, bridge.no as integer variables

remove first dbbuffer from dbbufferq((MAN-1)*3+bridge.no)
let btx.add(MAN,bridge.no)=add.bridge(dbbuffer)
let btx.man(MAN,bridge.no)=dest.MAN(dbbuffer)
file dbbuffer first in dbbufferq((MAN-1)*3+bridge.no)

end  '' btx.add.updata
''
''
''
''
routine vbbtx.add.update given stn.no

define stn.no as integer variable

remove first vbbbuffer from vbbbufferq(stn.no)
if stn.no=1
   if Vdest.MAN.bb(vbbbuffer)=3
        let Vbbtx.add(stn.no)=3
   else
        if Vdest.MAN.bb(vbbbuffer)=4
           let Vbbtx.add(stn.no)=4
        else
           if Vdest.MAN.bb(vbbbuffer)=6
                let Vbbtx.add(stn.no)=5
           always
        always
   always
else
if stn.no=2
   if Vdest.MAN.bb(vbbbuffer)=2
        let Vbbtx.add(stn.no)=6
   else
        if Vdest.MAN.bb(vbbbuffer)=4
           let Vbbtx.add(stn.no)=4
```

```
        always
   always
else
if stn.no=3
   if Vdest.MAN.bb(vbbbuffer)=1
        let Vbbtx.add(stn.no)=1
   else
        if Vdest.MAN.bb(vbbbuffer)=2
           let Vbbtx.add(stn.no)=6
        else
           if Vdest.MAN.bb(vbbbuffer)=6
                let Vbbtx.add(stn.no)=5
           always
        always
   always
else
if stn.no=4
   if Vdest.MAN.bb(vbbbuffer)=1
        let Vbbtx.add(stn.no)=1
   else
        if Vdest.MAN.bb(vbbbuffer)=2
           let Vbbtx.add(stn.no)=6
        else
           if Vdest.MAN.bb(vbbbuffer)=5
                let Vbbtx.add(stn.no)=2
           always
        always
   always
else
if stn.no=5
   if Vdest.MAN.bb(vbbbuffer)=3
        let Vbbtx.add(stn.no)=3
   else
        if Vdest.MAN.bb(vbbbuffer)=1
           let Vbbtx.add(stn.no)=1
        always
   always
else
if stn.no=6
   if Vdest.MAN.bb(vbbbuffer)=3
        let Vbbtx.add(stn.no)=3
   else
        if Vdest.MAN.bb(vbbbuffer)=4
           let Vbbtx.add(stn.no)=4
        else
           if Vdest.MAN.bb(vbbbuffer)=5
                let Vbbtx.add(stn.no)=2
           always
        always
   always
always
always
always
always
always
always

file vbbbuffer first in vbbbufferq(stn.no)

end ''vbbtx.add.update
''
''
''
''
routine bbtx.add.update given stn.no

define stn.no as integer variable

remove first dbbbuffer from dbbbufferq(stn.no)
if stn.no=1
   if dest.MAN.bb(dbbbuffer)=3
        let bbtx.add(stn.no)=3
   else
        if dest.MAN.bb(dbbbuffer)=4
           let bbtx.add(stn.no)=4
        else
```

```
            if dest.MAN.bb(dbbbuffer)=6
                  let bbtx.add(stn.no)=5
            always
         always
   always
else
if stn.no=2
   if dest.MAN.bb(dbbbuffer)=2
         let bbtx.add(stn.no)=6
   else
         if dest.MAN.bb(dbbbuffer)=4
            let bbtx.add(stn.no)=4
         always
   always
else
if stn.no=3
   if dest.MAN.bb(dbbbuffer)=1
         let bbtx.add(stn.no)=1
   else
         if dest.MAN.bb(dbbbuffer)=2
            let bbtx.add(stn.no)=6
         else
            if dest.MAN.bb(dbbbuffer)=6
                  let bbtx.add(stn.no)=5
            always
         always
   always
else
if stn.no=4
   if dest.MAN.bb(dbbbuffer)=1
         let bbtx.add(stn.no)=1
   else
         if dest.MAN.bb(dbbbuffer)=2
            let bbtx.add(stn.no)=6
         else
            if dest.MAN.bb(dbbbuffer)=5
                  let bbtx.add(stn.no)=2
            always
         always
   always
else
if stn.no=5
   if dest.MAN.bb(dbbbuffer)=3
         let bbtx.add(stn.no)=3
   else
         if dest.MAN.bb(dbbbuffer)=1
            let bbtx.add(stn.no)=1
         always
   always
else
if stn.no=6
   if dest.MAN.bb(dbbbuffer)=3
         let bbtx.add(stn.no)=3
   else
         if dest.MAN.bb(dbbbuffer)=4
            let bbtx.add(stn.no)=4
         else
            if dest.MAN.bb(dbbbuffer)=5
                  let bbtx.add(stn.no)=2
            always
         always
   always
always
always
always
always
always
always

file dbbbuffer first in dbbbufferq(stn.no)

end ''bbtx.add.update
''
''
''
''
```

```
''
routine voice.call.predet

define k, n, m, s, c, JJ, trial, a, b, x as integer variables

''
'' uniformly chosen the voice calls for inter MAN
''

if percent.inter.traffic.v>5
while inter(trunc.f(Vstn.act*percent.inter.traffic.v/100))=0
do
        let k=randi.f(1,Vstn.act,4)
        for n= 1 to trunc.f(Vstn.act*percent.inter.traffic.v/100)
          with k=inter(n)
        find the first case
        if none
          for n=1 to trunc.f(Vstn.act*percent.inter.traffic.v/100)
                with inter(n)=0
          find the first case
          if found
                let inter(n)=k
          always
        always
loop
always

''
'' pre-determine the inter v.c. destinations
''

if percent.inter.traffic.v>5
for k=1 to 6 ''by percent.inter.traffic
do
  for n=1 to N.stn+2
  do
   for m= 1 to Vstn.act
   do
     let trial=1
     if voice.add.dest(k,n,m)=0
        for x=1 to trunc.f(Vstn.act*percent.inter.traffic.v/100)
              with m=inter(x)
        find the first case
        if found
           let s=randi.f(1,6,8)
'REPEAT'   let c=randi.f(1,N.stn+2,4)
          while k=s
          do
            let s=randi.f(1,6,8)
          loop
          for JJ= 1 to trunc.f(Vstn.act*percent.inter.traffic.v/100)
            with voice.add.dest(s,c,inter(JJ))=0
          find the first case
          if found
            let voice.add.dest(k,n,m)=c
            let voice.add.dest(s,c,inter(JJ))=n
            let voice.man.dest(k,n,m)=s
            let voice.man.dest(s,c,inter(JJ))=k
            let trial=1
          else
            if trial>20
              for s back from 6 to 1
                with s<>k
                for a=1 to trunc.f(Vstn.act*percent.inter.traffic.v/100)
                  for b back from N.stn+2 to 1
                    with voice.add.dest(s,b,inter(a))=0
                  find the first case
                  if found
                    let voice.add.dest(k,n,m)=b
                    let voice.add.dest(s,b,inter(a))=n
                    let voice.man.dest(k,n,m)=s
                    let voice.man.dest(s,b,inter(a))=k
                    let trial=1
                  always
            else
              add 1 to trial
```

```
                go to 'REPEAT'
              always
            always
          always
        always
      loop
    loop
  loop
always

for m= 1 to Vstn.act              '' pre-determine the intra v.c. destinations
do
  for k=1 to 6 ''by percent.inter.traffic
  do
   for n=1 to N.stn+2
   do
        if voice.add.dest(k,n,m)=0
           let trial=1
'REPEAT1' let s=randi.f(1,N.stn+2,4)
          while n=s
          do
            let s=randi.f(1,N.stn+2,4)
          loop
          for JJ= 1 to Vstn.act
            with voice.add.dest(k,s,JJ)=0
          find the first case
          if found
            let voice.add.dest(k,n,m)=s
            let voice.add.dest(k,s,JJ)=n
            let voice.man.dest(k,n,m)=k
            let voice.man.dest(k,s,JJ)=k
            let trial=1
          else
            if trial>20
                for a= 1 to Vstn.act
                  for b back from N.stn+2 to 1
                    with voice.add.dest(k,b,a)=0 and b<>n
                  find the first case
                  if found
                    let voice.add.dest(k,n,m)=b
                    let voice.add.dest(k,b,a)=n
                    let voice.man.dest(k,n,m)=k
                    let voice.man.dest(k,b,a)=k
                    let trial=1
                  always
            else
              add 1 to trial
              go to 'REPEAT1'
            always
          always
         always
   loop
  loop
loop

for m=1 to trunc.f(Vstn.act*0.4*percent.inter.traffic.v/100)
do
        let d(m)=inter(m)
loop

while d(Vstn.act*0.4)=0           '' uniformly chosen the v.c. in talk period.
do
        let x=randi.f(1,Vstn.act,4)
      for n=1 to trunc.f(Vstn.act*percent.inter.traffic.v/100)
          with x=inter(n)
        find the first case
        if none
           for n= 1 to Vstn.act*0.4
                with x=d(n)
           find the first case
           if none
                for n=1 to Vstn.act*0.4
                        with d(n)=0
                find the first case
                if found
                        let d(n)=x
```

```
                always
            always
          always
loop

end '' voice.call.predet

*******************************************************************************
*                                                                             *
*       The following routines combine to form the backbone MAN.              *
*                                                                             *
*******************************************************************************
process BACKBONE.MAN

define m as integer variable
define star as real variable

let BBman=1
let BB.voice.tx=0
let BB.data.tx=0
let star=(N.stn+3)*SD+0.5+6/15

for m=1 to 6
do
   let BB.tx.counter.A(m)=-1
   let BB.Vtx.counter.A(m)=-1
   let BB.tx.counter.B(m)=-1
   let BB.Vtx.counter.B(m)=-1
loop

activate a BACKBONE.HEADA.VD.BUSA in star*slot.dur  .secs

activate a BACKBONE.HEADB.VD.BUSB in (star+0.5)*slot.dur  .secs

for m=1 to 4
do
   activate a BACKBONE.NODE.VD.BUSA in (star+m*60)*slot.dur  .secs
   activate a BACKBONE.NODE.VD.BUSB in (star+0.5+m*60)*slot.dur  .secs
loop

activate a BACKBONE.HEADB.VD.BUSA in (star+5*60)*slot.dur  .secs

activate a BACKBONE.HEADA.VD.BUSB in (star+0.5+5*60)*slot.dur  .secs

end '' BACKBONE.MAN
''
''
''
''
process BACKBONE.HEADA.VD.BUSA

*******************************************************************************
*                                                                             *
* this process describes the homogeneous bridge at the head station,          *
* refer to bus A, towards bus A in the BACKBONE MAN.                          *
* Slots travel along bus A are started from this bridge.                      *
*                                                                             *
*******************************************************************************

define stn.no, slotA, frame.loop as integer variables
define END.TIME.BBHAA as real variables

let stn.no=1
let slotA=1
let frame.loop=1
let END.TIME.BBHAA=END.TIME+((N.stn+3)*SD+0.5+6/15)*slot.dur

while TIME.V < END.TIME.BBHAA
do
   if slotA>spf*2                 '' initialize frames & slots
        let slotA=1
        add 1 to frame.loop
   always
   let BB.slot.add.A(slotA)=0
   let BB.B.A(slotA)=0
   let BB.DR.A(slotA)=0
```

```
    let BB.VR.A(slotA)=0
    let BB.MAN1.A(slotA)=0
    let BB.MAN2.A(slotA)=0
    let BB.inter.A(slotA)=0
    let BB.voice.A(slotA)=0
    let BB.delay.A(slotA)=0.0
    let BB.ini.node.A(slotA)=0

    if BB.Vtx.counter.A(stn.no)=0     '' TRANSMISSION FOR VOICE PACKET
         remove first vbbbuffer from vbbbufferq(stn.no)
         let BB.slot.add.A(slotA)=Vadd.bb(vbbbuffer)
         let BB.ini.node.A(slotA)=Vini.add.bb(vbbbuffer)
         let BB.MAN1.A(slotA)=Vini.MAN.bb(vbbbuffer)
         let BB.MAN2.A(slotA)=Vdest.MAN.bb(vbbbuffer)
         let BB.inter.A(slotA)=2
         let BB.voice.A(slotA)=2
         let BB.delay.A(slotA)=Vdelay.bb(vbbbuffer)
         let BB.Vtx.counter.A(stn.no)=-1              '' reset tx.counter
         let BB.B.A(slotA)=1                          '' set busy bit
         if frame.loop>5
let BB:vdelay(stn.no)=BB.vdelay(stn.no)+TIME.V-bbvdelay(vbbbuffer)
           add 1 to vnum.bb(stn.no)
           add 1 to BB.voice.tx
         always
           destroy this vbbbuffer
           subtract 1 from vbbbufsize(stn.no)
           if vbbbufferq(stn.no) is not empty
                call vbbtx.add.update giving stn.no
           else
                let Vbbtx.add(stn.no)=0
           always
    else
         if BB.Vtx.counter.A(stn.no)>0
            subtract 1 from BB.Vtx.counter.A(stn.no)
         else
    if BB.Vreq.counter.A(stn.no)>0 and BB.Vtx.counter.A(stn.no)<0
                subtract 1 from BB.Vreq.counter.A(stn.no)
           else
                if BB.tx.counter.A(stn.no)=0     ''TX FOR DATA PACKET
                   remove first dbbbuffer from dbbbufferq(stn.no)
                   let BB.slot.add.A(slotA)=add.bb(dbbbuffer)
                   let BB.ini.node.A(slotA)=ini.add.bb(dbbbuffer)
                   let BB.MAN1.A(slotA)=ini.MAN.bb(dbbbuffer)
                   let BB.MAN2.A(slotA)=dest.MAN.bb(dbbbuffer)
                   let BB.inter.A(slotA)=2
                   let BB.voice.A(slotA)=1
                   let BB.delay.A(slotA)=delay.bb(dbbbuffer)
                   let BB.tx.counter.A(stn.no)=-1  '' reset tx.counter
                   let BB.B.A(slotA)=1                          '' set busy bit
                   if frame.loop>5
let BB.ddelay(stn.no)=BB.ddelay(stn.no)+TIME.V-bbddelay(dbbbuffer)
                        add 1 to dnum.bb(stn.no)
                        add 1 to BB.data.tx
                  always
                  destroy this dbbbuffer
                  subtract 1 from dbbbufsize(stn.no)
                  if dbbbufferq(stn.no) is not empty
                        call bbtx.add.update giving stn.no
                  else
                        let bbtx.add(stn.no)=0
                  always
                else
                   if BB.tx.counter.A(stn.no)>0
                     subtract 1 from BB.tx.counter.A(stn.no)
                   else
    if BB.req.counter.A(stn.no)>0 and BB.tx.counter.A(stn.no)<0
                        subtract 1 from BB.req.counter.A(stn.no)
                     always
                   always
                always
             always
         always
    always

    add 1 to slotA
    wait slot.dur .secs
```

```
loop

end  '' BACKBONE.HEADA.VD.BUSA
''
''
''
''
process BACKBONE.HEADA.VD.BUSB

*************************************************************************
*                                                                       *
* this process describes the action of head station, refer to bus A,    *
* of the backbone MAN towards bus B ( request bus )                     *
*                                                                       *
*************************************************************************

define stn.no, slotB, b as integer variables
define END.TIME.BBHAB as real variable

let stn.no=1
let slotB=1
let END.TIME.BBHAB=END.TIME+((N.stn+3)*SD+0.5+6/15+0.5+5*60)*slot.dur

while TIME.V < END.TIME.BBHAB
do
   if slotB>spf*2
        let slotB=1
   always

   if Vbbtx.add(stn.no)>0 and BB.Vtx.counter.A(stn.no)<0
           let BB.Vtx.counter.A(stn.no)=BB.Vreq.counter.A(stn.no)
           let BB.Vreq.counter.A(stn.no)=0
   always

   if bbtx.add(stn.no)>0 and BB.tx.counter.A(stn.no)<0
           let BB.tx.counter.A(stn.no)=BB.req.counter.A(stn.no)
           let BB.req.counter.A(stn.no)=0
   always

   if BB.VR.B(slotB)=1
       add 1 to BB.Vreq.counter.A(stn.no)
   always

   if BB.DR.B(slotB)=1
       add 1 to BB.req.counter.A(stn.no)
   always

   if BB.B.B(slotB)=1 and BB.MAN2.B(slotB)=1
        let b=1
        if BB.voice.B(slotB)=2
          if vbbufsize(BB.MAN2.B(slotB),b) < vbbufferlimit
                add 1 to vbbufsize(BB.MAN2.B(slotB),b)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=BB.slot.add.B(slotB)
                let Vini.add(vbbuffer)=BB.ini.node.B(slotB)
                let Vdelay.bridge(vbbuffer)=BB.delay.B(slotB)
                let Vini.MAN(vbbuffer)=BB.MAN1.B(slotB)
                let Vdest.MAN(vbbuffer)=BB.MAN2.B(slotB)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(BB.MAN2.B(slotB),b)=1
                   let Vbtx.add(BB.MAN2.B(slotB),b)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(BB.MAN2.B(slotB),b)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((BB.MAN2.B(slotB)-1)*3+b)
                add 1 to bvpac.enter
           always
           add 1 to bvpac.total
        else
         if BB.voice.B(slotB)=1
          if dbbufsize(BB.MAN2.B(slotB),b) < dbbufferlimit
                add 1 to dbbufsize(BB.MAN2.B(slotB),b)
                create a dbbuffer
                let add.bridge(dbbuffer)=BB.slot.add.B(slotB)
                let ini.add(dbbuffer)=BB.ini.node.B(slotB)
                let delay.bridge(dbbuffer)=BB.delay.B(slotB)
```

```
                let ini.MAN(dbbuffer)=BB.MAN1.B(slotB)
                let dest.MAN(dbbuffer)=BB.MAN2.B(slotB)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(BB.MAN2.B(slotB),b)=1
                   let btx.add(BB.MAN2.B(slotB),b)=add.bridge(dbbuffer)
                   let btx.man(BB.MAN2.B(slotB),b)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((BB.MAN2.B(slotB)-1)*3+b)
                add 1 to bdpac.enter
             always
             add 1 to bdpac.total
          always
        always
   always

   add 1 to slotB
   wait slot.dur  .secs
loop

end  '' BACKBONE.HEADA.VD.BUSB
''
''
''
''
process BACKBONE.HEADB.VD.BUSA

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+                                                                       +
* this process describes the action of head station, refer to bus B,   *
* of the backbone MAN towards bus A ( request bus )                    *
+                                                                       +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

define stn.no, slotA, b as integer variables
define END.TIME.BBHBA as real variable

let stn.no=6
let slotA=1
let END.TIME.BBHBA=END.TIME+((N.stn+3)*SD+0.5+6/15+5*60)*slot.dur

while TIME.V < END.TIME.BBHBA
do
   if slotA>spf*2
        let slotA=1
   always

   if Vbbtx.add(stn.no)>0 and BB.Vtx.counter.B(stn.no)<0
          let BB.Vtx.counter.B(stn.no)=BB.Vreq.counter.B(stn.no)
          let BB.Vreq.counter.B(stn.no)=0
   always
   if bbtx.add(stn.no)>0 and BB.tx.counter.B(stn.no)<0
          let BB.tx.counter.B(stn.no)=BB.req.counter.B(stn.no)
          let BB.req.counter.B(stn.no)=0
   always
   if BB.VR.A(slotA)=1
        add 1 to BB.Vreq.counter.A(stn.no)
   always
   if BB.DR.A(slotA)=1
        add 1 to BB.req.counter.A(stn.no)
   always

   if BB.B.A(slotA)=1 and BB.MAN2.A(slotA)=2
        let b=1
        if BB.voice.A(slotA)=2
          if vbbufsize(BB.MAN2.A(slotA),b) < vbbufferlimit
                add 1 to vbbufsize(BB.MAN2.A(slotA),b)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=BB.slot.add.A(slotA)
                let Vini.add(vbbuffer)=BB.ini.node.A(slotA)
                let Vdelay.bridge(vbbuffer)=BB.delay.A(slotA)
                let Vini.MAN(vbbuffer)=BB.MAN1.A(slotA)
                let Vdest.MAN(vbbuffer)=BB.MAN2.A(slotA)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(BB.MAN2.A(slotA),b)=1
                   let Vbtx.add(BB.MAN2.A(slotA),b)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(BB.MAN2.A(slotA),b)=Vdest.MAN(vbbuffer)
```

