# Performance Analysis of an ATM Network with Multimedia Traffic - A Simulation Study

**Diane Elisabeth Gan**

**A thesis submitted in partial fulfilment of the requirements of the University of Greenwich for the degree of Doctor of Philosophy**

**March 1998**

# Abstract

Traffic and congestion control are important in enabling ATM networks to maintain the Quality of Service (QoS) required by end users. A Call Admission Control (CAC) strategy ensures that the network has sufficient resources available at the start of each call, but this does not prevent a traffic source from violating the negotiated contract. A policing strategy (User Parameter Control (UPC)) is also required to enforce the negotiated rates for a particular connection and to protect conforming users from network overload.

The aim of this work is to investigate traffic policing and bandwidth management at the User to Network Interface (UNI). A policing function is proposed which is based on the leaky bucket (LB) which offers improved performance for both real time (RT) traffic such as speech and video and non-real time (non-RT) traffic, mainly data by taking into account the QoS requirements. A video cell in violation of the negotiated bit rate causes the remainder of the slice to be discarded. This 'tail clipping' provides protection for the decoder from damaged video slices. Speech cells are coded using a frequency domain coder, which places the most significant bits of a double speech sample into a high priority cell and the least significant bits into a high priority cell. In the case of congestion, the low priority cell can be discarded with little impact on the intelligibility of the received speech. However, data cells require loss-free delivery and are buffered rather than being discarded or tagged for subsequent deletion. This triple strategy is termed the super leaky bucket (S-LB).

Separate queues for RT and non-RT traffic, are also proposed at the multiplexer, with non-pre-emptive priority service for RT traffic if the queue exceeds a predetermined threshold. If the RT queue continues to grow beyond a second threshold, then all low priority cells (mainly speech) are discarded. This scheme protects non-RT traffic from being tagged and subsequently discarded, by queuing the cells and also by throttling back non-RT sources during periods of congestion. It also prevents the RT cells from being delayed excessively in the multiplexer queue.

A simulation model has been designed and implemented to test the proposal. Realistic sources have been incorporated into the model to simulate the types of traffic which could be expected on an ATM network.

The results show that the S-LB outperforms the standard LB for video cells. The number of cells discarded and the resulting number of damaged video slices are significantly reduced. Dual queues with cyclic service at the multiplexer also reduce the delays experienced by RT cells. The QoS for all categories of traffic is preserved.

# Contents

# List of Figures

# List of Tables

# Acknowledgements

This work could not have been completed without the help and support of many friends and colleagues both here at the University and outside. However, there are a number of people who deserve a special mention. Firstly my husband, Nelson, who has worked tirelessly on the household chores to enable me to have sufficient time to write this thesis. My children and their respective partners who have always been a constant source of encouragement, supplying morale boosting support at the most strategic moments. I would like to thank Jason Scates for his prompt technical support in times of crisis. A special thank you to Dr Nigel Burton, of GPT, who never tired of answering my questions and giving support and advice throughout. Thanks also to Dr Keith Dimond of the University of Kent for his advice and encouragement throughout our association. And last but not least, a very special thank you to Dr Sati McKenzie, without whom this could never have been completed. Her patience, help and guidance, not to mention proof reading, have been invaluable throughout the course of this work.

# Chapter 1 - Introduction

## 1.1 Overview

The multimedia age is upon us and the new Broadband Integrated Services Digital Network (B-ISDN) is poised to provide the transport needs of the future, with the potential of huge bandwidths, which will accommodate new services. Asynchronous transfer mode (ATM) has been chosen as the switching and multiplexing technique for B-ISDN. These new networks use digital technology, and ATM is the best choice digital communication technology for multimedia networks.

Multimedia traffic is a combination of traffic types and these are classified into 5 main categories - data, speech, video, image and graphics. This discussion is mainly concerned with speech, data and video. Interactive multimedia will benefit from a single network that can provide :-

- high bandwidth
- digital switching and transmission
- controllable quality of service
- flexibility to carry any type of traffic

A big issue with ATM is the quality of service (QoS), which the user sees, and it is quantified by parameters such as cell error ratio, end-to-end delay, delay variation and cell loss. Cell loss may occur when data is corrupted by noise or when cells are dropped due to congestion. Error rates for digital transmission over optical fibres are very low. However, errors do occur, and this can cause significant visual glitches for video and annoying clicks during telephone conversations. Cell delay variation (CDV) also called jitter, is caused when clumping occurs as cells travel through the network switches and are delayed in queues. The inter-arrival time between adjacent cells shortens to the point where it affects the network and impacts on other traffic. A viewer watching a fast sports coverage might not notice a high cell error ratio, but would notice a high cell delay. Medical imaging requires extremely high-quality images and users would prefer a low cell error rate, while tolerating delays. So the QoS required does depend on the application used.