```
                always
                file this vbbuffer in vbbufferq((BB.MAN2.A(slotA)-1)*3+b)
                add 1 to bvpac.enter
          always
          add 1 to bvpac.total
        else
         if BB.voice.A(slotA)=1
          if dbbufsize(BB.MAN2.A(slotA),b) < dbbufferlimit
                add 1 to dbbufsize(BB.MAN2.A(slotA),b)
                create a dbbuffer
                let add.bridge(dbbuffer)=BB.slot.add.A(slotA)
                let ini.add(dbbuffer)=BB.ini.node.A(slotA)
                let delay.bridge(dbbuffer)=BB.delay.A(slotA)
                let ini.MAN(dbbuffer)=BB.MAN1.A(slotA)
                let dest.MAN(dbbuffer)=BB.MAN2.A(slotA)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(BB.MAN2.A(slotA),b)=1
                   let btx.add(BB.MAN2.A(slotA),b)=add.bridge(dbbuffer)
                   let btx.man(BB.MAN2.A(slotA),b)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((BB.MAN2.A(slotA)-1)*3+b)
                add 1 to bdpac.enter
          always
          add 1 to bdpac.total
         always
        always
   always

   add 1 to slotA
   wait slot.dur  .secs
loop

end  '' BACKBONE.HEADB.VD.BUSA
''
''
''
''
process BACKBONE.HEADB.VD.BUSB

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*                                                                       *
* this process describe the homogeneous bridge at the head station,     *
* refer to bus B, towards bus B in the BACKBONE MAN.                    *
* Slots travel along bus B are started from this bridge.                *
*                                                                       *
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

define stn.no, slotB, frame.loop as integer variables
define END.TIME.BBHAA as real variables

let stn.no=6
let slotB=1
let frame.loop=1
let END.TIME.BBHAA=END.TIME+((N.stn+3)*SD+0.5+6/15+0.5)*slot.dur

while TIME.V < END.TIME.BBHAA
do
   if slotB>spf*2                '' initialize frames & slots
        let slotB=1
        add 1 to frame.loop
   always
   let BB.slot.add.B(slotB)=0
   let BB.B.B(slotB)=0
   let BB.DR.B(slotB)=0
   let BB.VR.B(slotB)=0
   let BB.MAN1.B(slotB)=0
   let BB.MAN2.B(slotB)=0
   let BB.inter.B(slotB)=0
   let BB.voice.B(slotB)=0
   let BB.delay.B(slotB)=0.0
   let BB.ini.node.B(slotB)=0

   if BB.Vtx.counter.B(stn.no)=0     '' TRANSMISSION FOR VOICE PACKET
        remove first vbbbuffer from vbbbufferq(stn.no)
        let BB.slot.add.B(slotB)=Vadd.bb(vbbbuffer)
        let BB.ini.node.B(slotB)=Vini.add.bb(vbbbuffer)
```

```
        let BB.MAN1.B(slotB)=Vini.MAN.bb(vbbbuffer)
        let BB.MAN2.B(slotB)=Vdest.MAN.bb(vbbbuffer)
        let BB.inter.B(slotB)=2
        let BB.voice.B(slotB)=2
        let BB.delay.B(slotB)=Vdelay.bb(vbbbuffer)
        let BB.Vtx.counter.B(stn.no)=-1              '' reset tx.counter
        let BB.B.B(slotB)=1                          '' set busy bit
        if frame.loop>5
let BB.vdelay(stn.no)=BB.vdelay(stn.no)+TIME.V-bbvdelay(vbbbuffer)
          add 1 to vnum.bb(stn.no)
          add 1 to BB.voice.tx
        always
          destroy this vbbbuffer
          subtract 1 from vbbbufsize(stn.no)
          if vbbbufferq(stn.no) is not empty
               call vbbtx.add.update giving stn.no
          else
               let Vbbtx.add(stn.no)=0
          always
   else
        if BB.Vtx.counter.B(stn.no)>0
          subtract 1 from BB.Vtx.counter.B(stn.no)
        else
   if BB.Vreq.counter.B(stn.no)>0 and BB.Vtx.counter.B(stn.no)<0
               subtract 1 from BB.Vreq.counter.B(stn.no)
          else
               if BB.tx.counter.B(stn.no)=0    ''TX FOR DATA PACKET
                  remove first dbbbuffer from dbbbufferq(stn.no)
                  let BB.slot.add.B(slotB)=add.bb(dbbbuffer)
                  let BB.ini.node.B(slotB)=ini.add.bb(dbbbuffer)
                  let BB.MAN1.B(slotB)=ini.MAN.bb(dbbbuffer)
                  let BB.MAN2.B(slotB)=dest.MAN.bb(dbbbuffer)
                  let BB.inter.B(slotB)=2
                  let BB.voice.B(slotB)=1
                  let BB.delay.B(slotB)=delay.bb(dbbbuffer)
                  let BB.tx.counter.B(stn.no)=-1  '' reset tx.counter
                  let BB.B.B(slotB)=1                        '' set busy bit
                  if frame.loop>5
let BB.ddelay(stn.no)=BB.ddelay(stn.no)+TIME.V-bbddelay(dbbbuffer)
                       add 1 to dnum.bb(stn.no)
                       add 1 to BB.data.tx
                  always
                 destroy this dbbbuffer
                 subtract 1 from dbbbufsize(stn.no)
                 if dbbbufferq(stn.no) is not empty
                       call bbtx.add.update giving stn.no
                 else
                       let bbtx.add(stn.no)=0
                 always
               else
                  if BB.tx.counter.B(stn.no)>0
                    subtract 1 from BB.tx.counter.B(stn.no)
                  else
   if BB.req.counter.B(stn.no)>0 and BB.tx.counter.B(stn.no)<0
                       subtract 1 from BB.req.counter.B(stn.no)
                    always
                  always
               always
            always
        always
   always

   add 1 to slotB
   wait slot.dur .secs

loop

end  '' BACKBONE.HEADB.VD.BUSB

''
''
''
''
process BACKBONE.NODE.VD.BUSA
```

```
*******************************************************************************
*                                                                             *
* this process describe the action of the homogeneous bridges located at the  *
* backbone MAN rather than two heads towards bus A.                           *
*                                                                             *
*******************************************************************************


define stn.no, slotA, Tx, frame.loop as integer variables

let stn.no=BBman+1
let slotA=1
let frame.loop=1

while TIME.V < END.TIME+((N.stn+3)*SD+0.5+6/15+BBman*60)*slot.dur
do
   if slotA>spf*2               '' initialize frames & slots
        let slotA=1
        add 1 to frame.loop
   always
  if BB.B.A(slotA)=0
   if BB.Vtx.counter.A(stn.no)=0     '' TRANSMISSION FOR VOICE PACKET
        remove first vbbbuffer from vbbbufferq(stn.no)
        let BB.slot.add.A(slotA)=Vadd.bb(vbbbuffer)
        let BB.ini.node.A(slotA)=Vini.add.bb(vbbbuffer)
        let BB.MAN1.A(slotA)=Vini.MAN.bb(vbbbuffer)
        let BB.MAN2.A(slotA)=Vdest.MAN.bb(vbbbuffer)
        let BB.inter.A(slotA)=2
        let BB.voice.A(slotA)=2
        let BB.delay.A(slotA)=Vdelay.bb(vbbbuffer)
        let BB.Vtx.counter.A(stn.no)=-1              '' reset tx.counter
        let BB.B.A(slotA)=1                          '' set busy bit
        if frame.loop>5
let BB.vdelay(stn.no)=BB.vdelay(stn.no)+TIME.V-bbvdelay(vbbbuffer)
          add 1 to vnum.bb(stn.no)
          add 1 to BB.voice.tx
        always
        destroy this vbbbuffer
        subtract 1 from vbbbufsize(stn.no)
        if vbbbufferq(stn.no) is not empty
               call vbbtx.add.update giving stn.no
        else
               let Vbbtx.add(stn.no)=0
        always
        let Tx=1
   else
        if BB.Vtx.counter.A(stn.no)>0
          subtract 1 from BB.Vtx.counter.A(stn.no)
        else
   if BB.Vreq.counter.A(stn.no)>0 and BB.Vtx.counter.A(stn.no)<0
               subtract 1 from BB.Vreq.counter.A(stn.no)
          else
               if BB.tx.counter.A(stn.no)=0    ''TX FOR DATA PACKET
                  remove first dbbbuffer from dbbbufferq(stn.no)
                  let BB.slot.add.A(slotA)=add.bb(dbbbuffer)
                  let BB.ini.node.A(slotA)=ini.add.bb(dbbbuffer)
                  let BB.MAN1.A(slotA)=ini.MAN.bb(dbbbuffer)
                  let BB.MAN2.A(slotA)=dest.MAN.bb(dbbbuffer)
                  let BB.inter.A(slotA)=2
                  let BB.voice.A(slotA)=1
                  let BB.delay.A(slotA)=delay.bb(dbbbuffer)
                  let BB.tx.counter.A(stn.no)=-1  '' reset tx.counter
                  let BB.B.A(slotA)=1                        '' set busy bit
                  if frame.loop>5
let BB.ddelay(stn.no)=BB.ddelay(stn.no)+TIME.V-bbddelay(dbbbuffer)
                       add 1 to dnum.bb(stn.no)
                       add 1 to BB.data.tx
                  always
                 destroy this dbbbuffer
                 subtract 1 from dbbbufsize(stn.no)
                 if dbbbufferq(stn.no) is not empty
                       call bbtx.add.update giving stn.no
                 else
                       let bbtx.add(stn.no)=0
                 always
                 let Tx=1
```

```
                else
                   if BB.tx.counter.A(stn.no)>0
                     subtract 1 from BB.tx.counter.A(stn.no)
                   else
 if BB.req.counter.A(stn.no)>0 and BB.tx.counter.A(stn.no)<0
                        subtract 1 from BB.req.counter.A(stn.no)
                     always
                   always
                always
             always
       always
 always

else

 if BB.B.A(slotA)=1

   if stn.no=2 and BB.MAN2.A(slotA)=5
      if BB.voice.A(slotA)=2
        if vbbufsize(BB.MAN2.A(slotA),1) < vbbufferlimit
              add 1 to vbbufsize(BB.MAN2.A(slotA),1)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=BB.slot.add.A(slotA)
              let Vini.add(vbbuffer)=BB.ini.node.A(slotA)
              let Vdelay.bridge(vbbuffer)=BB.delay.A(slotA)
              let Vini.MAN(vbbuffer)=BB.MAN1.A(slotA)
              let Vdest.MAN(vbbuffer)=BB.MAN2.A(slotA)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(BB.MAN2.A(slotA),1)=1
                 let Vbtx.add(BB.MAN2.A(slotA),1)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(BB.MAN2.A(slotA),1)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((BB.MAN2.A(slotA)-1)*3+1)
              add 1 to bvpac.enter
         always
         add 1 to bvpac.total
      else
         if BB.voice.A(slotA)=1
           if dbbufsize(BB.MAN2.A(slotA),1) < dbbufferlimit
              add 1 to dbbufsize(BB.MAN2.A(slotA),1)
              create a dbbuffer
              let add.bridge(dbbuffer)=BB.slot.add.A(slotA)
              let ini.add(dbbuffer)=BB.ini.node.A(slotA)
              let delay.bridge(dbbuffer)=BB.delay.A(slotA)
              let ini.MAN(dbbuffer)=BB.MAN1.A(slotA)
              let dest.MAN(dbbuffer)=BB.MAN2.A(slotA)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(BB.MAN2.A(slotA),1)=1
                 let btx.add(BB.MAN2.A(slotA),1)=add.bridge(dbbuffer)
                 let btx.man(BB.MAN2.A(slotA),1)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((BB.MAN2.A(slotA)-1)*3+1)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
   else

   if stn.no=3 and BB.MAN2.A(slotA)=3
      if BB.voice.A(slotA)=2
        if vbbufsize(BB.MAN2.A(slotA),1) < vbbufferlimit
              add 1 to vbbufsize(BB.MAN2.A(slotA),1)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=BB.slot.add.A(slotA)
              let Vini.add(vbbuffer)=BB.ini.node.A(slotA)
              let Vdelay.bridge(vbbuffer)=BB.delay.A(slotA)
              let Vini.MAN(vbbuffer)=BB.MAN1.A(slotA)
              let Vdest.MAN(vbbuffer)=BB.MAN2.A(slotA)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(BB.MAN2.A(slotA),1)=1
                 let Vbtx.add(BB.MAN2.A(slotA),1)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(BB.MAN2.A(slotA),1)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((BB.MAN2.A(slotA)-1)*3+1)
              add 1 to bvpac.enter
```



```
         always
         add 1 to bvpac.total
      else
         if BB.voice.A(slotA)=1
           if dbbufsize(BB.MAN2.A(slotA),1) < dbbufferlimit
              add 1 to dbbufsize(BB.MAN2.A(slotA),1)
              create a dbbuffer
              let add.bridge(dbbuffer)=BB.slot.add.A(slotA)
              let ini.add(dbbuffer)=BB.ini.node.A(slotA)
              let delay.bridge(dbbuffer)=BB.delay.A(slotA)
              let ini.MAN(dbbuffer)=BB.MAN1.A(slotA)
              let dest.MAN(dbbuffer)=BB.MAN2.A(slotA)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(BB.MAN2.A(slotA),1)=1
                 let btx.add(BB.MAN2.A(slotA),1)=add.bridge(dbbuffer)
                 let btx.man(BB.MAN2.A(slotA),1)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((BB.MAN2.A(slotA)-1)*3+1)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
   else

   if stn.no=4 and BB.MAN2.A(slotA)=4
      if BB.voice.A(slotA)=2
        if vbbufsize(BB.MAN2.A(slotA),1) < vbbufferlimit
              add 1 to vbbufsize(BB.MAN2.A(slotA),1)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=BB.slot.add.A(slotA)
              let Vini.add(vbbuffer)=BB.ini.node.A(slotA)
              let Vdelay.bridge(vbbuffer)=BB.delay.A(slotA)
              let Vini.MAN(vbbuffer)=BB.MAN1.A(slotA)
              let Vdest.MAN(vbbuffer)=BB.MAN2.A(slotA)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(BB.MAN2.A(slotA),1)=1
                 let Vbtx.add(BB.MAN2.A(slotA),1)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(BB.MAN2.A(slotA),1)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((BB.MAN2.A(slotA)-1)*3+1)
              add 1 to bvpac.enter
         always
         add 1 to bvpac.total
      else
         if BB.voice.A(slotA)=1
           if dbbufsize(BB.MAN2.A(slotA),1) < dbbufferlimit
              add 1 to dbbufsize(BB.MAN2.A(slotA),1)
              create a dbbuffer
              let add.bridge(dbbuffer)=BB.slot.add.A(slotA)
              let ini.add(dbbuffer)=BB.ini.node.A(slotA)
              let delay.bridge(dbbuffer)=BB.delay.A(slotA)
              let ini.MAN(dbbuffer)=BB.MAN1.A(slotA)
              let dest.MAN(dbbuffer)=BB.MAN2.A(slotA)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(BB.MAN2.A(slotA),1)=1
                 let btx.add(BB.MAN2.A(slotA),1)=add.bridge(dbbuffer)
                 let btx.man(BB.MAN2.A(slotA),1)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((BB.MAN2.A(slotA)-1)*3+1)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
   else

   if stn.no=5 and BB.MAN2.A(slotA)=6
      if BB.voice.A(slotA)=2
        if vbbufsize(BB.MAN2.A(slotA),1) < vbbufferlimit
              add 1 to vbbufsize(BB.MAN2.A(slotA),1)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=BB.slot.add.A(slotA)
              let Vini.add(vbbuffer)=BB.ini.node.A(slotA)
              let Vdelay.bridge(vbbuffer)=BB.delay.A(slotA)
              let Vini.MAN(vbbuffer)=BB.MAN1.A(slotA)
```

```
                let Vdest.MAN(vbbuffer)=BB.MAN2.A(slotA)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(BB.MAN2.A(slotA),1)=1
                   let Vbtx.add(BB.MAN2.A(slotA),1)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(BB.MAN2.A(slotA),1)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((BB.MAN2.A(slotA)-1)*3+1)
                add 1 to bvpac.enter
             always
             add 1 to bvpac.total
          else
             if BB.voice.A(slotA)=1
               if dbbufsize(BB.MAN2.A(slotA),1) < dbbufferlimit
                add 1 to dbbufsize(BB.MAN2.A(slotA),1)
                create a dbbuffer
                let add.bridge(dbbuffer)=BB.slot.add.A(slotA)
                let ini.add(dbbuffer)=BB.ini.node.A(slotA)
                let delay.bridge(dbbuffer)=BB.delay.A(slotA)
                let ini.MAN(dbbuffer)=BB.MAN1.A(slotA)
                let dest.MAN(dbbuffer)=BB.MAN2.A(slotA)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(BB.MAN2.A(slotA),1)=1
                   let btx.add(BB.MAN2.A(slotA),1)=add.bridge(dbbuffer)
                   let btx.man(BB.MAN2.A(slotA),1)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((BB.MAN2.A(slotA)-1)*3+1)
                add 1 to bdpac.enter
              always
              add 1 to bdpac.total
             always
          always
       always
       always
       always
       always
      always
    always

    if BB.VR.A(slotA)=1
         add 1 to BB.Vreq.counter.B(stn.no)
    always

    if BB.DR.A(slotA)=1
         add 1 to BB.req.counter.B(stn.no)
    always

    if Tx=0
if BB.Vtx.counter.B(stn.no)<0 and 0<Vbbtx.add(stn.no)<stn.no and
          BB.VR.A(slotA)=0
            let BB.VR.A(slotA)=1
            let BB.Vtx.counter.B(stn.no)=BB.Vreq.counter.B(stn.no)
            let BB.Vreq.counter.B(stn.no)=0
          always

if BB.tx.counter.B(stn.no)<0 and stn.no>bbtx.add(stn.no)>0 and
          BB.DR.A(slotA)=0
            let BB.DR.A(slotA)=1
            let BB.tx.counter.B(stn.no)=BB.req.counter.B(stn.no)
            let BB.req.counter.B(stn.no)=0
          always
    always

    let Tx=0

    add 1 to slotA
    wait slot.dur .secs

loop

end  '' BACKBONE.NODE.VD.BUSA

''
''
''
''
process BACKBONE.NODE.VD.BUSB
```

```
*******************************************************************************
*                                                                             *
* this process describe the action of the homogeneous bridges located at the  *
* backbone MAN rather than two heads toward bus B.                            *
*                                                                             *
*******************************************************************************

define stn.no, slotB, Tx, frame.loop as integer variables

let stn.no=6-BBman
let slotB=1
let frame.loop=1
add 1 to BBman

while TIME.V < END.TIME+((N.stn+3)*SD+0.5+6/15+0.5+(BBman-1)*60)*slot.dur
do
   if slotB>spf*2                '' initialize frames & slots
        let slotB=1
        add 1 to frame.loop
    always
  if BB.B.B(slotB)=0
   if BB.Vtx.counter.B(stn.no)=0      '' TRANSMISSION FOR VOICE PACKET
        remove first vbbbuffer from vbbbufferq(stn.no)
        let BB.slot.add.B(slotB)=Vadd.bb(vbbbuffer)
        let BB.ini.node.B(slotB)=Vini.add.bb(vbbbuffer)
        let BB.MAN1.B(slotB)=Vini.MAN.bb(vbbbuffer)
        let BB.MAN2.B(slotB)=Vdest.MAN.bb(vbbbuffer)
        let BB.inter.B(slotB)=2
        let BB.voice.B(slotB)=2
        let BB.delay.B(slotB)=Vdelay.bb(vbbbuffer)
        let BB.Vtx.counter.B(stn.no)=-1              '' reset tx.counter
        let BB.B.B(slotB)=1                          '' set busy bit
        if frame.loop>5
let BB.vdelay(stn.no)=BB.vdelay(stn.no)+TIME.V-bbvdelay(vbbbuffer)
          add 1 to vnum.bb(stn.no)
          add 1 to BB.voice.tx
        always
        destroy this vbbbuffer
        subtract 1 from vbbbufsize(stn.no)
        if vbbbufferq(stn.no) is not empty
                call vbbtx.add.update giving stn.no
        else
                let Vbbtx.add(stn.no)=0
        always
        let Tx=1
   else
        if BB.Vtx.counter.B(stn.no)>0
          subtract 1 from BB.Vtx.counter.B(stn.no)
        else
   if BB.Vreq.counter.B(stn.no)>0 and BB.Vtx.counter.B(stn.no)<0
                subtract 1 from BB.Vreq.counter.B(stn.no)
           else
                if BB.tx.counter.B(stn.no)=0    ''TX FOR DATA PACKET
                   remove first dbbbuffer from dbbbufferq(stn.no)
                   let BB.slot.add.B(slotB)=add.bb(dbbbuffer)
                   let BB.ini.node.B(slotB)=ini.add.bb(dbbbuffer)
                   let BB.MAN1.B(slotB)=ini.MAN.bb(dbbbuffer)
                   let BB.MAN2.B(slotB)=dest.MAN.bb(dbbbuffer)
                   let BB.inter.B(slotB)=2
                   let BB.voice.B(slotB)=1
                   let BB.delay.B(slotB)=delay.bb(dbbbuffer)
                   let BB.tx.counter.B(stn.no)=-1  '' reset tx.counter
                   let BB.B.B(slotB)=1                          '' set busy bit
                   if frame.loop>5
let BB.ddelay(stn.no)=BB.ddelay(stn.no)+TIME.V-bbddelay(dbbbuffer)
                        add 1 to dnum.bb(stn.no)
                        add 1 to BB.data.tx
                   always
                   destroy this dbbbuffer
                   subtract 1 from dbbbufsize(stn.no)
                   if dbbbufferq(stn.no) is not empty
                        call bbtx.add.update giving stn.no
                   else
                        let bbtx.add(stn.no)=0
                   always
```