ATM defines a packet switched connection orientated communications protocol which provides the functionality of the OSI physical and datalink layers. Some higher level functions are also included. It uses fixed size packets called cells to facilitate fast switching and transmission. The protocol is simpler than those used in earlier data networks such as X.25. Features such as the link by link flow control have been omitted for improved efficiency and speed.

ATM is based on a slotted time system. The length of a slot is defined by the cell transmission time. A synchronous stream of fixed size slots are accessed asynchronously, as required. No bandwidth is consumed unless a cell is actually being transmitted. ATM is able to accommodate variable bit rate (VBR) transmission because time slots are allocated asynchronously. The allocation of bandwidth is not based on the peak rate, so it is not wasted when bursty sources (e.g. video) do not utilise it continuously. This means that several bursty sources can be multiplexed together, to achieve a bandwidth gain.

ATM does allow both bandwidth reservation and statistical multiplexing to be performed. As yet, the best policies for managing network bandwidth are not yet apparent. This is because the demands to be placed on the network are not fully understood, as there are very few fully operational ATM networks in existence.

The types of traffic expected to use ATM networks fall into two main categories and these are:-

- constant bit rate (CBR)
- variable bit rate (VBR)

These can further be sub-divided into real time (RT) and non-real time (non-RT) traffic. Video and speech are RT traffic sources, while data falls into the non-RT category. RT traffic has particular timing requirements, in that it cannot tolerate excessive delays or jitter. Data and video traffic also require error-free delivery, while speech can tolerate some loss of data (up to 10%) and still be intelligible.

ATM is attractive for future multimedia services chiefly because of the ability to support VBR transmission.

Benefits include :-

- reduced buffer delay in video codecs
- near constant picture quality
- efficient use of transmission capacity

However, additional problems arise due to :-

- higher probability of lost cells due to congestion
- greater cell delay variation
- higher network management costs

Congestion occurs when the load on the network or a portion of it exceeds capacity. This may result in cell loss and increased cell delay variation. This is particularly serious for VBR sources. Cells may be lost if all VBR sources output at their peak rates, due to buffers and switches becoming full and overflowing. Congestion also causes increased CDV as cells crossing the network experience varying waiting times and hence have varying transit delays through the network. The allocation of bandwidth also becomes more complex. If bandwidth is allocated at the peak rate, the utilisation of the link is very inefficient, since the mean bit rate is much lower than the peak bit rate.

Cell losses cause two main problems for video sources. Firstly, each cell contains 48 bytes of data, but because this may be compressed video it may represent a large area of the picture and the resulting degradation may persist for a while. Secondly, since compression algorithms make use of variable length coding, the loss of some data also causes loss of synchronisation and renders subsequent data unusable. For RT video there is no time to re-transmit the data. CDV also causes a major problem for clock recovery and synchronisation of the audio and video streams, which can lead to the loss of lip synchronisation within the picture.

Older networks, such as X.25 and TCP/IP, used various reactive control schemes such as telling upstream nodes to reduce transmission or re-routing packets round congestion hot spots. These techniques are not efficient on high speed networks. Proactive and preventative strategies are required. Control and policing at the User to Network Interface (UNI) is essential. At connection time a contract is negotiated between the originating system and the network. The contract will have QoS characteristics associated with it, such as mean and peak rates and tolerable cell loss requirements. The network then monitors all connections for contract

violations and this is called source policing. A Call Admission Control (CAC) strategy ensures that the network has sufficient resources available at the start of each call, but this does not prevent a traffic source from violating the negotiated contract. In addition a policing strategy (UPC) is also required to enforce the negotiated rates for a particular connection and to protect conforming calls by preventing network overload.

A widely used algorithm for policing at the UNI is the leaky bucket monitoring scheme. The leaky bucket behaves as a virtual first in first out (FIFO) buffer, which does not store cells or delay them. A counter is incremented each time a cell arrives. The counter is periodically decremented at a rate previously negotiated at call set-up. The leaky bucket allows bursty applications to gain access to their peak rate for brief periods, providing the average rate remains below the threshold. If a source transmits at a rate higher than negotiated then the leaky bucket 'overflows'. When this happens, the excess cells are either deleted or tagged as low priority and allowed onto the network. If congestion occurs at a switch then low priority cells may be discarded in favour of high priority cells.

Dimensioning the leaky bucket when policing VBR sources has been the subject of much research. It is critical to the performance as seen by the end user, as a VBR traffic source which is conforming to its requested mean rate may have cells discarded when a long burst of cells arrives which causes the leaky bucket to 'overflow'. Different implementations of the leaky bucket exist.

## 1.2 Contribution of Thesis

The aim of the present work is to investigate policing and bandwidth control at the UNI. The UNI is the access point to the network which can be a bottleneck and impose unacceptable delays on individual cells. If these are RT cells, then any additional delay can result in some cells arriving too late to be included in the decoded bit stream.

1. The leaky bucket may be positioned before or after the multiplexer. Does the positioning of the leaky bucket and the multiplexer have any effect either on the queue lengths or on the performance of the policing function itself?

2. Can the proposed policing function :-

   i. improve QoS for RT cells by :-
      - preventing excessive loss of video cells
      - reduce the number of damaged video slices
      - minimise lost speech cells
      - preventing excessive delays to RT cells

   ii. maintain QoS for non-RT cells by :-
      - giving reasonable end-to-end delays for data cells
      - protect data cells from being discarded

A simulation model has been designed and implemented to test these proposals.

The proposed policing mechanism is based on a triple strategy leaky bucket and takes into account the various QoS requirements of the different types of traffic which may be found on an ATM network. By not allowing traffic which would cause an overload, onto the network, and by monitoring the connections that are already active, congestion can be kept to a minimum.

The work done by Niestegge [NEIST90] and Di Nitto [NITT92] on the leaky bucket is extended in this work. The policing strategy proposed differentiates between RT traffic (speech and video) and non-RT traffic (mainly data). The RT traffic experiences minimal delays, while video traffic has its cell loss controlled so as to minimise the impact of lost cells on the final picture. If a video cell is tagged by the policing function and allowed onto the network, there is always the possibility that the cell may be discarded as it crosses the network. This could desynchronise the video stream, and cause annoying artefacts in the picture. In this work the rest of the video cells following are deleted as well. Tail-end clipping is performed at the slice layer within the video stream and the remainder of the slice is discarded. A slice is the smallest resynchronisation point within the video stream.

A frequency domain coder is assumed for speech coding, which divides two speech samples into low and high frequency components. These are then packetised into two cells, with the low frequency components of the signal in a high priority cell and the less important high frequency components in a low priority cell. The leaky bucket tags any violating speech cells as low priority cells which are allowed onto the network. Any tagged cells may see a higher loss probability

within the network, however it has been found that the loss of a speech cell has minimal impact on the end user.

Non-RT traffic is protected from being discarded by using a buffered leaky bucket, which does not tag or discard cells, but delays them in a buffer until the counter fails below the threshold.

Separate queues with cyclic service are proposed, for RT and non-RT traffic, within the multiplexer. Previous studies by [GAN95] have shown that cyclic service at the multiplexer benefits the RT traffic, while the non-RT traffic is slightly delayed, but not excessively. The work done by [HAV94], [CHANG94] and [KIM96a] on the scheduling of cells within a multiplexer is extended. The proposed multiplexer scheduling policy gives non-pre-emptive priority service for RT traffic, if the queue exceeds a predetermined threshold. If the RT queue continues to grow beyond a second threshold, then all low priority speech cells are discarded. Thresholds are also included for throttling back non-RT traffic if the non-RT queue grows too large during periods of congestion.

The proposed strategy is compared with the performance of a cyclic queue with a virtual leaky bucket. The first part of this work has been to model a set of realistic traffic sources (speech, video and data) and use them to look at the performance of a small scale ATM network as seen by the end user. A user site generates multimedia traffic, which passes through a leaky bucket and is then multiplexed onto the network. Cells are routed across the ATM network using dynamic routing.