```
                let Tx=1
              else
                if BB.tx.counter.B(stn.no)>0
                  subtract 1 from BB.tx.counter.B(stn.no)
                else
 if BB.req.counter.B(stn.no)>0 and BB.tx.counter.B(stn.no)<0
                    subtract 1 from BB.req.counter.B(stn.no)
                  always
                always
              always
            always
        always
 always

else

 if BB.B.B(slotB)=1

   if stn.no=2 and BB.MAN2.B(slotB)=5
      if BB.voice.B(slotB)=2
        if vbbufsize(BB.MAN2.B(slotB),1) < vbbufferlimit
              add 1 to vbbufsize(BB.MAN2.B(slotB),1)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=BB.slot.add.B(slotB)
              let Vini.add(vbbuffer)=BB.ini.node.B(slotB)
              let Vdelay.bridge(vbbuffer)=BB.delay.B(slotB)
              let Vini.MAN(vbbuffer)=BB.MAN1.B(slotB)
              let Vdest.MAN(vbbuffer)=BB.MAN2.B(slotB)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(BB.MAN2.B(slotB),1)=1
                 let Vbtx.add(BB.MAN2.B(slotB),1)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(BB.MAN2.B(slotB),1)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((BB.MAN2.B(slotB)-1)*3+1)
              add 1 to bvpac.enter
         always
         add 1 to bvpac.total
      else
         if BB.voice.B(slotB)=1
           if dbbufsize(BB.MAN2.B(slotB),1) < dbbufferlimit
              add 1 to dbbufsize(BB.MAN2.B(slotB),1)
              create a dbbuffer
              let add.bridge(dbbuffer)=BB.slot.add.B(slotB)
              let ini.add(dbbuffer)=BB.ini.node.B(slotB)
              let delay.bridge(dbbuffer)=BB.delay.B(slotB)
              let ini.MAN(dbbuffer)=BB.MAN1.B(slotB)
              let dest.MAN(dbbuffer)=BB.MAN2.B(slotB)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(BB.MAN2.B(slotB),1)=1
                 let btx.add(BB.MAN2.B(slotB),1)=add.bridge(dbbuffer)
                 let btx.man(BB.MAN2.B(slotB),1)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((BB.MAN2.B(slotB)-1)*3+1)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
   else

   if stn.no=3 and BB.MAN2.B(slotB)=3
      if BB.voice.B(slotB)=2
        if vbbufsize(BB.MAN2.B(slotB),1) < vbbufferlimit
              add 1 to vbbufsize(BB.MAN2.B(slotB),1)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=BB.slot.add.B(slotB)
              let Vini.add(vbbuffer)=BB.ini.node.B(slotB)
              let Vdelay.bridge(vbbuffer)=BB.delay.B(slotB)
              let Vini.MAN(vbbuffer)=BB.MAN1.B(slotB)
              let Vdest.MAN(vbbuffer)=BB.MAN2.B(slotB)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(BB.MAN2.B(slotB),1)=1
                 let Vbtx.add(BB.MAN2.B(slotB),1)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(BB.MAN2.B(slotB),1)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((BB.MAN2.B(slotB)-1)*3+1)
```

```
              add 1 to bvpac.enter
         always
         add 1 to bvpac.total
      else
         if BB.voice.B(slotB)=1
           if dbbufsize(BB.MAN2.B(slotB),1) < dbbufferlimit
              add 1 to dbbufsize(BB.MAN2.B(slotB),1)
              create a dbbuffer
              let add.bridge(dbbuffer)=BB.slot.add.B(slotB)
              let ini.add(dbbuffer)=BB.ini.node.B(slotB)
              let delay.bridge(dbbuffer)=BB.delay.B(slotB)
              let ini.MAN(dbbuffer)=BB.MAN1.B(slotB)
              let dest.MAN(dbbuffer)=BB.MAN2.B(slotB)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(BB.MAN2.B(slotB),1)=1
                 let btx.add(BB.MAN2.B(slotB),1)=add.bridge(dbbuffer)
                 let btx.man(BB.MAN2.B(slotB),1)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((BB.MAN2.B(slotB)-1)*3+1)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
   else

   if stn.no=4 and BB.MAN2.B(slotB)=4
      if BB.voice.B(slotB)=2
        if vbbufsize(BB.MAN2.B(slotB),1) < vbbufferlimit
              add 1 to vbbufsize(BB.MAN2.B(slotB),1)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=BB.slot.add.B(slotB)
              let Vini.add(vbbuffer)=BB.ini.node.B(slotB)
              let Vdelay.bridge(vbbuffer)=BB.delay.B(slotB)
              let Vini.MAN(vbbuffer)=BB.MAN1.B(slotB)
              let Vdest.MAN(vbbuffer)=BB.MAN2.B(slotB)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(BB.MAN2.B(slotB),1)=1
                 let Vbtx.add(BB.MAN2.B(slotB),1)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(BB.MAN2.B(slotB),1)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((BB.MAN2.B(slotB)-1)*3+1)
              add 1 to bvpac.enter
         always
         add 1 to bvpac.total
      else
         if BB.voice.B(slotB)=1
           if dbbufsize(BB.MAN2.B(slotB),1) < dbbufferlimit
              add 1 to dbbufsize(BB.MAN2.B(slotB),1)
              create a dbbuffer
              let add.bridge(dbbuffer)=BB.slot.add.B(slotB)
              let ini.add(dbbuffer)=BB.ini.node.B(slotB)
              let delay.bridge(dbbuffer)=BB.delay.B(slotB)
              let ini.MAN(dbbuffer)=BB.MAN1.B(slotB)
              let dest.MAN(dbbuffer)=BB.MAN2.B(slotB)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(BB.MAN2.B(slotB),1)=1
                 let btx.add(BB.MAN2.B(slotB),1)=add.bridge(dbbuffer)
                 let btx.man(BB.MAN2.B(slotB),1)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((BB.MAN2.B(slotB)-1)*3+1)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
   else

   if stn.no=5 and BB.MAN2.B(slotB)=6
      if BB.voice.B(slotB)=2
        if vbbufsize(BB.MAN2.B(slotB),1) < vbbufferlimit
              add 1 to vbbufsize(BB.MAN2.B(slotB),1)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=BB.slot.add.B(slotB)
              let Vini.add(vbbuffer)=BB.ini.node.B(slotB)
              let Vdelay.bridge(vbbuffer)=BB.delay.B(slotB)
```

```
*  head of one bus and connect each access MAN with the backbone MAN.          *
*                                                                              *
********************************************************************************

process BRIDGEA.VD.BUSA


define bridge.no, slotA, MAN, frame.loop as integer variables

let MAN=man.noAA
let bridge.no=1
let slotA=1
let frame.loop=1
add 1 to man.noAA

while TIME.V < END.TIME+(0.5+(MAN-1)/15)*slot.dur
do
   if slotA>spf         '' initialize frames & slots
        let slotA=1
        add 1 to frame.loop
   always

   let slot.add.A(MAN,slotA)=0
   let B.A(MAN,slotA)=0
   let DR.A(MAN,slotA)=0
   let VR.A(MAN,slotA)=0
   let MAN1.A(MAN,slotA)=0
   let MAN2.A(MAN,slotA)=0
   let inter.A(MAN,slotA)=0
   let voice.A(MAN,slotA)=0
   let delay.A(MAN,slotA)=0.0
   let bridge.A(MAN,slotA)=0
   let ini.node.A(MAN,slotA)=0

   if Vtx.counter.BA(MAN,bridge.no)=0     '' TRANSMISSION FOR VOICE PACKET
        remove first vbbuffer from vbbufferq((MAN-1)*3+bridge.no)
        let slot.add.A(MAN,slotA)=Vadd.bridge(vbbuffer)
        let ini.node.A(MAN,slotA)=Vini.add(vbbuffer)
        let MAN1.A(MAN,slotA)=Vini.MAN(vbbuffer)
        let MAN2.A(MAN,slotA)=Vdest.MAN(vbbuffer)
        let inter.A(MAN,slotA)=2
        let voice.A(MAN,slotA)=2
        let bridge.A(MAN,slotA)=bridge.no
        let delay.A(MAN,slotA)=Vdelay.bridge(vbbuffer)
        let Vtx.counter.BA(MAN,bridge.no)=-1          '' reset tx.counter
        let B.A(MAN,slotA)=1
                              '' set busy bit
        if frame.loop>5
let vdelay(MAN,bridge.no)=vdelay(MAN,bridge.no)+TIME.V-bvdelay(vbbuffer)
          add 1 to vnum.b(MAN,bridge.no)
          add 1 to voice.tx
        always

          destroy this vbbuffer
          subtract 1 from vbbufsize(MAN,bridge.no)

          if vbbufferq((MAN-1)*3+bridge.no) is not empty
               call vbtx.add.update giving MAN, bridge.no
          else
               let Vbtx.add(MAN,bridge.no)=0
               let Vbtx.man(MAN,bridge.no)=0
          always

   else
        if Vtx.counter.BA(MAN,bridge.no)>0
           subtract 1 from Vtx.counter.BA(MAN,bridge.no)
        else
   if Vreq.counter.BA(MAN,bridge.no)>0 and Vtx.counter.BA(MAN,bridge.no)<0
             subtract 1 from Vreq.counter.BA(MAN,bridge.no)
          else
               if tx.counter.BA(MAN,bridge.no)=0       ''TX FOR DATA PACKET
                  remove first dbbuffer from dbbufferq((MAN-1)*3+bridge.no)
                  let slot.add.A(MAN,slotA)=add.bridge(dbbuffer)
                  let ini.node.A(MAN,slotA)=ini.add(dbbuffer)
                  let MAN1.A(MAN,slotA)=ini.MAN(dbbuffer)
                  let MAN2.A(MAN,slotA)=dest.MAN(dbbuffer)
```

```
               let Vini.MAN(vbbuffer)=BB.MAN1.B(slotB)
               let Vdest.MAN(vbbuffer)=BB.MAN2.B(slotB)
               let bvdelay(vbbuffer)=TIME.V
               if vbbufsize(BB.MAN2.B(slotB),1)=1
                  let Vbtx.add(BB.MAN2.B(slotB),1)=Vadd.bridge(vbbuffer)
                  let Vbtx.man(BB.MAN2.B(slotB),1)=Vdest.MAN(vbbuffer)
               always
               file this vbbuffer in vbbufferq((BB.MAN2.B(slotB)-1)*3+1)
               add 1 to bvpac.enter
          always
          add 1 to bvpac.total
        else
          if BB.voice.B(slotB)=1
            if dbbufsize(BB.MAN2.B(slotB),1) < dbbufferlimit
               add 1 to dbbufsize(BB.MAN2.B(slotB),1)
               create a dbbuffer
               let add.bridge(dbbuffer)=BB.slot.add.B(slotB)
               let ini.add(dbbuffer)=BB.ini.node.B(slotB)
               let delay.bridge(dbbuffer)=BB.delay.B(slotB)
               let ini.MAN(dbbuffer)=BB.MAN1.B(slotB)
               let dest.MAN(dbbuffer)=BB.MAN2.B(slotB)
               let bddelay(dbbuffer)=TIME.V
               if dbbufsize(BB.MAN2.B(slotB),1)=1
                  let btx.add(BB.MAN2.B(slotB),1)=add.bridge(dbbuffer)
                  let btx.man(BB.MAN2.B(slotB),1)=dest.MAN(dbbuffer)
               always
               file this dbbuffer in dbbufferq((BB.MAN2.B(slotB)-1)*3+1)
               add 1 to bdpac.enter
            always
            add 1 to bdpac.total
          always
        always
     always
     always
     always
     always
    always
   always

   if BB.VR.B(slotB)=1
        add 1 to BB.Vreq.counter.A(stn.no)
   always

   if BB.DR.B(slotB)=1
        add 1 to BB.req.counter.A(stn.no)
   always

   if Tx=0
if BB.Vtx.counter.A(stn.no)<0 and Vbbtx.add(stn.no)>stn.no and
         BB.VR.B(slotB)=0
           let BB.VR.B(slotB)=1
           let BB.Vtx.counter.A(stn.no)=BB.Vreq.counter.A(stn.no)
           let BB.Vreq.counter.A(stn.no)=0
        always

if BB.tx.counter.A(stn.no)<0 and stn.no<bbtx.add(stn.no) and
         BB.DR.B(slotB)=0
           let BB.DR.B(slotB)=1
           let BB.tx.counter.A(stn.no)=BB.req.counter.A(stn.no)
           let BB.req.counter.A(stn.no)=0
        always
   always

   let Tx=0

   add 1 to slotB
   wait slot.dur .secs

loop

end  '' BACKBONE.NODE.VD.BUSB


********************************************************************************
*                                                                              *
*  The BRIDGEA routines represent the homogeneous bridges that located at the  *
```

```
                let inter.A(MAN,slotA)=2
                let voice.A(MAN,slotA)=1
                let bridge.A(MAN,slotA)=bridge.no
                let delay.A(MAN,slotA)=delay.bridge(dbbuffer)
                let tx.counter.BA(MAN,bridge.no)=-1  '' reset tx.counter
                let B.A(MAN,slotA)=1                 '' set busy bit
                if frame.loop>5
let ddelay(MAN,bridge.no)=ddelay(MAN,bridge.no)+TIME.V-bddelay(dbbuffer)
                     add 1 to dnum.b(MAN,bridge.no)
                     add 1 to data.tx
                 always
               destroy this dbbuffer
               subtract 1 from dbbufsize(MAN,bridge.no)
               if dbbufferq((MAN-1)*3+bridge.no) is not empty
                     call btx.add.update giving MAN, bridge.no
               else
                     let btx.add(MAN,bridge.no)=0
                     let btx.man(MAN,bridge.no)=0
               always
             else
                if tx.counter.BA(MAN,bridge.no)>0
                  subtract 1 from tx.counter.BA(MAN,bridge.no)
                else
   if req.counter.BA(MAN,bridge.no)>0 and tx.counter.BA(MAN,bridge.no)<0
                     subtract 1 from req.counter.BA(MAN,bridge.no)
                  always
                always
              always
           always
      always
   always

   add 1 to slotA
   wait slot.dur .secs

loop

end  '' BRIDGEA.VD.BUSA
''
''
''
process BRIDGEA.VD.BUSB

*****************************************************************************
*                                                                           *
* this process describe the homogeneous bridge as the head station,         *
* refer to bus A, towards bus B.                                            *
*                                                                           *
*****************************************************************************

define bridge.no, slotB, MAN, b as integer variables
define END.TIME.BRIDGEAB as real variable

let MAN=man.noAB
let bridge.no=1
let slotB=1
add 1 to man.noAB
let END.TIME.BRIDGEAB=END.TIME+((N.stn+3)*SD+0.5+(MAN-1)/15)*slot.dur

if MAN=1
   go to 'first'
else
   if MAN=2
        go to 'second'
   else
        if MAN=3
           go to 'third'
        else
           if MAN=4
                go to 'forth'
           else
                if MAN=5
                   go to 'fifth'
                else
                   go to 'sixth'
```

```
'req'   if slotB>spf
        let slotB=1
   always

   if Vbtx.add(MAN,bridge.no)>0 and Vtx.counter.BA(MAN,bridge.no)<0
           let Vtx.counter.BA(MAN,bridge.no)=Vreq.counter.BA(MAN,bridge.no)
           let Vreq.counter.BA(MAN,bridge.no)=0
   always

   if btx.add(MAN,bridge.no)>0 and tx.counter.BA(MAN,bridge.no)<0
           let tx.counter.BA(MAN,bridge.no)=req.counter.BA(MAN,bridge.no)
           let req.counter.BA(MAN,bridge.no)=0
   always

   if VR.B(MAN,slotB)=1
        add 1 to Vreq.counter.BA(MAN,bridge.no)
   always

   if DR.B(MAN,slotB)=1
        add 1 to req.counter.BA(MAN,bridge.no)
   always

if MAN=1
   go to 'reqdone1'
else
   if MAN=2
        go to 'reqdone2'
   else
        if MAN=3
           go to 'reqdone3'
        else
           if MAN=4
                go to 'reqdone4'
           else
                if MAN=5
                   go to 'reqdone5'
                else
                   go to 'reqdone6'


'first' while TIME.V < END.TIME.BRIDGEAB
do
   go to 'req'
'reqdone1'   if B.B(MAN,slotB)=1 and (MAN2.B(MAN,slotB)=3 or
                MAN2.B(MAN,slotB)=4 or MAN2.B(MAN,slotB)=6)
        let b=1
        if voice.B(MAN,slotB)=2
          if vbbbufsize(b) < vbbufferlimit
                add 1 to vbbbufsize(b)
                create a vbbbuffer
                let Vadd.bb(vbbbuffer)=slot.add.B(MAN,slotB)
                let Vini.add.bb(vbbbuffer)=ini.node.B(MAN,slotB)
                let Vdelay.bb(vbbbuffer)=delay.B(MAN,slotB)
                let Vini.MAN.bb(vbbbuffer)=MAN1.B(MAN,slotB)
                let Vdest.MAN.bb(vbbbuffer)=MAN2.B(MAN,slotB)
                let bbvdelay(vbbbuffer)=TIME.V
                file this vbbbuffer in vbbbufferq(b)
                if vbbbufsize(b)=1
                   call vbbtx.add.update giving b
                always
                add 1 to bvpac.enter
          always
          add 1 to bvpac.total
        else
         if voice.B(MAN,slotB)=1
          if dbbbufsize(b) < dbbufferlimit
                add 1 to dbbbufsize(b)
                create a dbbbuffer
                let add.bb(dbbbuffer)=slot.add.B(MAN,slotB)
                let ini.add.bb(dbbbuffer)=ini.node.B(MAN,slotB)
                let delay.bb(dbbbuffer)=delay.B(MAN,slotB)
                let ini.MAN.bb(dbbbuffer)=MAN1.B(MAN,slotB)
                let dest.MAN.bb(dbbbuffer)=MAN2.B(MAN,slotB)
                let bbddelay(dbbbuffer)=TIME.V
                file this dbbbuffer in dbbbufferq(b)
                if dbbbufsize(b)=1
```

```
                call bbtx.add.update giving b
              always
              add 1 to bdpac.enter
          always
          add 1 to bdpac.total
        always
      always
   always

   add 1 to slotB
   wait slot.dur  .secs

loop
go to 'finish'