The results of the simulations show that cyclic queues do prevent RT cells from being blocked by non-RT cells at the multiplexer. RT cells experience minimal delays, even when the utilisation at the UNI is high. There are fewer video cells discarded and a smaller number of slices are damaged. Only low priority speech cells are discarded when the RT queue length becomes too long, to keep the queuing delays for RT cells to a minimum. This results in less than 0.01% of the low priority cells being dropped. Data cells experience no losses and the overall non-RT delays incurred are within bounds.

## 1.3 The Layout of the Thesis

The layout of the rest of this thesis is as follows. Chapter 2 provides a general background to related research in this area. The ATM layers are discussed and some relevant protocol issues are introduced. Currently used general traffic models and specific speech, video and data models are examined. The modelling of multiplexers is reviewed and current ATM switch techniques are discussed. Connection admission and congestion control techniques and algorithms are also reviewed. Chapter 3 describes the simulation model used for the experiments and the proposed policing strategy. Chapter 4 presents the results of the simulation experiments and examines the implications. A summary and general conclusions are given in Chapter 5.

# Chapter 2 - Background and Related Work

## 2.1 Historical Perspective

The first transfer mode used to relay information from one place to another was telegraphy. A message was transported from relay station to relay station by human operators, using a key to generate pulses down a wire. Each message had an attached source and destination address and the operator decided what to do with individual messages. This was a form of "packet switching" and it used binary codes e.g. Morse, to transfer the information.

At the end of the last century, circuit switching was introduced using POTS (Plain Old Telephone System). Initially, the circuit was set up manually by the operator, but eventually automatic electromechanical switches were introduced. These were eventually replaced by electrical switches. The transfer mode of the POTS is still circuit switching, and since most of the traffic over these networks is voice transfer there was no reason to change this.

Up to 50% of a telephone conversation comprises silence. For expensive circuits, e.g. satellites, it was found that economies could be made by filling the silence period of one conversation with the active period of another. This was done by coding speech using TASI (Time Assignment by Speech Interpolation). However, the additional complexity and cost of equipment to make these gains was not justified for short to medium distance connections.

When computers were first connected together to transfer data, modems were used over the existing circuit switched telephone network. A modem was necessary to convert digital computer data into an analogue signal and back again. Since the POTS provided an existing network which covered most of the globe, it was a natural choice to make use of it to transfer data.

Computer data applications are characteristically very bursty in nature. Typically they have silence periods even greater than the 50% found in speech and so circuit switching was not the ideal solution for data applications. In the sixties, X.25 for packet switching was standardised. Packet switching became popular, since companies did not have to pay for circuits that stood idle for most of the time. Network resources were only charged for when data was actually

transferred. The disadvantage of packet switching was that each packet carried an additional overhead in the form of a packet header, used for routing, error correction, flow control and sequence numbering.

To ensure the necessary reliability over low quality transmission links, link-by-link error control was essential. Data packets were error-checked and re-transmissions were requested if a packet was corrupted. Re-transmissions increased the delays experienced, but improved the accuracy of received data. X.25 packets are variable length, which also requires complex buffer management within the network. Slow transmission speeds, typically 64 Kb/s, caused large delays, but since the traffic carried was not real time, this did not present a problem.

The concept of Integrated System Digital Networks (ISDN) was first proposed in 1984, providing a digital network with much higher bandwidth allowing a wider variety of applications to use the same network. This made possible the concept of transferring real time and non-real time traffic on the same network, in an integrated way, which paved the way for the introduction of narrowband ISDN (N-ISDN). There are two interfaces, one for basic access and the other for primary access. Basic access provides two 64 Kb/s channels and a 16 Kb/s signalling channel. Primary access has a channel capacity of 1.544 Mb/s and 2.048 Mb/s, which included a 64 Kb/s signalling channel. It was soon realised that these channel capacities were too restrictive and that higher bandwidths were required for the interconnection of LAN's and for high density data, such as video and image transfer.

Frame relay was conceived during the standardisation of N-ISDN and was introduced in 1988 as an alternative to X.25. Frame relay is a streamlined technique for packet switching, which has less overhead than traditional packet switching. This is due to the fact that ISDN transmission is over high-quality, reliable links, many of which are optical fibre, so link-by-link error control is no longer needed. Error and flow control can then be performed end-to-end. By streamlining the protocol, lower delays and higher throughput are possible.