'second' while TIME.V < END.TIME.BRIDGEAB
do
   go to 'req'
'reqdone2'   if B.B(MAN,slotB)=1 and (MAN2.B(MAN,slotB)=3 or
             MAN2.B(MAN,slotB)=4 or MAN2.B(MAN,slotB)=5)
       let b=6
       if voice.B(MAN,slotB)=2
         if vbbbufsize(b) < vbbufferlimit
              add 1 to vbbbufsize(b)
              create a vbbbuffer
              let Vadd.bb(vbbbuffer)=slot.add.B(MAN,slotB)
              let Vini.add.bb(vbbbuffer)=ini.node.B(MAN,slotB)
              let Vdelay.bb(vbbbuffer)=delay.B(MAN,slotB)
              let Vini.MAN.bb(vbbbuffer)=MAN1.B(MAN,slotB)
              let Vdest.MAN.bb(vbbbuffer)=MAN2.B(MAN,slotB)
              let bbvdelay(vbbbuffer)=TIME.V
              file this vbbbuffer in vbbbufferq(b)
              if vbbbufsize(b)=1
                 call vbbtx.add.update giving b
              always
              add 1 to bvpac.enter
          always
          add 1 to bvpac.total
       else
        if voice.B(MAN,slotB)=1
         if dbbbufsize(b) < dbbufferlimit
              add 1 to dbbbufsize(b)
              create a dbbbuffer
              let add.bb(dbbbuffer)=slot.add.B(MAN,slotB)
              let ini.add.bb(dbbbuffer)=ini.node.B(MAN,slotB)
              let delay.bb(dbbbuffer)=delay.B(MAN,slotB)
              let ini.MAN.bb(dbbbuffer)=MAN1.B(MAN,slotB)
              let dest.MAN.bb(dbbbuffer)=MAN2.B(MAN,slotB)
              let bbddelay(dbbbuffer)=TIME.V
              file this dbbbuffer in dbbbufferq(b)
              if dbbbufsize(b)=1
                 call bbtx.add.update giving b
              always
              add 1 to bdpac.enter
          always
          add 1 to bdpac.total
        always
       always
   always

   add 1 to slotB
   wait slot.dur  .secs

loop
go to 'finish'


'third' while TIME.V < END.TIME.BRIDGEAB
do
   go to 'req'
'reqdone3'   if B.B(MAN,slotB)=1 and (MAN2.B(MAN,slotB)=1 or
             MAN2.B(MAN,slotB)=2 or MAN2.B(MAN,slotB)=6)
       let b=3
       if voice.B(MAN,slotB)=2
         if vbbbufsize(b) < vbbufferlimit
              add 1 to vbbbufsize(b)
              create a vbbbuffer
              let Vadd.bb(vbbbuffer)=slot.add.B(MAN,slotB)
              let Vini.add.bb(vbbbuffer)=ini.node.B(MAN,slotB)
              let Vdelay.bb(vbbbuffer)=delay.B(MAN,slotB)
              let Vini.MAN.bb(vbbbuffer)=MAN1.B(MAN,slotB)
              let Vdest.MAN.bb(vbbbuffer)=MAN2.B(MAN,slotB)
              let bbvdelay(vbbbuffer)=TIME.V
              file this vbbbuffer in vbbbufferq(b)
              if vbbbufsize(b)=1
                 call vbbtx.add.update giving b
              always
              add 1 to bvpac.enter
          always
          add 1 to bvpac.total
        else
         if voice.B(MAN,slotB)=1
          if dbbbufsize(b) < dbbufferlimit
              add 1 to dbbbufsize(b)
              create a dbbbuffer
              let add.bb(dbbbuffer)=slot.add.B(MAN,slotB)
              let ini.add.bb(dbbbuffer)=ini.node.B(MAN,slotB)
              let delay.bb(dbbbuffer)=delay.B(MAN,slotB)
              let ini.MAN.bb(dbbbuffer)=MAN1.B(MAN,slotB)
              let dest.MAN.bb(dbbbuffer)=MAN2.B(MAN,slotB)
              let bbddelay(dbbbuffer)=TIME.V
              file this dbbbuffer in dbbbufferq(b)
              if dbbbufsize(b)=1
                 call bbtx.add.update giving b
              always
              add 1 to bdpac.enter
          always
          add 1 to bdpac.total
         always
        always
   always

   add 1 to slotB
   wait slot.dur  .secs

loop
go to 'finish'


'forth' while TIME.V < END.TIME.BRIDGEAB
do
   go to 'req'
'reqdone4'   if B.B(MAN,slotB)=1 and (MAN2.B(MAN,slotB)=1 or
             MAN2.B(MAN,slotB)=2 or MAN2.B(MAN,slotB)=5)
       let b=4
       if voice.B(MAN,slotB)=2
         if vbbbufsize(b) < vbbufferlimit
              add 1 to vbbbufsize(b)
              create a vbbbuffer
              let Vadd.bb(vbbbuffer)=slot.add.B(MAN,slotB)
              let Vini.add.bb(vbbbuffer)=ini.node.B(MAN,slotB)
              let Vdelay.bb(vbbbuffer)=delay.B(MAN,slotB)
              let Vini.MAN.bb(vbbbuffer)=MAN1.B(MAN,slotB)
              let Vdest.MAN.bb(vbbbuffer)=MAN2.B(MAN,slotB)
              let bbvdelay(vbbbuffer)=TIME.V
              file this vbbbuffer in vbbbufferq(b)
              if vbbbufsize(b)=1
                 call vbbtx.add.update giving b
              always
              add 1 to bvpac.enter
          always
          add 1 to bvpac.total
        else
         if voice.B(MAN,slotB)=1
          if dbbbufsize(b) < dbbufferlimit
              add 1 to dbbbufsize(b)
              create a dbbbuffer
              let add.bb(dbbbuffer)=slot.add.B(MAN,slotB)
              let ini.add.bb(dbbbuffer)=ini.node.B(MAN,slotB)
              let delay.bb(dbbbuffer)=delay.B(MAN,slotB)
              let ini.MAN.bb(dbbbuffer)=MAN1.B(MAN,slotB)
```

```
                let dest.MAN.bb(dbbbuffer)=MAN2.B(MAN,slotB)
                let bbddelay(dbbbuffer)=TIME.V
                file this dbbbuffer in dbbbufferq(b)
                if dbbbufsize(b)=1
                   call bbtx.add.update giving b
                always
                add 1 to bdpac.enter
             always
             add 1 to bdpac.total
          always
        always
   always

   add 1 to slotB
   wait slot.dur  .secs

loop
go to 'finish'


'fifth' while TIME.V < END.TIME.BRIDGEAB
do
   go to 'req'
'reqdone5'   if B.B(MAN,slotB)=1 and (MAN2.B(MAN,slotB)=2 or
                MAN2.B(MAN,slotB)=4)
        let b=2
        if voice.B(MAN,slotB)=2
          if vbbbufsize(b) < vbbufferlimit
                add 1 to vbbbufsize(b)
                create a vbbbuffer
                let Vadd.bb(vbbbuffer)=slot.add.B(MAN,slotB)
                let Vini.add.bb(vbbbuffer)=ini.node.B(MAN,slotB)
                let Vdelay.bb(vbbbuffer)=delay.B(MAN,slotB)
                let Vini.MAN.bb(vbbbuffer)=MAN1.B(MAN,slotB)
                let Vdest.MAN.bb(vbbbuffer)=MAN2.B(MAN,slotB)
                let bbvdelay(vbbbuffer)=TIME.V
                file this vbbbuffer in vbbbufferq(b)
                if vbbbufsize(b)=1
                   call vbbtx.add.update giving b
                always
                add 1 to bvpac.enter
           always
           add 1 to bvpac.total
        else
         if voice.B(MAN,slotB)=1
          if dbbbufsize(b) < dbbufferlimit
                add 1 to dbbbufsize(b)
                create a dbbbuffer
                let add.bb(dbbbuffer)=slot.add.B(MAN,slotB)
                let ini.add.bb(dbbbuffer)=ini.node.B(MAN,slotB)
                let delay.bb(dbbbuffer)=delay.B(MAN,slotB)
                let ini.MAN.bb(dbbbuffer)=MAN1.B(MAN,slotB)
                let dest.MAN.bb(dbbbuffer)=MAN2.B(MAN,slotB)
                let bbddelay(dbbbuffer)=TIME.V
                file this dbbbuffer in dbbbufferq(b)
                if dbbbufsize(b)=1
                   call bbtx.add.update giving b
                always
                add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
        always
   always

   add 1 to slotB
   wait slot.dur  .secs

loop
go to 'finish'


'sixth' while TIME.V < END.TIME.BRIDGEAB
do
   go to 'req'
'reqdone6'   if B.B(MAN,slotB)=1 and (MAN2.B(MAN,slotB)=1 or
                MAN2.B(MAN,slotB)=3)
        let b=5
        if voice.B(MAN,slotB)=2
          if vbbbufsize(b) < vbbufferlimit
                add 1 to vbbbufsize(b)
                create a vbbbuffer
                let Vadd.bb(vbbbuffer)=slot.add.B(MAN,slotB)
                let Vini.add.bb(vbbbuffer)=ini.node.B(MAN,slotB)
                let Vdelay.bb(vbbbuffer)=delay.B(MAN,slotB)
                let Vini.MAN.bb(vbbbuffer)=MAN1.B(MAN,slotB)
                let Vdest.MAN.bb(vbbbuffer)=MAN2.B(MAN,slotB)
                let bbvdelay(vbbbuffer)=TIME.V
                file this vbbbuffer in vbbbufferq(b)
                if vbbbufsize(b)=1
                   call vbbtx.add.update giving b
                always
                add 1 to bvpac.enter
           always
           add 1 to bvpac.total
        else
         if voice.B(MAN,slotB)=1
          if dbbbufsize(b) < dbbufferlimit
                add 1 to dbbbufsize(b)
                create a dbbbuffer
                let add.bb(dbbbuffer)=slot.add.B(MAN,slotB)
                let ini.add.bb(dbbbuffer)=ini.node.B(MAN,slotB)
                let delay.bb(dbbbuffer)=delay.B(MAN,slotB)
                let ini.MAN.bb(dbbbuffer)=MAN1.B(MAN,slotB)
                let dest.MAN.bb(dbbbuffer)=MAN2.B(MAN,slotB)
                let bbddelay(dbbbuffer)=TIME.V
                file this dbbbuffer in dbbbufferq(b)
                if dbbbufsize(b)=1
                   call bbtx.add.update giving b
                always
                add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
        always
   always

   add 1 to slotB
   wait slot.dur  .secs

loop

'finish' end  '' BRIDGEA.VD.BUSB

''
''
''
''
''

********************************************************************************
*                                                                              *
*  The BRIDGEB routines represent the homogeneous bridges that located at the  *
*  head of another bus for connecting two access MANs together.                *
*                                                                              *
********************************************************************************

process BRIDGEB.VD.BUSA


define bridge.no, slotA, MAN as integer variables
define END.TIME.BRIDGEBA as real variable


let MAN=man.noBA
let bridge.no=3
let slotA=1
add 1 to man.noBA


let END.TIME.BRIDGEBA=END.TIME+((N.stn+3)*SD+(MAN-1)/15)*slot.dur
```

```
if MAN=1
   go to 'first'
else
   if MAN=2
        go to 'second'
   else
        if MAN=3
           go to 'third'
        else
           if MAN=4
                go to 'forth'
           else
                if MAN=5
                   go to 'fifth'
                else
                   go to 'sixth'

'req'   if slotA>spf
        let slotA=1
   always

   if Vbtx.add(MAN,bridge.no)>0 and Vtx.counter.BB(MAN,bridge.no)<0
           let Vtx.counter.BB(MAN,bridge.no)=Vreq.counter.BB(MAN,bridge.no)
           let Vreq.counter.BB(MAN,bridge.no)=0
   always

   if btx.add(MAN,bridge.no)>0 and tx.counter.BB(MAN,bridge.no)<0
           let tx.counter.BB(MAN,bridge.no)=req.counter.BB(MAN,bridge.no)
           let req.counter.BB(MAN,bridge.no)=0
   always

   if VR.A(MAN,slotA)=1
        add 1 to Vreq.counter.BB(MAN,bridge.no)
   always

   if DR.A(MAN,slotA)=1
        add 1 to req.counter.BB(MAN,bridge.no)
   always

if MAN=1
   go to 'reqdone1'
else
   if MAN=2
        go to 'reqdone2'
   else
        if MAN=3
           go to 'reqdone3'
        else
           if MAN=4
                go to 'reqdone4'
           else
                if MAN=5
                   go to 'reqdone5'
                else
                   go to 'reqdone6'


'first' while TIME.V < END.TIME.BRIDGEBA
do
   go to 'req'
'reqdone1'   if B.A(MAN,slotA)=1 and MAN2.A(MAN,slotA)=5
        if voice.A(MAN,slotA)=2
          if vbbufsize(MAN2.A(MAN,slotA),2) < vbbufferlimit
                add 1 to vbbufsize(MAN2.A(MAN,slotA),2)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
                let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
                let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
                let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
                let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(MAN2.A(MAN,slotA),2)=1
                   let Vbtx.add(MAN2.A(MAN,slotA),2)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(MAN2.A(MAN,slotA),2)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
                add 1 to bvpac.enter
          always
          add 1 to bvpac.total
        else
          if voice.A(MAN,slotA)=1
            if dbbufsize(MAN2.A(MAN,slotA),2) < dbbufferlimit
                add 1 to dbbufsize(MAN2.A(MAN,slotA),2)
                create a dbbuffer
                let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
                let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
                let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
                let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
                let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(MAN2.A(MAN,slotA),2)=1
                   let btx.add(MAN2.A(MAN,slotA),2)=add.bridge(dbbuffer)
                   let btx.man(MAN2.A(MAN,slotA),2)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
                add 1 to bdpac.enter
            always
            add 1 to bdpac.total
          always
        always
   always

   add 1 to slotA
   wait slot.dur  .secs

loop
go to 'finish'

'second' while TIME.V < END.TIME.BRIDGEBA
do
   go to 'req'
'reqdone2'   if B.A(MAN,slotA)=1 and MAN2.A(MAN,slotA)=6
        if voice.A(MAN,slotA)=2
          if vbbufsize(MAN2.A(MAN,slotA),2) < vbbufferlimit
                add 1 to vbbufsize(MAN2.A(MAN,slotA),2)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
                let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
                let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
                let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
                let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(MAN2.A(MAN,slotA),2)=1
                   let Vbtx.add(MAN2.A(MAN,slotA),2)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(MAN2.A(MAN,slotA),2)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
                add 1 to bvpac.enter
          always
          add 1 to bvpac.total
        else
          if voice.A(MAN,slotA)=1
          if dbbufsize(MAN2.A(MAN,slotA),2) < dbbufferlimit
                add 1 to dbbufsize(MAN2.A(MAN,slotA),2)
                create a dbbuffer
                let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
                let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
                let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
                let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
                let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(MAN2.A(MAN,slotA),2)=1
                   let btx.add(MAN2.A(MAN,slotA),2)=add.bridge(dbbuffer)
                   let btx.man(MAN2.A(MAN,slotA),2)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
                add 1 to bdpac.enter
            always
            add 1 to bdpac.total
          always
        always
   always
```

```
   add 1 to slotA
   wait slot.dur  .secs

loop
go to 'finish'


'third' while TIME.V < END.TIME.BRIDGEBA
do
   go to 'req'
'reqdone3'   if B.A(MAN,slotA)=1 and MAN2.A(MAN,slotA)=5
        if voice.A(MAN,slotA)=2
          if vbbufsize(MAN2.A(MAN,slotA),3) < vbbufferlimit
                add 1 to vbbufsize(MAN2.A(MAN,slotA),3)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
                let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
                let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
                let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
                let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(MAN2.A(MAN,slotA),3)=1
                   let Vbtx.add(MAN2.A(MAN,slotA),3)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(MAN2.A(MAN,slotA),3)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+3)
                add 1 to bvpac.enter
           always
           add 1 to bvpac.total
        else
           if voice.A(MAN,slotA)=1
          if dbbufsize(MAN2.A(MAN,slotA),3) < dbbufferlimit
                add 1 to dbbufsize(MAN2.A(MAN,slotA),3)
                create a dbbuffer
                let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
                let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
                let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
                let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
                let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(MAN2.A(MAN,slotA),3)=1
                   let btx.add(MAN2.A(MAN,slotA),3)=add.bridge(dbbuffer)
                   let btx.man(MAN2.A(MAN,slotA),3)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+3)
                add 1 to bdpac.enter
             always
             add 1 to bdpac.total
           always
        always
   always

   add 1 to slotA
   wait slot.dur  .secs

loop
go to 'finish'


'forth' while TIME.V < END.TIME.BRIDGEBA
do
   go to 'req'
'reqdone4'   if B.A(MAN,slotA)=1 and MAN2.A(MAN,slotA)=6
        if voice.A(MAN,slotA)=2
          if vbbufsize(MAN2.A(MAN,slotA),3) < vbbufferlimit
                add 1 to vbbufsize(MAN2.A(MAN,slotA),3)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
                let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
                let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
                let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
                let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(MAN2.A(MAN,slotA),3)=1
                   let Vbtx.add(MAN2.A(MAN,slotA),3)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(MAN2.A(MAN,slotA),3)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+3)
                add 1 to bvpac.enter
           always
           add 1 to bvpac.total
        else
           if voice.A(MAN,slotA)=1
          if dbbufsize(MAN2.A(MAN,slotA),3) < dbbufferlimit
                add 1 to dbbufsize(MAN2.A(MAN,slotA),3)
                create a dbbuffer
                let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
                let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
                let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
                let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
                let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(MAN2.A(MAN,slotA),3)=1
                   let btx.add(MAN2.A(MAN,slotA),3)=add.bridge(dbbuffer)
                   let btx.man(MAN2.A(MAN,slotA),3)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+3)
                add 1 to bdpac.enter
             always
             add 1 to bdpac.total
           always
        always
   always

   add 1 to slotA
   wait slot.dur  .secs

loop
go to 'finish'


'fifth' while TIME.V < END.TIME.BRIDGEBA
do
   go to 'req'
'reqdone5'   if B.A(MAN,slotA)=1 and (MAN2.A(MAN,slotA)=3 or
                MAN2.A(MAN,slotA)=6)
        if voice.A(MAN,slotA)=2
          if vbbufsize(MAN2.A(MAN,slotA),3) < vbbufferlimit
                add 1 to vbbufsize(MAN2.A(MAN,slotA),3)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
                let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
                let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
                let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
                let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(MAN2.A(MAN,slotA),3)=1
                   let Vbtx.add(MAN2.A(MAN,slotA),3)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(MAN2.A(MAN,slotA),3)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+3)
                add 1 to bvpac.enter
           always
           add 1 to bvpac.total
        else
           if voice.A(MAN,slotA)=1
          if dbbufsize(MAN2.A(MAN,slotA),3) < dbbufferlimit
                add 1 to dbbufsize(MAN2.A(MAN,slotA),3)
                create a dbbuffer
                let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
                let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
                let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
                let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
                let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(MAN2.A(MAN,slotA),3)=1
                   let btx.add(MAN2.A(MAN,slotA),3)=add.bridge(dbbuffer)
                   let btx.man(MAN2.A(MAN,slotA),3)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+3)
                add 1 to bdpac.enter
```

```
            always
            add 1 to bdpac.total
          always
       always
   always

   add 1 to slotA
   wait slot.dur  .secs

loop
go to 'finish'