As the telecommunication companies replaced the copper-based trunk lines of the POTS with optical fibre, the capacity available increased enormously. This made possible the concept of Broadband ISDN (B-ISDN) as a single high-speed, integrated digital network, which will be deployed to create a world-wide networking technology based on a common set of user

interfaces. Currently B-ISDN interfaces support access speeds up to 622 Mb/s with the possibility of faster rates as switching technology improves.

As the standardisation of B-ISDN progressed CCITT (now known as ITU-T) classified four categories of Broadband applications. These are:-

1) conversational services - e.g. video phones, video conferencing
2) messaging services - e.g. video mail service, document mail service
3) retrieval services - e.g. text, data, graphics, sound, still and moving pictures
4) distribution services - (i) with user control - full channel broadcast
   (ii) without user control

Asynchronous transfer mode (ATM) has been chosen by ITU-T as the multiplexing and switching protocol for B-ISDN. This has influenced design choices for B-ISDN which include cell payload size and the use of virtual channels. It has been designed to operate at significantly higher data rates than frame relay. ATM attempts to provide the advantages of both circuit and packet switching.

## 2.2 The ATM Protocol

ATM comprises the ATM layer and the ATM Adaptation layer (AAL), see Figure 2.4.

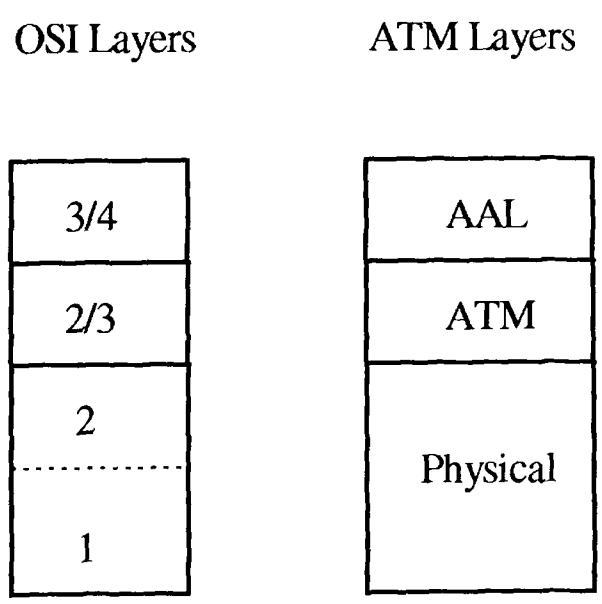



**Figure 2.1 OSI Layers and Corresponding ATM Layers**

The ATM reference model corresponds to the bottom three layers of the OSI model [TAN96]. However, it does not map directly onto the OSI model, as the functions of some of the ATM layers fall into two layers in the OSI model, see Figure 2.1.

### 2.2.1 The ATM Cell Format

ATM uses fixed size packets, called cells. A cell consists of a 48 octet payload and a 5 octet header, as shown in Figure 2.2. The choice of cell size is intended to prevent excessive waste of bandwidth due to partially filled cells. The high speed switching which is associated with ATM networks is only possible with fixed size cells, as they can be switched more efficiently, due to reduced complexity of the hardware required. The use of a small size cell also reduces queuing delays by keeping the processing time at switches and multiplexers to a minimum. This is an important consideration when transporting RT traffic, e.g. video, which outputs a nearly constant stream of data. A fixed cell size also helps to reduce long packetisation delays for low bandwidth services, such as voice, which is discussed more fully in Section 2.2.1.1.