'sixth' while TIME.V < END.TIME.BRIDGEBA
do
   go to 'req'
'reqdone6'   if B.A(MAN,slotA)=1 and (MAN2.A(MAN,slotA)=4 or
             MAN2.A(MAN,slotA)=5)
       if voice.A(MAN,slotA)=2
         if vbbufsize(MAN2.A(MAN,slotA),3) < vbbufferlimit
              add 1 to vbbufsize(MAN2.A(MAN,slotA),3)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
              let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
              let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
              let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
              let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(MAN2.A(MAN,slotA),3)=1
                 let Vbtx.add(MAN2.A(MAN,slotA),3)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(MAN2.A(MAN,slotA),3)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+3)
              add 1 to bvpac.enter
          always
          add 1 to bvpac.total
       else
          if voice.A(MAN,slotA)=1
         if dbbufsize(MAN2.A(MAN,slotA),3) < dbbufferlimit
              add 1 to dbbufsize(MAN2.A(MAN,slotA),3)
              create a dbbuffer
              let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
              let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
              let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
              let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
              let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(MAN2.A(MAN,slotA),3)=1
                 let btx.add(MAN2.A(MAN,slotA),3)=add.bridge(dbbuffer)
                 let btx.man(MAN2.A(MAN,slotA),3)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+3)
              add 1 to bdpac.enter
            always
            add 1 to bdpac.total
          always
       always
   always

   add 1 to slotA
   wait slot.dur  .secs
loop

'finish' end  '' BRIDGEB.VD.BUSA

''
''
''
''
process BRIDGEB.VD.BUSB

*****************************************************************************
*                                                                           *
* this process describe the homogeneous bridge as the head station,         *
* refer to bus B, towards bus B. Slots travel along bus B are started       *
* from this bridge.                                                         *
*                                                                           *
*****************************************************************************

define bridge.no, slotB, MAN, frame.loop as integer variables

let MAN=man.noBB
let bridge.no=3
let slotB=1
let frame.loop=1
add 1 to man.noBB

while TIME.V < END.TIME+(0.5+(MAN-1)/15)*slot.dur
do
   if slotB>spf         '' initialize frames & slots
        let slotB=1
        add 1 to frame.loop
   always
   let slot.add.B(MAN,slotB)=0
   let B.B(MAN,slotB)=0
   let DR.B(MAN,slotB)=0
   let VR.B(MAN,slotB)=0
   let MAN1.B(MAN,slotB)=0
   let MAN2.B(MAN,slotB)=0
   let inter.B(MAN,slotB)=0
   let voice.B(MAN,slotB)=0
   let delay.B(MAN,slotB)=0.0
   let bridge.B(MAN,slotB)=0
   let ini.node.B(MAN,slotB)=0

   if Vtx.counter.BB(MAN,bridge.no)=0     '' TRANSMISSION FOR VOICE PACKET
        remove first vbbuffer from vbbufferq((MAN-1)*3+bridge.no)
        let slot.add.B(MAN,slotB)=Vadd.bridge(vbbuffer)
        let ini.node.B(MAN,slotB)=Vini.add(vbbuffer)
        let MAN1.B(MAN,slotB)=Vini.MAN(vbbuffer)
        let MAN2.B(MAN,slotB)=Vdest.MAN(vbbuffer)
        let inter.B(MAN,slotB)=2
        let voice.B(MAN,slotB)=2
        let bridge.B(MAN,slotB)=bridge.no
        let delay.B(MAN,slotB)=Vdelay.bridge(vbbuffer)
        let Vtx.counter.BB(MAN,bridge.no)=-1          '' reset tx.counter
        let B.B(MAN,slotB)=1                          '' set busy bit
        if frame.loop>5
let vdelay(MAN,bridge.no)=vdelay(MAN,bridge.no)+TIME.V-bvdelay(vbbuffer)
          add 1 to vnum.b(MAN,bridge.no)
          add 1 to voice.tx
        always
          destroy this vbbuffer
          subtract 1 from vbbufsize(MAN,bridge.no)
          if vbbufferq((MAN-1)*3+bridge.no) is not empty
               call vbtx.add.update giving MAN, bridge.no
          else
               let Vbtx.add(MAN,bridge.no)=0
               let Vbtx.man(MAN,bridge.no)=0
          always
   else
        if Vtx.counter.BB(MAN,bridge.no)>0
          subtract 1 from Vtx.counter.BB(MAN,bridge.no)
        else
   if Vreq.counter.BB(MAN,bridge.no)>0 and Vtx.counter.BB(MAN,bridge.no)<0
             subtract 1 from Vreq.counter.BB(MAN,bridge.no)
          else
               if tx.counter.BB(MAN,bridge.no)=0       ''TX FOR DATA PACKET
                  remove first dbbuffer from dbbufferq((MAN-1)*3+bridge.no)
                  let slot.add.B(MAN,slotB)=add.bridge(dbbuffer)
                  let ini.node.B(MAN,slotB)=ini.add(dbbuffer)
                  let MAN1.B(MAN,slotB)=ini.MAN(dbbuffer)
                  let MAN2.B(MAN,slotB)=dest.MAN(dbbuffer)
                  let inter.B(MAN,slotB)=2
                  let voice.B(MAN,slotB)=1
                  let bridge.B(MAN,slotB)=bridge.no
                  let delay.B(MAN,slotB)=delay.bridge(dbbuffer)
                  let tx.counter.BB(MAN,bridge.no)=-1  '' reset tx.counter
                  let B.B(MAN,slotB)=1                    '' set busy bit
                  if frame.loop>5
let ddelay(MAN,bridge.no)=ddelay(MAN,bridge.no)+TIME.V-bddelay(dbbuffer)
                       add 1 to dnum.b(MAN,bridge.no)
                       add 1 to data.tx
```

```
                    always
                  destroy this dbbuffer
                  subtract 1 from dbbufsize(MAN,bridge.no)
                  if dbbufferq((MAN-1)*3+bridge.no) is not empty
                        call btx.add.update giving MAN, bridge.no
                  else
                        let btx.add(MAN,bridge.no)=0
                        let btx.man(MAN,bridge.no)=0
                  always
                else
                   if tx.counter.BB(MAN,bridge.no)>0
                     subtract 1 from tx.counter.BB(MAN,bridge.no)
                   else
   if req.counter.BB(MAN,bridge.no)>0 and tx.counter.BB(MAN,bridge.no)<0
                        subtract 1 from req.counter.BB(MAN,bridge.no)
                     always
                   always
                 always
               always
          always
   always

   add 1 to slotB
   wait slot.dur .secs

loop

end  '' BRIDGEB.VD.BUSB


********************************************************************************
*                                                                              *
*  The BRIDGEC routines represent the homogeneous bridges that located at the  *
*  middle of the buses for connecting two access MANs together.                *
*                                                                              *
********************************************************************************

process BRIDGEC.VD.BUSA


define bridge.no, slotA, MAN, Tx, b, frame.loop as integer variables

let MAN=man.noCA
let bridge.no=2
let slotA=1
let frame.loop=1
add 1 to man.noCA

while TIME.V < END.TIME+((N.stn+3)*SD/2+(MAN-1)/15)*slot.dur
do
   if slotA>spf         '' initialize frames & slots
        let slotA=1
        add 1 to frame.loop
   always

  if B.A(MAN,slotA)=0
   if Vtx.counter.BA(MAN,bridge.no)=0     '' TRANSMISSION FOR VOICE PACKET
        remove first vbbuffer from vbbufferq((MAN-1)*3+bridge.no)
        let slot.add.A(MAN,slotA)=Vadd.bridge(vbbuffer)
        let ini.node.A(MAN,slotA)=Vini.add(vbbuffer)
        let MAN1.A(MAN,slotA)=Vini.MAN(vbbuffer)
        let MAN2.A(MAN,slotA)=Vdest.MAN(vbbuffer)
        let inter.A(MAN,slotA)=2
        let voice.A(MAN,slotA)=2
        let bridge.A(MAN,slotA)=bridge.no
        let delay.A(MAN,slotA)=Vdelay.bridge(vbbuffer)
        let Vtx.counter.BA(MAN,bridge.no)=-1          '' reset tx.counter
        let B.A(MAN,slotA)=1
                          '' set busy bit
        if frame.loop>5
let vdelay(MAN,bridge.no)=vdelay(MAN,bridge.no)+TIME.V-bvdelay(vbbuffer)
          add 1 to vnum.b(MAN,bridge.no)
          add 1 to voice.tx
        always

        destroy this vbbuffer
        subtract 1 from vbbufsize(MAN,bridge.no)

        if vbbufferq((MAN-1)*3+bridge.no) is not empty
              call vbtx.add.update giving MAN, bridge.no
        else
              let Vbtx.add(MAN,bridge.no)=0
              let Vbtx.man(MAN,bridge.no)=0
        always

        let Tx=1
   else
        if Vtx.counter.BA(MAN,bridge.no)>0
           subtract 1 from Vtx.counter.BA(MAN,bridge.no)
        else
   if Vreq.counter.BA(MAN,bridge.no)>0 and Vtx.counter.BA(MAN,bridge.no)<0
              subtract 1 from Vreq.counter.BA(MAN,bridge.no)
           else
              if tx.counter.BA(MAN,bridge.no)=0       ''TX FOR DATA PACKET
                 remove first dbbuffer from dbbufferq((MAN-1)*3+bridge.no)
                 let slot.add.A(MAN,slotA)=add.bridge(dbbuffer)
                 let ini.node.A(MAN,slotA)=ini.add(dbbuffer)
                 let MAN1.A(MAN,slotA)=ini.MAN(dbbuffer)
                 let MAN2.A(MAN,slotA)=dest.MAN(dbbuffer)
                 let inter.A(MAN,slotA)=2
                 let voice.A(MAN,slotA)=1
                 let bridge.A(MAN,slotA)=bridge.no
                 let delay.A(MAN,slotA)=delay.bridge(dbbuffer)
                 let tx.counter.BA(MAN,bridge.no)=-1  '' reset tx.counter
                 let B.A(MAN,slotA)=1                      '' set busy bit
                 if frame.loop>5
let ddelay(MAN,bridge.no)=ddelay(MAN,bridge.no)+TIME.V-bddelay(dbbuffer)
                       add 1 to dnum.b(MAN,bridge.no)
                       add 1 to data.tx
                 always
                 destroy this dbbuffer
                 subtract 1 from dbbufsize(MAN,bridge.no)
                 if dbbufferq((MAN-1)*3+bridge.no) is not empty
                      call btx.add.update giving MAN, bridge.no
                 else
                      let btx.add(MAN,bridge.no)=0
                      let btx.man(MAN,bridge.no)=0
                 always
                 let Tx=1
              else
                 if tx.counter.BA(MAN,bridge.no)>0
                   subtract 1 from tx.counter.BA(MAN,bridge.no)
                 else
   if req.counter.BA(MAN,bridge.no)>0 and tx.counter.BA(MAN,bridge.no)<0
                      subtract 1 from req.counter.BA(MAN,bridge.no)
                   always
                 always
              always
           always
        always
   always
  else
   if B.A(MAN,slotA)=1
     if MAN=1 and MAN2.A(MAN,slotA)=2
        if voice.A(MAN,slotA)=2
          if vbbufsize(MAN2.A(MAN,slotA),2) < vbbufferlimit
                add 1 to vbbufsize(MAN2.A(MAN,slotA),2)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
                let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
                let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
                let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
                let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
                let bvdelay(vbbuffer)=TIME.V

                if vbbufsize(MAN2.A(MAN,slotA),2)=1
                   let Vbtx.add(MAN2.A(MAN,slotA),2)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(MAN2.A(MAN,slotA),2)=Vdest.MAN(vbbuffer)
                always

                file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
                add 1 to bvpac.enter
```

```
            always
            add 1 to bvpac.total
         else
            if voice.A(MAN,slotA)=1
              if dbbufsize(MAN2.A(MAN,slotA),2) < dbbufferlimit
                 add 1 to dbbufsize(MAN2.A(MAN,slotA),2)
                 create a dbbuffer
                 let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
                 let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
                 let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
                 let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
                 let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
                 let bddelay(dbbuffer)=TIME.V
                 if dbbufsize(MAN2.A(MAN,slotA),2)=1
                    let btx.add(MAN2.A(MAN,slotA),2)=add.bridge(dbbuffer)
                    let btx.man(MAN2.A(MAN,slotA),2)=dest.MAN(dbbuffer)
                 always
                 file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
                 add 1 to bdpac.enter
              always
              add 1 to bdpac.total
            always
      always
   else


   if MAN=2 and MAN2.A(MAN,slotA)=1
      if voice.A(MAN,slotA)=2
        if vbbufsize(MAN2.A(MAN,slotA),2) < vbbufferlimit
              add 1 to vbbufsize(MAN2.A(MAN,slotA),2)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
              let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
              let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
              let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
              let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(MAN2.A(MAN,slotA),2)=1
                 let Vbtx.add(MAN2.A(MAN,slotA),2)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(MAN2.A(MAN,slotA),2)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
              add 1 to bvpac.enter
        always
        add 1 to bvpac.total
      else
         if voice.A(MAN,slotA)=1
           if dbbufsize(MAN2.A(MAN,slotA),2) < dbbufferlimit
              add 1 to dbbufsize(MAN2.A(MAN,slotA),2)
              create a dbbuffer
              let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
              let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
              let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
              let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
              let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(MAN2.A(MAN,slotA),2)=1
                 let btx.add(MAN2.A(MAN,slotA),2)=add.bridge(dbbuffer)
                 let btx.man(MAN2.A(MAN,slotA),2)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
   else


   if MAN=3 and MAN2.A(MAN,slotA)=4
      if voice.A(MAN,slotA)=2
        if vbbufsize(MAN2.A(MAN,slotA),2) < vbbufferlimit
              add 1 to vbbufsize(MAN2.A(MAN,slotA),2)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
              let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
              let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
              let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
              let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(MAN2.A(MAN,slotA),2)=1
                 let Vbtx.add(MAN2.A(MAN,slotA),2)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(MAN2.A(MAN,slotA),2)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
              add 1 to bvpac.enter
        always
        add 1 to bvpac.total
      else
         if voice.A(MAN,slotA)=1
           if dbbufsize(MAN2.A(MAN,slotA),2) < dbbufferlimit
              add 1 to dbbufsize(MAN2.A(MAN,slotA),2)
              create a dbbuffer
              let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
              let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
              let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
              let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
              let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(MAN2.A(MAN,slotA),2)=1
                 let btx.add(MAN2.A(MAN,slotA),2)=add.bridge(dbbuffer)
                 let btx.man(MAN2.A(MAN,slotA),2)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
   else


   if MAN=4 and MAN2.A(MAN,slotA)=3
      if voice.A(MAN,slotA)=2
        if vbbufsize(MAN2.A(MAN,slotA),2) < vbbufferlimit
              add 1 to vbbufsize(MAN2.A(MAN,slotA),2)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
              let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
              let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
              let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
              let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(MAN2.A(MAN,slotA),2)=1
                 let Vbtx.add(MAN2.A(MAN,slotA),2)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(MAN2.A(MAN,slotA),2)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
              add 1 to bvpac.enter
        always
        add 1 to bvpac.total
      else
         if voice.A(MAN,slotA)=1
           if dbbufsize(MAN2.A(MAN,slotA),2) < dbbufferlimit
              add 1 to dbbufsize(MAN2.A(MAN,slotA),2)
              create a dbbuffer
              let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
              let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
              let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
              let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
              let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
              let bddelay(dbbuffer)=TIME.V
              if dbbufsize(MAN2.A(MAN,slotA),2)=1
                 let btx.add(MAN2.A(MAN,slotA),2)=add.bridge(dbbuffer)
                 let btx.man(MAN2.A(MAN,slotA),2)=dest.MAN(dbbuffer)
              always
              file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+2)
              add 1 to bdpac.enter
           always
           add 1 to bdpac.total
         always
      always
```

```
else

if MAN=5 and MAN2.A(MAN,slotA)=1
   let b=3
   if voice.A(MAN,slotA)=2
     if vbbufsize(MAN2.A(MAN,slotA),b) < vbbufferlimit
           add 1 to vbbufsize(MAN2.A(MAN,slotA),b)
           create a vbbuffer
           let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
           let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
           let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
           let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
           let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
           let bvdelay(vbbuffer)=TIME.V
           if vbbufsize(MAN2.A(MAN,slotA),b)=1
              let Vbtx.add(MAN2.A(MAN,slotA),b)=Vadd.bridge(vbbuffer)
              let Vbtx.man(MAN2.A(MAN,slotA),b)=Vdest.MAN(vbbuffer)
           always
           file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+b)
           add 1 to bvpac.enter
      always
      add 1 to bvpac.total
   else
      if voice.A(MAN,slotA)=1
        if dbbufsize(MAN2.A(MAN,slotA),b) < dbbufferlimit
           add 1 to dbbufsize(MAN2.A(MAN,slotA),b)
           create a dbbuffer
           let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
           let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
           let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
           let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
           let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
           let bddelay(dbbuffer)=TIME.V
           if dbbufsize(MAN2.A(MAN,slotA),b)=1
              let btx.add(MAN2.A(MAN,slotA),b)=add.bridge(dbbuffer)
              let btx.man(MAN2.A(MAN,slotA),b)=dest.MAN(dbbuffer)
           always
           file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+b)
           add 1 to bdpac.enter
        always
        add 1 to bdpac.total
      always
   always
else

if MAN=6 and MAN2.A(MAN,slotA)=2
   let b=3
   if voice.A(MAN,slotA)=2
     if vbbufsize(MAN2.A(MAN,slotA),b) < vbbufferlimit
           add 1 to vbbufsize(MAN2.A(MAN,slotA),b)
           create a vbbuffer
           let Vadd.bridge(vbbuffer)=slot.add.A(MAN,slotA)
           let Vini.add(vbbuffer)=ini.node.A(MAN,slotA)
           let Vdelay.bridge(vbbuffer)=delay.A(MAN,slotA)
           let Vini.MAN(vbbuffer)=MAN1.A(MAN,slotA)
           let Vdest.MAN(vbbuffer)=MAN2.A(MAN,slotA)
           let bvdelay(vbbuffer)=TIME.V
           if vbbufsize(MAN2.A(MAN,slotA),b)=1
              let Vbtx.add(MAN2.A(MAN,slotA),b)=Vadd.bridge(vbbuffer)
              let Vbtx.man(MAN2.A(MAN,slotA),b)=Vdest.MAN(vbbuffer)
           always
           file this vbbuffer in vbbufferq((MAN2.A(MAN,slotA)-1)*3+b)
           add 1 to bvpac.enter
      always
      add 1 to bvpac.total
   else
      if voice.A(MAN,slotA)=1
        if dbbufsize(MAN2.A(MAN,slotA),b) < dbbufferlimit
           add 1 to dbbufsize(MAN2.A(MAN,slotA),b)
           create a dbbuffer
           let add.bridge(dbbuffer)=slot.add.A(MAN,slotA)
           let ini.add(dbbuffer)=ini.node.A(MAN,slotA)
           let delay.bridge(dbbuffer)=delay.A(MAN,slotA)
           let ini.MAN(dbbuffer)=MAN1.A(MAN,slotA)
           let dest.MAN(dbbuffer)=MAN2.A(MAN,slotA)
           let bddelay(dbbuffer)=TIME.V
           if dbbufsize(MAN2.A(MAN,slotA),b)=1
              let btx.add(MAN2.A(MAN,slotA),b)=add.bridge(dbbuffer)
              let btx.man(MAN2.A(MAN,slotA),b)=dest.MAN(dbbuffer)
           always
           file this dbbuffer in dbbufferq((MAN2.A(MAN,slotA)-1)*3+b)
           add 1 to bdpac.enter
        always
        add 1 to bdpac.total
      always
   always
else

   always
  always
  always
  always
  always
  always
  always
 always
always

if VR.A(MAN,slotA)=1
     add 1 to Vreq.counter.BB(MAN,bridge.no)
always
if DR.A(MAN,slotA)=1
     add 1 to req.counter.BB(MAN,bridge.no)
always

if Tx=0
if Vtx.counter.BB(MAN,bridge.no)<0 and 25>=Vbtx.add(MAN,bridge.no)>0 and
       VR.A(MAN,slotA)=0
         let VR.A(MAN,slotA)=1
         let Vtx.counter.BB(MAN,bridge.no)=Vreq.counter.BB(MAN,bridge.no)
         let Vreq.counter.BB(MAN,bridge.no)=0
      always
if tx.counter.BB(MAN,bridge.no)<0 and 25>=btx.add(MAN,bridge.no)>0 and
       DR.A(MAN,slotA)=0
         let DR.A(MAN,slotA)=1
         let tx.counter.BB(MAN,bridge.no)=req.counter.BB(MAN,bridge.no)
         let req.counter.BB(MAN,bridge.no)=0
      always
always

let Tx=0

add 1 to slotA
wait slot.dur .secs

loop

end  '' BRIDGEC.VD.BUSA
''
''
''
process BRIDGEC.VD.BUSB

*****************************************************************************
*                                                                           *
* this process describe the action of the homogeneous bridge located at the *
* middle of the buses towards bus B.                                        *
*                                                                           *
*****************************************************************************

define bridge.no, slotB, MAN, Tx, b, frame.loop as integer variables

let MAN=man.noCB
let bridge.no=2
let slotB=1
let frame.loop=1
add 1 to man.noCB

while TIME.V < END.TIME+((N.stn+3)*SD/2+0.5+(MAN-1)/15)*slot.dur
do
   if slotB>spf         '' initialize frames & slots
        let slotB=1
        add 1 to frame.loop
```

```
   always
 if B.B(MAN,slotB)=0
   if Vtx.counter.BB(MAN,bridge.no)=0     '' TRANSMISSION FOR VOICE PACKET
        remove first vbbuffer from vbbufferq((MAN-1)*3+bridge.no)
        let slot.add.B(MAN,slotB)=Vadd.bridge(vbbuffer)
        let ini.node.B(MAN,slotB)=Vini.add(vbbuffer)
        let MAN1.B(MAN,slotB)=Vini.MAN(vbbuffer)
        let MAN2.B(MAN,slotB)=Vdest.MAN(vbbuffer)
        let inter.B(MAN,slotB)=2
        let voice.B(MAN,slotB)=2
        let bridge.B(MAN,slotB)=bridge.no
        let delay.B(MAN,slotB)=Vdelay.bridge(vbbuffer)
        let Vtx.counter.BB(MAN,bridge.no)=-1          '' reset tx.counter
        let B.B(MAN,slotB)=1
                              '' set busy bit
        if frame.loop>5
let vdelay(MAN,bridge.no)=vdelay(MAN,bridge.no)+TIME.V-bvdelay(vbbuffer)
          add 1 to vnum.b(MAN,bridge.no)
          add 1 to voice.tx
        always

        destroy this vbbuffer
        subtract 1 from vbbufsize(MAN,bridge.no)
        if vbbufferq((MAN-1)*3+bridge.no) is not empty
                call vbtx.add.update giving MAN, bridge.no
        else
                let Vbtx.add(MAN,bridge.no)=0
                let Vbtx.man(MAN,bridge.no)=0
        always
        let Tx=1
   else
        if Vtx.counter.BB(MAN,bridge.no)>0
           subtract 1 from Vtx.counter.BB(MAN,bridge.no)
        else
   if Vreq.counter.BB(MAN,bridge.no)>0 and Vtx.counter.BB(MAN,bridge.no)<0
             subtract 1 from Vreq.counter.BB(MAN,bridge.no)
          else
                if tx.counter.BB(MAN,bridge.no)=0       ''TX FOR DATA PACKET
                   remove first dbbuffer from dbbufferq((MAN-1)*3+bridge.no)
                   let slot.add.B(MAN,slotB)=add.bridge(dbbuffer)
                   let ini.node.B(MAN,slotB)=ini.add(dbbuffer)
                   let MAN1.B(MAN,slotB)=ini.MAN(dbbuffer)
                   let MAN2.B(MAN,slotB)=dest.MAN(dbbuffer)
                   let inter.B(MAN,slotB)=2
                   let voice.B(MAN,slotB)=1
                   let bridge.B(MAN,slotB)=bridge.no
                   let delay.B(MAN,slotB)=delay.bridge(dbbuffer)
                   let tx.counter.BB(MAN,bridge.no)=-1  '' reset tx.counter
                   let B.B(MAN,slotB)=1
                     '' set busy bit
                   if frame.loop>5
let ddelay(MAN,bridge.no)=ddelay(MAN,bridge.no)+TIME.V-bddelay(dbbuffer)
                        add 1 to dnum.b(MAN,bridge.no)
                        add 1 to data.tx
                   always

                   destroy this dbbuffer
                   subtract 1 from dbbufsize(MAN,bridge.no)
                   if dbbufferq((MAN-1)*3+bridge.no) is not empty
                        call btx.add.update giving MAN, bridge.no
                   else
                        let btx.add(MAN,bridge.no)=0
                        let btx.man(MAN,bridge.no)=0
                   always
                   let Tx=1
                else
                   if tx.counter.BB(MAN,bridge.no)>0
                     subtract 1 from tx.counter.BB(MAN,bridge.no)
                   else
   if req.counter.BB(MAN,bridge.no)>0 and tx.counter.BB(MAN,bridge.no)<0
                        subtract 1 from req.counter.BB(MAN,bridge.no)
                     always
                   always
                always
             always
        always
```

```
   always
  else
   if B.B(MAN,slotB)=1

     if MAN=1 and MAN2.B(MAN,slotB)=2
        if voice.B(MAN,slotB)=2
          if vbbufsize(MAN2.B(MAN,slotB),2) < vbbufferlimit
                add 1 to vbbufsize(MAN2.B(MAN,slotB),2)
                create a vbbuffer
                let Vadd.bridge(vbbuffer)=slot.add.B(MAN,slotB)
                let Vini.add(vbbuffer)=ini.node.B(MAN,slotB)
                let Vdelay.bridge(vbbuffer)=delay.B(MAN,slotB)
                let Vini.MAN(vbbuffer)=MAN1.B(MAN,slotB)
                let Vdest.MAN(vbbuffer)=MAN2.B(MAN,slotB)
                let bvdelay(vbbuffer)=TIME.V
                if vbbufsize(MAN2.B(MAN,slotB),2)=1
                   let Vbtx.add(MAN2.B(MAN,slotB),2)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(MAN2.B(MAN,slotB),2)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((MAN2.B(MAN,slotB)-1)*3+2)
                add 1 to bvpac.enter
           always
           add 1 to bvpac.total
        else
           if voice.B(MAN,slotB)=1
             if dbbufsize(MAN2.B(MAN,slotB),2) < dbbufferlimit
                add 1 to dbbufsize(MAN2.B(MAN,slotB),2)
                create a dbbuffer
                let add.bridge(dbbuffer)=slot.add.B(MAN,slotB)
                let ini.add(dbbuffer)=ini.node.B(MAN,slotB)
                let delay.bridge(dbbuffer)=delay.B(MAN,slotB)
                let ini.MAN(dbbuffer)=MAN1.B(MAN,slotB)
                let dest.MAN(dbbuffer)=MAN2.B(MAN,slotB)
                let bddelay(dbbuffer)=TIME.V
                if dbbufsize(MAN2.B(MAN,slotB),2)=1
                   let btx.add(MAN2.B(MAN,slotB),2)=add.bridge(dbbuffer)
                   let btx.man(MAN2.B(MAN,slotB),2)=dest.MAN(dbbuffer)
                always
                file this dbbuffer in dbbufferq((MAN2.B(MAN,slotB)-1)*3+2)
                add 1 to bdpac.enter
             always
             add 1 to bdpac.total
           always
     always
   else

   if MAN=2 and MAN2.B(MAN,slotB)=1
      if voice.B(MAN,slotB)=2
        if vbbufsize(MAN2.B(MAN,slotB),2) < vbbufferlimit
              add 1 to vbbufsize(MAN2.B(MAN,slotB),2)
              create a vbbuffer
              let Vadd.bridge(vbbuffer)=slot.add.B(MAN,slotB)
              let Vini.add(vbbuffer)=ini.node.B(MAN,slotB)
              let Vdelay.bridge(vbbuffer)=delay.B(MAN,slotB)
              let Vini.MAN(vbbuffer)=MAN1.B(MAN,slotB)
              let Vdest.MAN(vbbuffer)=MAN2.B(MAN,slotB)
              let bvdelay(vbbuffer)=TIME.V
              if vbbufsize(MAN2.B(MAN,slotB),2)=1
                 let Vbtx.add(MAN2.B(MAN,slotB),2)=Vadd.bridge(vbbuffer)
                 let Vbtx.man(MAN2.B(MAN,slotB),2)=Vdest.MAN(vbbuffer)
              always
              file this vbbuffer in vbbufferq((MAN2.B(MAN,slotB)-1)*3+2)
              add 1 to bvpac.enter
         always
         add 1 to bvpac.total
      else
         if voice.B(MAN,slotB)=1
           if dbbufsize(MAN2.B(MAN,slotB),2) < dbbufferlimit
              add 1 to dbbufsize(MAN2.B(MAN,slotB),2)
              create a dbbuffer
              let add.bridge(dbbuffer)=slot.add.B(MAN,slotB)
              let ini.add(dbbuffer)=ini.node.B(MAN,slotB)
              let delay.bridge(dbbuffer)=delay.B(MAN,slotB)
              let ini.MAN(dbbuffer)=MAN1.B(MAN,slotB)
              let dest.MAN(dbbuffer)=MAN2.B(MAN,slotB)
              let bddelay(dbbuffer)=TIME.V
```

```
            if dbbufsize(MAN2.B(MAN,slotB),2)=1
               let btx.add(MAN2.B(MAN,slotB),2)=add.bridge(dbbuffer)
               let btx.man(MAN2.B(MAN,slotB),2)=dest.MAN(dbbuffer)
            always
            file this dbbuffer in dbbufferq((MAN2.B(MAN,slotB)-1)*3+2)
            add 1 to bdpac.enter
          always
          add 1 to bdpac.total
        always
     always
else


if MAN=3 and MAN2.B(MAN,slotB)=4
   if voice.B(MAN,slotB)=2
     if vbbufsize(MAN2.B(MAN,slotB),2) < vbbufferlimit
          add 1 to vbbufsize(MAN2.B(MAN,slotB),2)
          create a vbbuffer
          let Vadd.bridge(vbbuffer)=slot.add.B(MAN,slotB)
          let Vini.add(vbbuffer)=ini.node.B(MAN,slotB)
          let Vdelay.bridge(vbbuffer)=delay.B(MAN,slotB)
          let Vini.MAN(vbbuffer)=MAN1.B(MAN,slotB)
          let Vdest.MAN(vbbuffer)=MAN2.B(MAN,slotB)
          let bvdelay(vbbuffer)=TIME.V
          if vbbufsize(MAN2.B(MAN,slotB),2)=1
             let Vbtx.add(MAN2.B(MAN,slotB),2)=Vadd.bridge(vbbuffer)
             let Vbtx.man(MAN2.B(MAN,slotB),2)=Vdest.MAN(vbbuffer)
          always
          file this vbbuffer in vbbufferq((MAN2.B(MAN,slotB)-1)*3+2)
          add 1 to bvpac.enter
     always
     add 1 to bvpac.total
   else
     if voice.B(MAN,slotB)=1
       if dbbufsize(MAN2.B(MAN,slotB),2) < dbbufferlimit
          add 1 to dbbufsize(MAN2.B(MAN,slotB),2)
          create a dbbuffer
          let add.bridge(dbbuffer)=slot.add.B(MAN,slotB)
          let ini.add(dbbuffer)=ini.node.B(MAN,slotB)
          let delay.bridge(dbbuffer)=delay.B(MAN,slotB)
          let ini.MAN(dbbuffer)=MAN1.B(MAN,slotB)
          let dest.MAN(dbbuffer)=MAN2.B(MAN,slotB)
          let bddelay(dbbuffer)=TIME.V
          if dbbufsize(MAN2.B(MAN,slotB),2)=1
             let btx.add(MAN2.B(MAN,slotB),2)=add.bridge(dbbuffer)
             let btx.man(MAN2.B(MAN,slotB),2)=dest.MAN(dbbuffer)
          always
          file this dbbuffer in dbbufferq((MAN2.B(MAN,slotB)-1)*3+2)
          add 1 to bdpac.enter
        always
        add 1 to bdpac.total
     always
   always
else

if MAN=4 and MAN2.B(MAN,slotB)=3
   if voice.B(MAN,slotB)=2
     if vbbufsize(MAN2.B(MAN,slotB),2) < vbbufferlimit
          add 1 to vbbufsize(MAN2.B(MAN,slotB),2)
          create a vbbuffer
          let Vadd.bridge(vbbuffer)=slot.add.B(MAN,slotB)
          let Vini.add(vbbuffer)=ini.node.B(MAN,slotB)
          let Vdelay.bridge(vbbuffer)=delay.B(MAN,slotB)
          let Vini.MAN(vbbuffer)=MAN1.B(MAN,slotB)
          let Vdest.MAN(vbbuffer)=MAN2.B(MAN,slotB)
          let bvdelay(vbbuffer)=TIME.V
          if vbbufsize(MAN2.B(MAN,slotB),2)=1
             let Vbtx.add(MAN2.B(MAN,slotB),2)=Vadd.bridge(vbbuffer)
             let Vbtx.man(MAN2.B(MAN,slotB),2)=Vdest.MAN(vbbuffer)
          always
          file this vbbuffer in vbbufferq((MAN2.B(MAN,slotB)-1)*3+2)
          add 1 to bvpac.enter
     always
     add 1 to bvpac.total
   else
     if voice.B(MAN,slotB)=1
```



```
       if dbbufsize(MAN2.B(MAN,slotB),2) < dbbufferlimit
          add 1 to dbbufsize(MAN2.B(MAN,slotB),2)
          create a dbbuffer
          let add.bridge(dbbuffer)=slot.add.B(MAN,slotB)
          let ini.add(dbbuffer)=ini.node.B(MAN,slotB)
          let delay.bridge(dbbuffer)=delay.B(MAN,slotB)
          let ini.MAN(dbbuffer)=MAN1.B(MAN,slotB)
          let dest.MAN(dbbuffer)=MAN2.B(MAN,slotB)
          let bddelay(dbbuffer)=TIME.V
          if dbbufsize(MAN2.B(MAN,slotB),2)=1
             let btx.add(MAN2.B(MAN,slotB),2)=add.bridge(dbbuffer)
             let btx.man(MAN2.B(MAN,slotB),2)=dest.MAN(dbbuffer)
          always
          file this dbbuffer in dbbufferq((MAN2.B(MAN,slotB)-1)*3+2)
          add 1 to bdpac.enter
        always
        add 1 to bdpac.total
     always
   always
else


if MAN=5 and MAN2.B(MAN,slotB)=1
   let b=3
   if voice.B(MAN,slotB)=2
     if vbbufsize(MAN2.B(MAN,slotB),b) < vbbufferlimit
          add 1 to vbbufsize(MAN2.B(MAN,slotB),b)
          create a vbbuffer
          let Vadd.bridge(vbbuffer)=slot.add.B(MAN,slotB)
          let Vini.add(vbbuffer)=ini.node.B(MAN,slotB)
          let Vdelay.bridge(vbbuffer)=delay.B(MAN,slotB)
          let Vini.MAN(vbbuffer)=MAN1.B(MAN,slotB)
          let Vdest.MAN(vbbuffer)=MAN2.B(MAN,slotB)
          let bvdelay(vbbuffer)=TIME.V
          if vbbufsize(MAN2.B(MAN,slotB),b)=1
             let Vbtx.add(MAN2.B(MAN,slotB),b)=Vadd.bridge(vbbuffer)
             let Vbtx.man(MAN2.B(MAN,slotB),b)=Vdest.MAN(vbbuffer)
          always
          file this vbbuffer in vbbufferq((MAN2.B(MAN,slotB)-1)*3+b)
          add 1 to bvpac.enter
     always
     add 1 to bvpac.total
   else
     if voice.B(MAN,slotB)=1
       if dbbufsize(MAN2.B(MAN,slotB),b) < dbbufferlimit
          add 1 to dbbufsize(MAN2.B(MAN,slotB),b)
          create a dbbuffer
          let add.bridge(dbbuffer)=slot.add.B(MAN,slotB)
          let ini.add(dbbuffer)=ini.node.B(MAN,slotB)
          let delay.bridge(dbbuffer)=delay.B(MAN,slotB)
          let ini.MAN(dbbuffer)=MAN1.B(MAN,slotB)
          let dest.MAN(dbbuffer)=MAN2.B(MAN,slotB)
          let bddelay(dbbuffer)=TIME.V
          if dbbufsize(MAN2.B(MAN,slotB),b)=1
             let btx.add(MAN2.B(MAN,slotB),b)=add.bridge(dbbuffer)
             let btx.man(MAN2.B(MAN,slotB),b)=dest.MAN(dbbuffer)
          always
          file this dbbuffer in dbbufferq((MAN2.B(MAN,slotB)-1)*3+b)
          add 1 to bdpac.enter
        always
        add 1 to bdpac.total
     always
   always
else

if MAN=6 and MAN2.B(MAN,slotB)=2
   let b=3
   if voice.B(MAN,slotB)=2
     if vbbufsize(MAN2.B(MAN,slotB),b) < vbbufferlimit
          add 1 to vbbufsize(MAN2.B(MAN,slotB),b)
          create a vbbuffer
          let Vadd.bridge(vbbuffer)=slot.add.B(MAN,slotB)
          let Vini.add(vbbuffer)=ini.node.B(MAN,slotB)
          let Vdelay.bridge(vbbuffer)=delay.B(MAN,slotB)
          let Vini.MAN(vbbuffer)=MAN1.B(MAN,slotB)
          let Vdest.MAN(vbbuffer)=MAN2.B(MAN,slotB)
```

```
                if vbbufsize(MAN2.B(MAN,slotB),b)=1
                   let Vbtx.add(MAN2.B(MAN,slotB),b)=Vadd.bridge(vbbuffer)
                   let Vbtx.man(MAN2.B(MAN,slotB),b)=Vdest.MAN(vbbuffer)
                always
                file this vbbuffer in vbbufferq((MAN2.B(MAN,slotB)-1)*3+b)
                add 1 to bvpac.enter
             always
             add 1 to bvpac.total
          else
             if voice.B(MAN,slotB)=1
               if dbbufsize(MAN2.B(MAN,slotB),b) < dbbufferlimit
                  add 1 to dbbufsize(MAN2.B(MAN,slotB),b)
                  create a dbbuffer
                  let add.bridge(dbbuffer)=slot.add.B(MAN,slotB)
                  let ini.add(dbbuffer)=ini.node.B(MAN,slotB)
                  let delay.bridge(dbbuffer)=delay.B(MAN,slotB)
                  let ini.MAN(dbbuffer)=MAN1.B(MAN,slotB)
                  let dest.MAN(dbbuffer)=MAN2.B(MAN,slotB)
                  let bddelay(dbbuffer)=TIME.V
                  if dbbufsize(MAN2.B(MAN,slotB),b)=1
                     let btx.add(MAN2.B(MAN,slotB),b)=add.bridge(dbbuffer)
                     let btx.man(MAN2.B(MAN,slotB),b)=dest.MAN(dbbuffer)
                  always
                  file this dbbuffer in dbbufferq((MAN2.B(MAN,slotB)-1)*3+b)
                  add 1 to bdpac.enter
               always
               add 1 to bdpac.total
             always
          always
       always
       always
       always
       always
       always
       always
     always
   always

   if VR.B(MAN,slotB)=1
        add 1 to Vreq.counter.BA(MAN,bridge.no)
   always

   if DR.B(MAN,slotB)=1
        add 1 to req.counter.BA(MAN,bridge.no)
   always

   if Tx=0
if Vtx.counter.BA(MAN,bridge.no)<0 and 25<Vbtx.add(MAN,bridge.no) and
         VR.B(MAN,slotB)=0
           let VR.B(MAN,slotB)=1
           let Vtx.counter.BA(MAN,bridge.no)=Vreq.counter.BA(MAN,bridge.no)
           let Vreq.counter.BA(MAN,bridge.no)=0
        always

if tx.counter.BA(MAN,bridge.no)<0 and 25<btx.add(MAN,bridge.no) and
         DR.B(MAN,slotB)=0
           let DR.B(MAN,slotB)=1
           let tx.counter.BA(MAN,bridge.no)=req.counter.BA(MAN,bridge.no)
           let req.counter.BA(MAN,bridge.no)=0
        always
   always

   let Tx=0

   add 1 to slotB
   wait slot.dur .secs

loop

end  '' BRIDGEC.VD.BUSB

*******************************************************************************
*                                                                             *
*      The STATION routines describe the action of each access MAN node.      *
*                                                                             *
*******************************************************************************
```