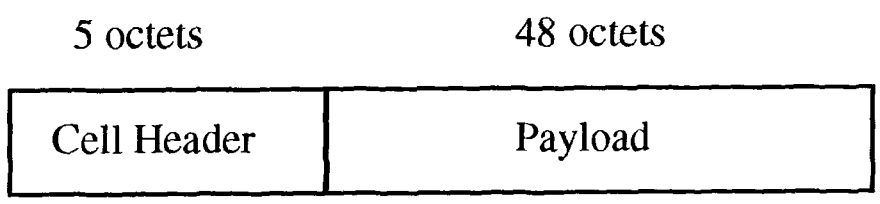


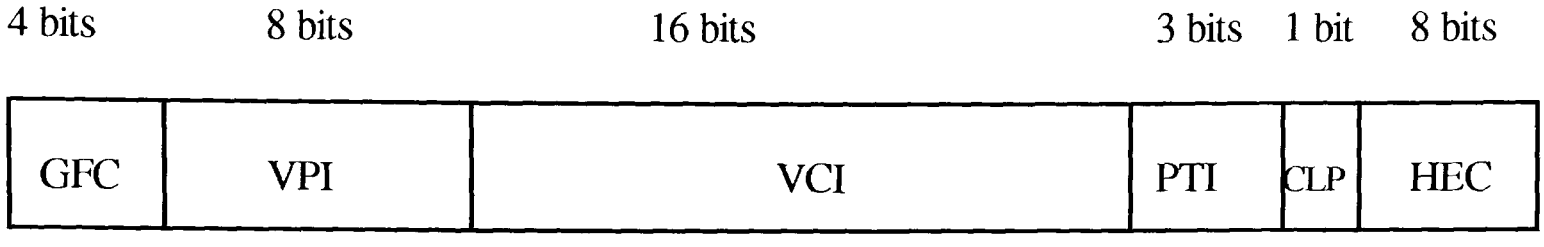

**Figure 2.2 ATM Cell Format**

#### 2.2.1.1 ATM Cell Payload

The size of the payload relative to the cell header is also an important consideration. A longer information field means that the cell makes more efficient use of bandwidth for the same size header. This can also cause problems. For example, increasing the overall size of the cell to 64 octets, increases the associated packetisation delay. This can be a problem for speech cells as longer delays mean that echo cancellors are needed for speech connections. One solution is to only partially fill speech cells which reduces the packetisation delay, so that echo cancellors are not needed. Alternatively, decreasing the cell size to 32 octets also removes the need for echo cancellors but the efficiency of the cell is reduced.

### 2.2.1.2 ATM Cell Header

The ATM cell header format is shown in Figure 2.3 (a) and (b). Each of the fields is discussed below. An explanation of virtual channel and virtual path connections can be found in Section 2.2.4.




**(a) ATM Cell Header at the User to Network Interface (UNI)**

Key

| | | | |
|---|---|---|---|
| GFC | Generic Flow Control | VPI | Virtual Path Identifier |
| PTI | Payload Type | VCI | Virtual Channel Identifier |
| HEC | Header Error Check | CLP | Cell Loss Priority |


**(b) ATM Cell Header at the Network to Network Interface (NNI)**

**Figure 2.3 ATM Cell Header at (a) UNI and (b) NNI**

### Generic Flow Control

The generic flow control (GFC) field is only used at the user-to-network interface (UNI). It is not used within the network, as it becomes part of the virtual path identifier field. The GFC field can be used by users to control the flow of cells at the local UNI and is not part of the ATM standard.

The first bit of the GFC has been used to provide a simple stop/go flow control protocol across the UNI, [ARN95], for all traffic. The traffic is divided into two classes, with constant bit rate (CBR) , which is mainly speech, assigned to one queue and all other traffic assigned to a "bursty" queue. The traffic in the "bursty" queue is subjected to traffic conditioning or shaping, to smooth it, and has a rate control mechanism to restrict the output. The second and third bits were assigned as reset indicators for the two separate credit-controlled queues in the subscriber premises equipment.

The idea of using the GFC field to flow control traffic sources has now been largely abandoned according to [JAIN96].

**Virtual Channel Identifier**

The virtual channel identifier (VCI) is the address label used to route the cells of a connection to the destination. The VCI is assigned on a hop by hop basis, and is changed as the cell passes through a switch. Each VCI is only valid for a single link between two ATM nodes (see Figure 2.7).

**Virtual Path Identifier**

The virtual path identifier (VPI) is used for routing bundles of virtual channel connections in a virtual path. It has the effect of creating a semi-permanent connection between end points. When a connection is assigned to a virtual path, the VCI does not change as the cell passes through the switches. Switching in the ATM nodes is done using only the VPI field in the cell.

**Payload Type Identifier**

The payload type identifier (PTI) field is used to define the type of cell, for example, user data, signalling, etc. and to allow some congestion control information to be passed across the network.

The first bit of the PTI field indicates that the cell contains user information (set to 0) or network management information (set to 1). If the first bit indicates user information, then the second bit indicates if congestion has occurred and the third bit is for end user information.

The undefined 111 code of the PTI field may be used to send a reset request to any network component.

### Cell Loss Priority

The cell loss priority (CLP) bit is used to designate cells of low priority, which may be dropped in case of network congestion. This may happen anywhere in the network.

It was proposed [PLATT94] that the CLP bit should be redefined to be a "recovery needed" bit for an enhanced "go back N" scheme for data traffic within the network.