```
process STATION.VD.BUSA

''
'' this process describe the action of a node towards bus A
'' and calculate the packets delay.
''

define MAN, throw, a, b, c, d, e as integer variables  '' index
define stn.no, slotA, frame.loop, Tx as integer variables
define END.TIME.A, vtime.wait as real variables

let stn.no=II
let MAN=man.noSA
add 1 to man.noSA
if man.noSA=7
   let man.noSA=1
   let man.noPS=1
   let man.noVS=1
   add 1 to II
always
add 1 to stn.act
let slotA=1
let frame.loop=1
let Tx=0
let END.TIME.A=END.TIME+(stn.no*SD+(MAN-1)/15)*slot.dur

while TIME.V < END.TIME.A
do
   if slotA>spf
        let slotA=1
        add 1 to frame.loop
   always

   if B.A(MAN,slotA)=0
        if Vtx.counter.A(MAN,stn.no)=0       '' TRANSMISSION FOR VOICE CELL
          remove first vsbuffer from vsbufferq((MAN-1)*50+stn.no)
          let slot.add.A(MAN,slotA)=Vtemp.add(vsbuffer)
          let ini.node.A(MAN,slotA)=stn.no
          let MAN1.A(MAN,slotA)=MAN
          let MAN2.A(MAN,slotA)=Vsdest.man(vsbuffer)
          let delay.A(MAN,slotA)=Vdelay.temp(vsbuffer)
          let voice.A(MAN,slotA)=2
          let bridge.A(MAN,slotA)=0
          let Vtx.counter.A(MAN,stn.no)=-1          '' reset tx.counter
          let B.A(MAN,slotA)=1                      '' set busy bit
          if MAN1.A(MAN,slotA)<>MAN2.A(MAN,slotA)
               let inter.A(MAN,slotA)=2
          else
               let inter.A(MAN,slotA)=1
          always
          destroy this vsbuffer
          subtract 1 from vsbufsize(MAN,stn.no)

          if vsbufferq((MAN-1)*50+stn.no) is not empty
               call vstx.add.update giving MAN, stn.no
          else
               let Vtx.add(MAN,stn.no)=0
               let Vtx.man(MAN,stn.no)=0
          always

          if frame.loop>5
            add 1 to voice.tx
          always
          let Tx=1
        else
          if Vtx.counter.A(MAN,stn.no)>0  '' slot reserved
               subtract 1 from Vtx.counter.A(MAN,stn.no)
          else
             if Vreq.counter.A(MAN,stn.no)>0 and Vtx.counter.A(MAN,stn.no)<0
                subtract 1 from Vreq.counter.A(MAN,stn.no)
             else
                if tx.counter.A(MAN,stn.no)=0 '' TRANSMISSION FOR DATA PAC.
                  remove first dsbuffer from dsbufferq((MAN-1)*50+stn.no)
                  let slot.add.A(MAN,slotA)=temp.add(dsbuffer)
                  let ini.node.A(MAN,slotA)=stn.no
```

```
                let MAN1.A(MAN,slotA)=MAN
                let MAN2.A(MAN,slotA)=sdest.man(dsbuffer)
                let delay.A(MAN,slotA)=delay.temp(dsbuffer)
                let voice.A(MAN,slotA)=1
                let bridge.A(MAN,slotA)=0
                let tx.counter.A(MAN,stn.no)=-1  '' reset tx.counter
                let B.A(MAN,slotA)=1             '' set busy bit
                if MAN1.A(MAN,slotA)<>MAN2.A(MAN,slotA)
                     let inter.A(MAN,slotA)=2
                else
                     let inter.A(MAN,slotA)=1
                always
                destroy this dsbuffer
                subtract 1 from dsbufsize(MAN,stn.no)
                if dsbufferq((MAN-1)*50+stn.no) is not empty
                     call stx.add.update giving MAN, stn.no
                else
                     let tx.add(MAN,stn.no)=0
                     let tx.man(MAN,stn.no)=0
                always
                if frame.loop>5
                  add 1 to data.tx
                always
                let Tx=1
              else
                if tx.counter.A(MAN,stn.no)>0  '' slot reserved
                     subtract 1 from tx.counter.A(MAN,stn.no)
                else
            if req.counter.A(MAN,stn.no)>0 and tx.counter.A(MAN,stn.no)<0
                     subtract 1 from req.counter.A(MAN,stn.no)
                  always
                always
              always
            always
          always
        always
    else
 if B.A(MAN,slotA)=1 and MAN2.A(MAN,slotA)=MAN and
    slot.add.A(MAN,slotA)=stn.no and frame.loop>5
        let a=MAN1.A(MAN,slotA)
        let b=ini.node.A(MAN,slotA)
        let c=inter.A(MAN,slotA)
        let d=voice.A(MAN,slotA)
        let e=bridge.A(MAN,slotA)
        let delay(a,b,c,d)=delay(a,b,c,d)+TIME.V-delay.A(MAN,slotA)
        add 1 to num.re(a,b,c,d)
        if c=2 and e>0
let delay.via.bridge(a,e,d)=delay.via.bridge(a,e,d)+TIME.V-delay.A(MAN,slotA)
          add 1 to num.via.bridge(a,e,d)
        always
 always
always

''
'' discard voice packets that cannot be
'' transmitted within 5.5ms
''
''
        let throw=0
        while vsbufferq((MAN-1)*50+stn.no) is not empty
        do
          remove first vsbuffer from vsbufferq((MAN-1)*50+stn.no)
          let vtime.wait=TIME.V-Vdelay.temp(vsbuffer)
          if vtime.wait > 0.0055
                destroy this vsbuffer
                subtract 1 from vsbufsize(MAN,stn.no)
                add 1 to discard.vpacket(MAN,stn.no)
                let throw=1
                if Vtx.counter.A(MAN,stn.no)>=0
Vreq.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)+Vtx.counter.A(MAN,stn.no)
Vtx.counter.A(MAN,stn.no)=-1
                else
                   if Vtx.counter.B(MAN,stn.no)>=0
Vreq.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)+Vtx.counter.B(MAN,stn.no)
Vtx.counter.B(MAN,stn.no)=-1
                 always
```

```
                always
                let Vtx.add(MAN,stn.no)=0
                let Vtx.man(MAN,stn.no)=0
          else
                file vsbuffer first in vsbufferq((MAN-1)*50+stn.no)
                if throw=1
                        call vstx.add.update giving MAN, stn.no
                always
                go to 'OUT'
          always
        loop

'OUT'   if VR.A(MAN,slotA)=1
          add 1 to Vreq.counter.B(MAN,stn.no)
        always
        if DR.A(MAN,slotA)=1
          add 1 to req.counter.B(MAN,stn.no)
        always

   if Tx=0
        if VR.A(MAN,slotA)=0 and Vtx.counter.B(MAN,stn.no)<0 and
          Vtx.add(MAN,stn.no)>0
                if MAN=Vtx.man(MAN,stn.no)
                   go to 'localV'
                else
                   go to 'interV'
                always

'data'  if DR.A(MAN,slotA)=0 and tx.counter.B(MAN,stn.no)<0 and
          tx.add(MAN,stn.no)>0
                if MAN=tx.man(MAN,stn.no)
                   go to 'localD'
                else
                   go to 'interD'
                always
   always

'sreqdone'   let Tx=0
   add 1 to slotA
   wait slot.dur .secs

loop
go to 'finish'

'localV' if stn.no>Vtx.add(MAN,stn.no)>0
           let VR.A(MAN,slotA)=1
           let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
           let Vreq.counter.B(MAN,stn.no)=0
         always
         go to 'data'

'interV' if MAN=1
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=3 or
                Vtx.man(MAN,stn.no)=4 or Vtx.man(MAN,stn.no)=6)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (Vtx.man(MAN,stn.no)=2 or
                Vtx.man(MAN,stn.no)=3 or Vtx.man(MAN,stn.no)=4 or
                Vtx.man(MAN,stn.no)=6)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
            always
           always
         else
          if MAN=2
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=3 or
                Vtx.man(MAN,stn.no)=4 or Vtx.man(MAN,stn.no)=5)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=3 or Vtx.man(MAN,stn.no)=4 or
```

```
   go to 'data'

'localD' if stn.no>tx.add(MAN,stn.no)>0
           let DR.A(MAN,slotA)=1
           let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
           let req.counter.B(MAN,stn.no)=0
         always
         go to 'sreqdone'

'interD' if MAN=1
           if stn.no<=25 and (tx.man(MAN,stn.no)=3 or
                tx.man(MAN,stn.no)=4 or tx.man(MAN,stn.no)=6)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (tx.man(MAN,stn.no)=2 or
                tx.man(MAN,stn.no)=3 or tx.man(MAN,stn.no)=4 or
                tx.man(MAN,stn.no)=6)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
             always
           always
         else

          if MAN=2
           if stn.no<=25 and (tx.man(MAN,stn.no)=3 or
                tx.man(MAN,stn.no)=4 or tx.man(MAN,stn.no)=5)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=3 or tx.man(MAN,stn.no)=4 or
                tx.man(MAN,stn.no)=5)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
             always
           always
         else

          if MAN=3
           if stn.no<=25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=2 or tx.man(MAN,stn.no)=6)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=2 or tx.man(MAN,stn.no)=4 or
                tx.man(MAN,stn.no)=6)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
             always
           always
         else

          if MAN=4
           if stn.no<=25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=2 or tx.man(MAN,stn.no)=5)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=2 or tx.man(MAN,stn.no)=3 or
                tx.man(MAN,stn.no)=5)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
             always
           always
         else
```



```
                Vtx.man(MAN,stn.no)=5)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
             always
           always
         else

          if MAN=3
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=2 or Vtx.man(MAN,stn.no)=6)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=2 or Vtx.man(MAN,stn.no)=4 or
                Vtx.man(MAN,stn.no)=6)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
             always
           always
         else

          if MAN=4
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=2 or Vtx.man(MAN,stn.no)=5)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=2 or Vtx.man(MAN,stn.no)=3 or
                Vtx.man(MAN,stn.no)=5)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
             always
           always
         else

          if MAN=5
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=2 or Vtx.man(MAN,stn.no)=4)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=2 or Vtx.man(MAN,stn.no)=4)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
             always
           always
         else

          if MAN=6
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=1 or Vtx.man(MAN,stn.no)=3)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=2 or Vtx.man(MAN,stn.no)=3)
                let VR.A(MAN,slotA)=1
                let Vtx.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)
                let Vreq.counter.B(MAN,stn.no)=0
             always
           always
         always
        always
       always
      always
     always
    always
```

```
          if MAN=5
           if stn.no<=25 and (tx.man(MAN,stn.no)=2 or tx.man(MAN,stn.no)=4)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=2 or tx.man(MAN,stn.no)=4)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
             always
           always
          else

          if MAN=6
           if stn.no<=25 and (tx.man(MAN,stn.no)=1 or tx.man(MAN,stn.no)=3)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
           else
             if stn.no>25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=2 or tx.man(MAN,stn.no)=3)
                let DR.A(MAN,slotA)=1
                let tx.counter.B(MAN,stn.no)=req.counter.B(MAN,stn.no)
                let req.counter.B(MAN,stn.no)=0
             always
           always
         always
        always
       always
      always
     always
    always
   go to 'sreqdone'

'finish' end  '' STATION.VD.BUSA
''
''
''
process STATION.VD.BUSB

*****************************************************************
*                                                               *
* this process describe the action of a node towards bus B      *
* and calculate the packets delay.                              *
*                                                               *
*****************************************************************

define MAN, throw, a, b, c, d, e as integer variables  '' index
define stn.no, slotB, frame.loop, Tx as integer variables
define END.TIME.B, vtime.wait as real variables

let stn.no=N.stn-LL+3
let MAN=man.noSB
add 1 to man.noSB
if man.noSB=7
   let man.noSB=1
   add 1 to LL
always
let slotB=1
let frame.loop=1
let Tx=0
let END.TIME.B=END.TIME+((N.stn+3-stn.no)*SD+0.5+(MAN-1)/15)*slot.dur

while TIME.V < END.TIME.B
do
   if slotB>spf
        let slotB=1
        add 1 to frame.loop
   always

   if B.B(MAN,slotB)=0
        if Vtx.counter.B(MAN,stn.no)=0       '' TRANSMISSION FOR VOICE CELL
          remove first vsbuffer from vsbufferq((MAN-1)*50+stn.no)
          let slot.add.B(MAN,slotB)=Vtemp.add(vsbuffer)
          let ini.node.B(MAN,slotB)=stn.no
          let MAN1.B(MAN,slotB)=MAN
          let MAN2.B(MAN,slotB)=Vsdest.man(vsbuffer)
          let delay.B(MAN,slotB)=Vdelay.temp(vsbuffer)
          let voice.B(MAN,slotB)=2
          let bridge.B(MAN,slotB)=0
          let Vtx.counter.B(MAN,stn.no)=-1          '' reset tx.counter
          let B.B(MAN,slotB)=1                      '' set busy bit
          if MAN1.B(MAN,slotB)<>MAN2.B(MAN,slotB)
               let inter.B(MAN,slotB)=2
          else
               let inter.B(MAN,slotB)=1
          always
          destroy this vsbuffer
          subtract 1 from vsbufsize(MAN,stn.no)

          if vsbufferq((MAN-1)*50+stn.no) is not empty
               call vstx.add.update giving MAN, stn.no
          else
               let Vtx.add(MAN,stn.no)=0
               let Vtx.man(MAN,stn.no)=0
          always

          if frame.loop>5
            add 1 to voice.tx
          always
          let Tx=1
        else
          if Vtx.counter.B(MAN,stn.no)>0  '' slot reserved
               subtract 1 from Vtx.counter.B(MAN,stn.no)
          else
             if Vreq.counter.B(MAN,stn.no)>0 and Vtx.counter.B(MAN,stn.no)<0
                subtract 1 from Vreq.counter.B(MAN,stn.no)
             else
                if tx.counter.B(MAN,stn.no)=0 '' TRANSMISSION FOR DATA PAC.
                  remove first dsbuffer from dsbufferq((MAN-1)*50+stn.no)
                  let slot.add.B(MAN,slotB)=temp.add(dsbuffer)
                  let ini.node.B(MAN,slotB)=stn.no
                  let MAN1.B(MAN,slotB)=MAN
                  let MAN2.B(MAN,slotB)=sdest.man(dsbuffer)
                  let delay.B(MAN,slotB)=delay.temp(dsbuffer)
                  let voice.B(MAN,slotB)=1
                  let bridge.B(MAN,slotB)=0
                  let tx.counter.B(MAN,stn.no)=-1  '' reset tx.counter
                  let B.B(MAN,slotB)=1             '' set busy bit
                  if MAN1.B(MAN,slotB)<>MAN2.B(MAN,slotB)
                       let inter.B(MAN,slotB)=2
                  else
                       let inter.B(MAN,slotB)=1
                  always
                  destroy this dsbuffer
                  subtract 1 from dsbufsize(MAN,stn.no)
                  if dsbufferq((MAN-1)*50+stn.no) is not empty
                       call stx.add.update giving MAN, stn.no
                  else
                       let tx.add(MAN,stn.no)=0
                       let tx.man(MAN,stn.no)=0
                  always
                  if frame.loop>5
                    add 1 to data.tx
                  always
                  let Tx=1
                else
                  if tx.counter.B(MAN,stn.no)>0  '' slot reserved
                       subtract 1 from tx.counter.B(MAN,stn.no)
                  else
                if req.counter.B(MAN,stn.no)>0 and tx.counter.B(MAN,stn.no)<0
                       subtract 1 from req.counter.B(MAN,stn.no)
                    always
                  always
                 always
               always
             always
   else
if B.B(MAN,slotB)=1 and MAN2.B(MAN,slotB)=MAN and
```

```
    slot.add.B(MAN,slotB)=stn.no and frame.loop>5
        let a=MAN1.B(MAN,slotB)
        let b=ini.node.B(MAN,slotB)
        let c=inter.B(MAN,slotB)
        let d=voice.B(MAN,slotB)
        let e=bridge.B(MAN,slotB)
        let delay(a,b,c,d)=delay(a,b,c,d)+TIME.V-delay.B(MAN,slotB)
        add 1 to num.re(a,b,c,d)
        if c=2 and e>0
let delay.via.bridge(a,e,d)=delay.via.bridge(a,e,d)+TIME.V-delay.B(MAN,slotB)
          add 1 to num.via.bridge(a,e,d)
        always
 always
always
        let throw=0
        while vsbufferq((MAN-1)*50+stn.no) is not empty
        do
          remove first vsbuffer from vsbufferq((MAN-1)*50+stn.no)
          let vtime.wait=TIME.V-Vdelay.temp(vsbuffer)
          if vtime.wait > 0.0055
                destroy this vsbuffer
                subtract 1 from vsbufsize(MAN,stn.no)
                add 1 to discard.vpacket(MAN,stn.no)
                let throw=1
                if Vtx.counter.A(MAN,stn.no)>=0
Vreq.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)+Vtx.counter.A(MAN,stn.no)
Vtx.counter.A(MAN,stn.no)=-1
                else
                   if Vtx.counter.B(MAN,stn.no)>=0
Vreq.counter.B(MAN,stn.no)=Vreq.counter.B(MAN,stn.no)+Vtx.counter.B(MAN,stn.no)
Vtx.counter.B(MAN,stn.no)=-1
                     always
                always
                let Vtx.add(MAN,stn.no)=0
                let Vtx.man(MAN,stn.no)=0
          else
                file vsbuffer first in vsbufferq((MAN-1)*50+stn.no)
                if throw=1
                       call vstx.add.update giving MAN, stn.no
                always
                go to 'OUT'
          always
        loop

'OUT'   if VR.B(MAN,slotB)=1
           add 1 to Vreq.counter.A(MAN,stn.no)
        always

        if DR.B(MAN,slotB)=1
           add 1 to req.counter.A(MAN,stn.no)
        always

   if Tx=0
        if VR.B(MAN,slotB)=0 and Vtx.counter.A(MAN,stn.no)<0 and
           Vtx.add(MAN,stn.no)>0
                if MAN=Vtx.man(MAN,stn.no)
                   go to 'localV'
                else
                   go to 'interV'
                always


'data'  if DR.B(MAN,slotB)=0 and tx.counter.A(MAN,stn.no)<0 and
           tx.add(MAN,stn.no)>0
                if MAN=tx.man(MAN,stn.no)
                   go to 'localD'
                else
                   go to 'interD'
                always
   always

'sreqdone'   let Tx=0
   add 1 to slotB
   wait slot.dur .secs

loop
```

```
go to 'finish'

'localV' if stn.no<Vtx.add(MAN,stn.no)
           let VR.B(MAN,slotB)=1
           let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
           let Vreq.counter.A(MAN,stn.no)=0
         always
         go to 'data'

'interV' if MAN=1
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=2 or
                Vtx.man(MAN,stn.no)=5)
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and Vtx.man(MAN,stn.no)=5
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
             always
           always
         else

          if MAN=2
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=6)
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and Vtx.man(MAN,stn.no)=6
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
             always
           always
         else

          if MAN=3
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=4 or
                Vtx.man(MAN,stn.no)=5)
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and Vtx.man(MAN,stn.no)=5
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
             always
           always
         else

          if MAN=4
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=3 or
                Vtx.man(MAN,stn.no)=6)
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and Vtx.man(MAN,stn.no)=6
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
             always
           always
         else

          if MAN=5
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=1 or
                Vtx.man(MAN,stn.no)=3 or Vtx.man(MAN,stn.no)=6)
                let VR.B(MAN,slotB)=1
                let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
                let Vreq.counter.A(MAN,stn.no)=0
           else
```

```
            if stn.no>25 and (Vtx.man(MAN,stn.no)=3 or Vtx.man(MAN,stn.no)=6)
               let VR.B(MAN,slotB)=1
               let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
               let Vreq.counter.A(MAN,stn.no)=0
             always
           always
         else
          if MAN=6
           if stn.no<=25 and (Vtx.man(MAN,stn.no)=2 or
                Vtx.man(MAN,stn.no)=4 or Vtx.man(MAN,stn.no)=5)
               let VR.B(MAN,slotB)=1
               let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
               let Vreq.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and (Vtx.man(MAN,stn.no)=4 or Vtx.man(MAN,stn.no)=5)
               let VR.B(MAN,slotB)=1
               let Vtx.counter.A(MAN,stn.no)=Vreq.counter.A(MAN,stn.no)
               let Vreq.counter.A(MAN,stn.no)=0
             always
           always
          always
         always
        always
       always
      always
     always
    go to 'data'


'localD' if stn.no<tx.add(MAN,stn.no)
           let DR.B(MAN,slotB)=1
           let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
           let req.counter.A(MAN,stn.no)=0
         always
         go to 'sreqdone'


'interD' if MAN=1
           if stn.no<=25 and (tx.man(MAN,stn.no)=2 or
                tx.man(MAN,stn.no)=5)
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and tx.man(MAN,stn.no)=5
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
             always
           always
         else
          if MAN=2
           if stn.no<=25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=6)
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and tx.man(MAN,stn.no)=6
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
             always
           always
         else
          if MAN=3
           if stn.no<=25 and (tx.man(MAN,stn.no)=4 or
                tx.man(MAN,stn.no)=5)
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and tx.man(MAN,stn.no)=5
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
```

```
             always
           always
         else
          if MAN=4
           if stn.no<=25 and (tx.man(MAN,stn.no)=3 or
                tx.man(MAN,stn.no)=6)
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and tx.man(MAN,stn.no)=6
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
             always
           always
         else
          if MAN=5
           if stn.no<=25 and (tx.man(MAN,stn.no)=1 or
                tx.man(MAN,stn.no)=3 or tx.man(MAN,stn.no)=6)
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and (tx.man(MAN,stn.no)=3 or tx.man(MAN,stn.no)=6)
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
             always
           always
         else
          if MAN=6
           if stn.no<=25 and (tx.man(MAN,stn.no)=2 or
                tx.man(MAN,stn.no)=4 or tx.man(MAN,stn.no)=5)
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
           else
             if stn.no>25 and (tx.man(MAN,stn.no)=4 or tx.man(MAN,stn.no)=5)
               let DR.B(MAN,slotB)=1
               let tx.counter.A(MAN,stn.no)=req.counter.A(MAN,stn.no)
               let req.counter.A(MAN,stn.no)=0
             always
           always
         always
        always
       always
      always
     always
    go to 'sreqdone'

'finish' end  '' STATION.VD.BUSB


*****************************************************************************
*                                                                           *
*  The STOP routine is responsible to stop the simulation and calculate all *
*  the statistical results in the simulation.                               *
*                                                                           *
*****************************************************************************

process STOP.SIM

define K, L, sum.tx, total.discard.vpacket, total.vpacket as integer variables
define pac.arr.rate, ddelay.intra, vdelay.intra, ddelay.inter, vdelay.inter,
       U.d, percent.vpacket.discard, U.v, U, BU.v, BU.d as real variables
define avg.delay.d.intra, avg.delay.v.intra, avg.delay.d.inter,
       avg.delay.v.inter, avg.Vdelay.via.bridge,
       avg.ddelay.via.bridge as 1-dimensional, real array
define meandelay.d.intra, meandelay.v.intra, meandelay.d.inter,
       meandelay.v.inter, mean.Vdelay.via.bridge,
       mean.ddelay.via.bridge as 2-dimensional, real array
define sum.discard.vpacket, sum.vpacket as 1-dimensional, integer array
define bridge.vpacket, bridge.dpacket as integer variables
define avg.bridge.vdelay, avg.bridge.ddelay as real variables
```

```
define bridge.vdelay, bridge.ddelay as 2-dimensional, real array

reserve meandelay.d.intra(*,*), meandelay.v.intra(*,*),
        meandelay.d.inter(*,*), meandelay.v.inter(*,*) as 6 by N.stn+2
reserve avg.delay.d.intra(*), avg.delay.v.intra(*), avg.delay.d.inter(*),
        avg.delay.v.inter(*), sum.discard.vpacket(*), sum.vpacket(*) as 6
reserve bridge.vdelay(*,*), bridge.ddelay(*,*), mean.Vdelay.via.bridge(*,*),
        mean.ddelay.via.bridge(*,*) as 6 by 3
reserve avg.Vdelay.via.bridge(*), avg.ddelay.via.bridge(*) as 3

''
'' Intra-MAN traffic (per MAN)
''

for L= 1 to 6
do
  for K = 1 to N.stn+2  '' find the mean data packet(intra) delay/MAN/node
    with num.re(L,K,1,1)>0
  do
   let meandelay.d.intra(L,K)=delay(L,K,1,1)/num.re(L,K,1,1)
  loop
loop
for L = 1 to 6
do
 for K = 1 to N.stn+2   '' find the mean delay of data packets(intra)/MAN
        with meandelay.d.intra(L,K)>0.0
   compute
        avg.delay.d.intra(L) as the mean of meandelay.d.intra(L,K)

loop

for L = 1 to 6
do
  for K = 1 to N.stn+2  '' find the mean voice cell(intra) delay/MAN/node
        with num.re(L,K,1,2)>0
  do
   let meandelay.v.intra(L,K)=delay(L,K,1,2)/num.re(L,K,1,2)
  loop
loop
for L = 1 to 6
do
 for K = 1 to N.stn+2   '' find the mean delay of voice cells(intra)/MAN
        with meandelay.v.intra(L,K)>0.0
   compute
        avg.delay.v.intra(L) as the mean of meandelay.v.intra(L,K)

loop

''
'' Inter-MAN traffic (per MAN)
''

for L= 1 to 6
do
  for K = 1 to N.stn+2  '' find the mean data packet(inter) delay/MAN/node
        with num.re(L,K,2,1)>0
  do
   let meandelay.d.inter(L,K)=delay(L,K,2,1)/num.re(L,K,2,1)
  loop
loop
for L = 1 to 6
do
 for K = 1 to N.stn+2   '' find the mean delay of data packets(inter)/MAN
        with meandelay.d.inter(L,K)>0.0
   compute
        avg.delay.d.inter(L) as the mean of meandelay.d.inter(L,K)

loop

for L = 1 to 6
do
  for K = 1 to N.stn+2  '' find the mean voice cell(inter) delay/MAN/node
        with num.re(L,K,2,2)>0
  do
   let meandelay.v.inter(L,K)=delay(L,K,2,2)/num.re(L,K,2,2)
  loop
```

```
loop
for L = 1 to 6
do
 for K = 1 to N.stn+2   '' find the mean delay of voice cells(inter)/MAN
        with meandelay.v.inter(L,K)>0.0
   compute
        avg.delay.v.inter(L) as the mean of meandelay.v.inter(L,K)

loop

''
'' Average voice and data delay
''

 for K = 1 to 6         '' find the mean delay of data packets(intra)
        with avg.delay.d.intra(K)>0.0
   compute
        ddelay.intra as the mean of avg.delay.d.intra(K)

 for K = 1 to 6         '' find the mean delay of voice cells(intra)
        with avg.delay.v.intra(K)>0.0
   compute
        vdelay.intra as the mean of avg.delay.v.intra(K)

 for K = 1 to 6         '' find the mean delay of data packets(inter)
        with avg.delay.d.inter(K)>0.0
   compute
        ddelay.inter as the mean of avg.delay.d.inter(K)

 for K = 1 to 6         '' find the mean delay of voice cells(inter)
        with avg.delay.v.inter(K)>0.0
   compute
        vdelay.inter as the mean of avg.delay.v.inter(K)

''
'' Bridges waiting delay and Utilization
''

for K=1 to 6
   for L=1 to 3
   compute
        bridge.vpacket as the sum of vnum.b(K,L)

for K=1 to 6
   for L=1 to 3
   compute
        bridge.dpacket as the sum of dnum.b(K,L)

for K=1 to 6
do
   for L=1 to 3
        with vnum.b(K,L)>0
   do
        let bridge.vdelay(K,L)=vdelay(K,L)/vnum.b(K,L)
   loop
loop
for K=1 to 6
do
   for L=1 to 3
        with dnum.b(K,L)>0
   do
        let bridge.ddelay(K,L)=ddelay(K,L)/dnum.b(K,L)
   loop
loop

for K=1 to 6
   for L=1 to 3
        with bridge.vdelay(K,L)>0.0
   compute
        avg.bridge.vdelay as the mean of bridge.vdelay(K,L)

for K=1 to 6
   for L=1 to 3
        with bridge.ddelay(K,L)>0.0
   compute
        avg.bridge.ddelay as the mean of bridge.ddelay(K,L)
```

```
''
'' Average Tx. delay of packets via which bridge
''

for K=1 to 6
do
   for L=1 to 3
        with num.via.bridge(K,L,2)>0
   do
 let mean.Vdelay.via.bridge(K,L)=delay.via.bridge(K,L,2)/num.via.bridge(K,L,2)
   loop
loop

for K=1 to 6
do
   for L=1 to 3
        with num.via.bridge(K,L,1)>0
   do
 let mean.ddelay.via.bridge(K,L)=delay.via.bridge(K,L,1)/num.via.bridge(K,L,1)
   loop
loop

for L=1 to 3
do
   for K=1 to 6
        with mean.Vdelay.via.bridge(K,L)>0.0
   compute
        avg.Vdelay.via.bridge(L) as the mean of mean.Vdelay.via.bridge(K,L)
loop

for L=1 to 3
do
   for K=1 to 6
        with mean.ddelay.via.bridge(K,L)>0.0
   compute
        avg.ddelay.via.bridge(L) as the mean of mean.ddelay.via.bridge(K,L)
loop

''
''
''

for K = 1 to 6
do
  for L = 1 to N.stn+2
   compute
        sum.discard.vpacket(K) as the sum of discard.vpacket(K,L)
loop

for K = 1 to 6
do
  for L = 1 to N.stn+2
   compute
        sum.vpacket(K) as the sum of Vnum(K,L)
loop

  for L = 1 to 6
   compute
        total.discard.vpacket as the sum of sum.discard.vpacket(L)

  for L = 1 to 6
   compute
        total.vpacket as the sum of sum.vpacket(L)


let sum.tx=voice.tx+data.tx
let U = sum.tx/(num.slot-5*spf)/2/6
let U.d = data.tx/(num.slot-5*spf)/2/6
let U.v = voice.tx/(num.slot-5*spf)/2/6*48/53
let BU.v=bridge.vpacket/(num.slot-5*spf)/2/6*48/53
let BU.d=bridge.dpacket/(num.slot-5*spf)/2/6
let pac.arr.rate=Poisson.arr*stn.act*slot.dur
let percent.vpacket.discard=total.discard.vpacket/total.vpacket*100


print 24 lines with (N.stn+2)*6, SD, Poisson.arr, pac.arr.rate, num.slot,
                    ddelay.intra*1000, vdelay.intra*1000, ddelay.inter*1000,
                    vdelay.inter*1000, U, U.d, U.v, stn.act,
                    Vstn.act, num.buffer, percent.vpacket.discard,
                    percent.inter.traffic, Rv/1000, BU.v, BU.d,
                    avg.bridge.vdelay*1000, avg.bridge.ddelay*1000
thus
        STANDARD (Inter and Intra-MAN):
                          number of nodes:          *  nodes
   distance between 2 adjacent nodes:               *  slots
                 Poisson arrival rate:     ****.***  packets/sec
                  Packet arrival rate:     ****.***  packets/slot
      number of slots for simulation:               *  slots/bus
 average data packet tx.delay(intra):     *.******  ms
average voice packet tx.delay(intra):     *.******  ms
 average data packet tx.delay(inter):     *.******  ms
average voice packet tx.delay(inter):     *.******  ms
                          Utilization:      **.****
              data packet Utilization:      **.****
             voice packet Utilization:      **.****
                    nodes activated:          *
               voice calls activated:          * per node
                    number of buffer:          * per data traffic
percentage of voice packet discarded:       *.** %
     percentage of inter-MAN traffic:          * %
                  voice coding rate:          * kbps
     Bridge voice packet Utilization:      **.****
      Bridge data packet Utilization:      **.****
average voice packet w.delay(bridge):     *.****** ms
 average data packet w.delay(bridge):     *.****** ms


for K=1 to 6
do
   print 3 line with K, avg.delay.d.intra(K)*1000, avg.delay.v.intra(K)*1000,
                    avg.delay.d.inter(K)*1000, avg.delay.v.inter(K)*1000,
                    sum.discard.vpacket(K)/sum.vpacket(K)*100
   thus
MAN: *          avg.ddelay(intra):  *.****** ms; avg.vdelay(intra): *.****** ms;
                avg.ddelay(inter):  *.****** ms; avg.vdelay(inter): *.****** ms;
                % V.discard: *.** %
   skip 1 output lines
   for L=1 to 3
   do
   print 1 line with L, bridge.vdelay(K,L)*1000, bridge.ddelay(K,L)*1000 thus
  Bridge: *    avg.vdelay:  *.****** ms; avg.ddelay:  *.****** ms
   loop
   skip 1 output lines
loop


print 1 line with voice.tx, data.tx thus
# of v.packet tx:       *; # of d.packet tx:       *


print 1 line with bridge.vpacket, bridge.dpacket thus
# of bridges' v.packet tx:          *; # of bridges' d.packet tx:          *

for K=1 to 6
do
        for L=1 to 3
        do
        print 1 line with K, L, vnum.b(K,L), dnum.b(K,L) thus
MAN:  *; bridge: *; # of voice packet tx:       *; # of data packet tx:       *
        loop
loop

''
''
''
''
''

skip 1 output lines
print 4 line with bvpac.total, bvpac.enter, bvpac.total-bvpac.enter,
                  bdpac.total, bdpac.enter, bdpac.total-bdpac.enter
```

```
thus
total # of voice packet enter bridges:          *
# voice packet enter buffer:          *; # of voice packet discarded:      *
total # of data packet enter bridges:                *
# data packet enter buffer:           *; # of data packet discarded:       *

''
''
skip 1 output lines
for K=1 to 6
do
       print 1 line with K, man.count(K) thus
   MAN:  *; # of data packets destinated:       *
loop

skip 1 output lines
for K=1 to 6
do
       for L=1 to 3
       do
       print 1 line with K, L, vbbufsize(K,L), dbbufsize(K,L) thus
 MAN:  *; Bridge:  *; v.packet in buffer:       *; d.packet in buffer:      *
       loop
loop
''
''
for L=1 to 6
do
   skip 1 output lines
   print 2 lines with L thus
MAN:  *
                        intra                              inter
   for K=1 to N.stn+2
   do
       print 1 line with K, meandelay.d.intra(L,K)*1000,
       meandelay.v.intra(L,K)*1000, meandelay.d.inter(L,K)*1000,
       meandelay.v.inter(L,K)*1000 thus
 node:  *; d: *.****** ms; v: *.****** ms; d: *.****** ms; v: *.****** ms;
   loop
loop

skip 1 output lines
for L=1 to 3
do
print 1 line with L, avg.Vdelay.via.bridge(L)*1000,
                avg.ddelay.via.bridge(L)*1000 thus
 Bridge: *; avg.voice.tx.delay:  *.****** ms; avg.data.tx.delay:  *.****** ms
   for K=1 to 6
   do
       print 1 line with K, mean.Vdelay.via.bridge(K,L)*1000,
                        mean.ddelay.via.bridge(K,L)*1000 thus
    MAN: *; avg.voice.tx.delay:  *.****** ms; avg.data.tx.delay:  *.****** ms
   loop
   skip 1 output lines
loop
''
''
for K=1 to 6*(N.stn+2)
do
   while vsbufferq(K) is not empty
   do
       remove first vsbuffer from vsbufferq(K)
       destroy this vsbuffer
   loop
   while dsbufferq(K) is not empty
   do
       remove first dsbuffer from dsbufferq(K)
       destroy this dsbuffer
   loop
loop
for K=1 to 6*3
do
   while vbbufferq(K) is not empty
   do
       remove first vbbuffer from vbbufferq(K)
       destroy this vbbuffer
   loop
   while dbbufferq(K) is not empty
   do
       remove first dbbuffer from dbbufferq(K)
       destroy this dbbuffer
   loop
loop

close 8
end   '' STOP.SIM
```

References

[1] Andrew D. Malyan, Leslie J. Ng, Victor C. M. Leung and Robert W. Donaldson, "Network Architecture and Signalling for Wireless Personal Communications," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 6, pp. 830–841, August 1993.

[2] Nanjian Qian, "Call Control Signalling for Personal Communications Over Interconnected Metropolitan Area Network," Master's thesis, UBC Department of Electrical Engineering M.A.Sc. Thesis, April 1993.

[3] IEEE 802.6 Committee, *Distributed Queue Dual Bus (DQDB) Subnetwork of a Metropolitan Area Network (MAN).* Institute of Electrical and Electronics Engineers, Inc., New York, NY, 1991.

[4] I. S. Venieris, J. D. Angelopoulos and G. I. Stassinopoulos, "DQDB MAN as a Transit Network for ATM CPNs," *Proc. INFOCOM, Chicago*, pp. 2351–2357, 1992.

[5] Wiiliam Stallings, *Data and Computer Communication.* New York: Macmillan Publishing Company, 4th ed., 1994.

[6] Oran Sharon and Adrian Segall, "A simple Scheme for Slot Reuse Without Latency for a Dual Bus Configuration," *IEEE/ACM Transactions on Networking*, vol. 1, no. 1, pp. 96–104, February 1993.

[7] P. J. Kiviat, H. Markowitz ,and R. Villanueva, *Simscript II.5 Programming Language.* California: CACI, 1987.

[8] Wiiliam H. Beyer, *CRC Standard Mathematical Tables.* Florida: CRC Press Inc., 28th ed., 1987.

[9] Kiriakos Apostolidis, "A Reservation Protocol for Packet Voice and Data Integration in Unidirectional Bus Networks," *IEEE Transactions on Communications*, vol. 41, no. 3, pp. 478–485, March 1993.

[10] Xiaomei Qian, Sharad Kumar, Dhadesugoor Vaman, Shukri Wakid and David Cypher, "Provision of Isochronous Service on IEEE 802.6," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 1778–1784, February/March/April 1994.

[11] L. J. Ng, R. W. Donaldson and A. D. Malayan, "Distributed Architectures and Databases for Intelligent Personal Communication Networks," *ICWC*, pp. 300–304, 1992.

[12] James Lane, "ATM kints voice, data on any net," *IEEE SPECTRUM*, pp. 42–45, February 1994.

[13] Cem Ersoy and Shivendra S. Panwar, "Topological Design of Interconnected LAN/MAN Networks," *IEEE Journal on Selected Areas in Communications*, vol. 11, no. 8, pp. 1172–1182, October 1993.

[14] Oran Sharon and Adrian Segall, "On the Efficiency of Slot Reuse in the Dual Bus Configuration," *IEEE/ACM Transactions on Networking*, vol. 2, no. 1, pp. 89–100, February 1994.

[15] Kathleen S. Meier-Hellstern and David J. Goodman, "Network Protocols for the Cellular Packet Switch," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 89–100, February/March/April 1994.

[16] Kin K. Leung, Bhaskar Sengupta and Raymond W. Yeung, "A Credit Manager for Traffic Regulation in High-Speed Networks: A Queueing Analysis," *IEEE Transactions on Networking*, vol. 1, no. 2, pp. 236–245, April 1993.

[17] Riaz Ahmad and Fred Halsall, "Interconnecting High-Speed LANs and Backbones," *IEEE Network Magazine*, pp. 36–43, September 1993.

[18] B. Grela-M'Poko, M. Mehmet-Ali and J. F. Hayes, "An Approximate Performance Analysis of Interconnected Token-Passing Rings," *IEEE Transactions on Communications*, vol. 41, no. 8, pp. 1142–1146, August 1993.

[19] Joseph C. S. Cheung, Mark A. Beach and Joseph P. McGeehan, "Network Planning for Third-Generation Mobil Radio Systems," *IEEE Communications Magazine*, vol. 32, no. 11, pp. 54–59, November 1994.

[20] Wolfgang Fischer, Eugen Wallmeier, Thomas Worster, Simon P. Davis and Andrew Hayter, "Data Communications Using ATM: Architecture, Protocols, and Resource Management," *IEEE Communications Magazine*, vol. 32, no. 8, pp. 24–33, August 1994.

[21] Brett J. Vickers and Tatsuya Suda, "Connectionless Service for Public ATM Networks," *IEEE Communications Magazine*, vol. 32, no. 8, pp. 34–42, August 1994.

[22] Toshikazu Suzuki, "ATM Adaptation Layer Protocol," *IEEE Communications Magazine*, vol. 32, no. 4, pp. 80–83, April 1994.

[23] Brian W. Unger, Douglas J. Goetz and Stephen W. Maryka, "Simulation of SS7 Common Channel Signaling," *IEEE Communications Magazine*, vol. 32, no. 3, pp. 52–62, March 1994.

[24] William E. Burr, Shukri Wakid, Xiaomei Qian and Dhadesugoor Vaman, "A Comparsion of FDDI Asynchronous Mode and DQDB Queue Arbitrated Mode Data Transmission for Metropolitan Area Network Applications," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 1758–1768, February/March/April 1994.

[25] Douglas W. Browning, "Flow Control in High-Speed Communication Networks," *IEEE Transactions on Communications*, vol. 42, no. 7, pp. 2480–2489, July 1994.

[26] Dimitri Bertsekas and Robert Gallager, *Data Networks*. New Jersey: Prentice-Hall Inc, 1992.

[27] Leslie J. Ng, "Distributed Architectures and Database for Personal Communications Networks and Services," Master's thesis, UBC Department of Electrical Engineering M.A.Sc. Thesis, March 1993.