In the basic loss priority scheme, low priority cells are accepted if the buffer queue length is below a threshold. This effectively divides the buffer into two areas; below the threshold is shared by all cells and above the threshold is dedicated to high priority cells. The area above the threshold does not need to be large for the scheme to be effective.

## 2.2.2 ATM Adaptation Layer (AAL)

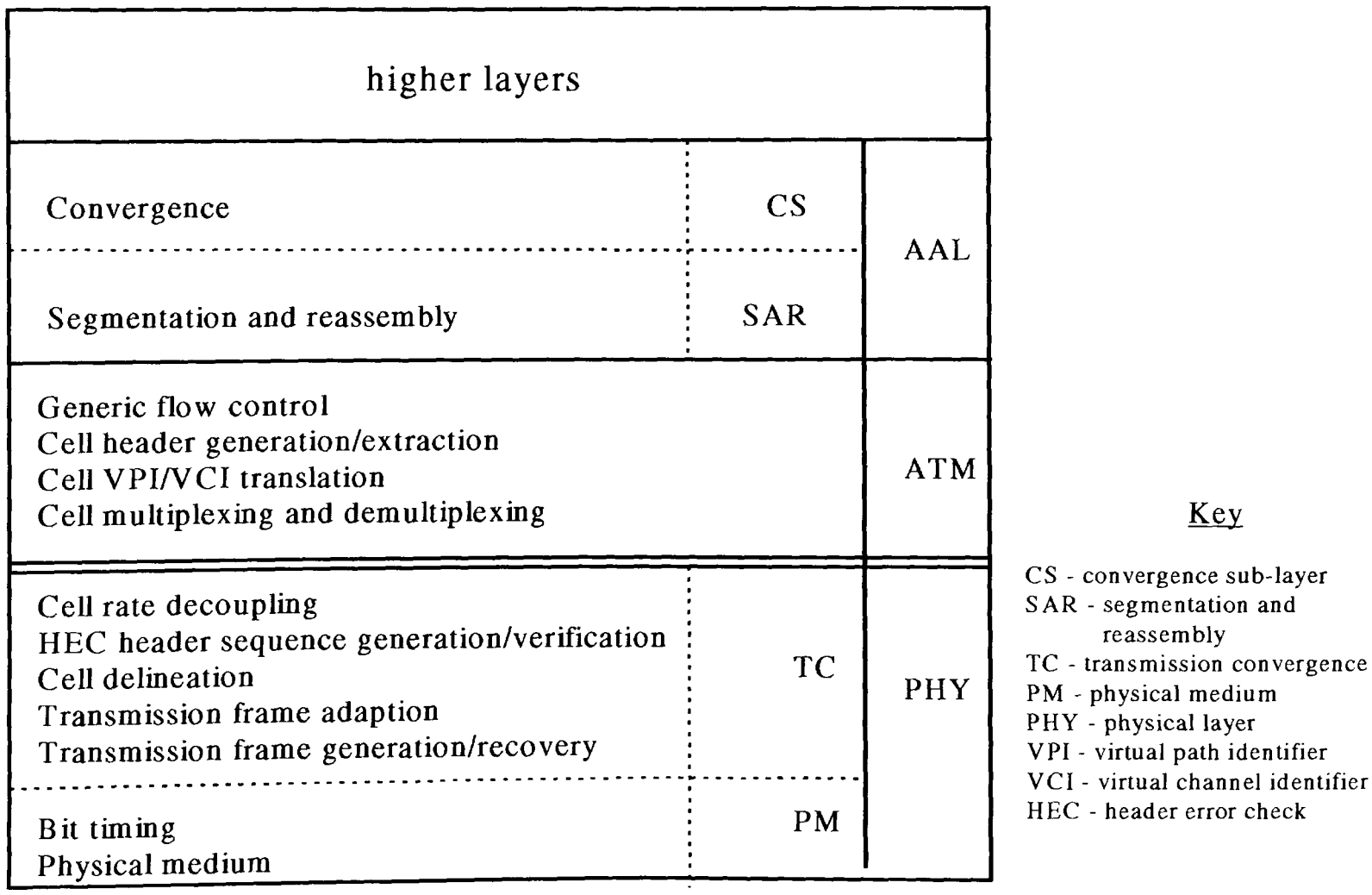
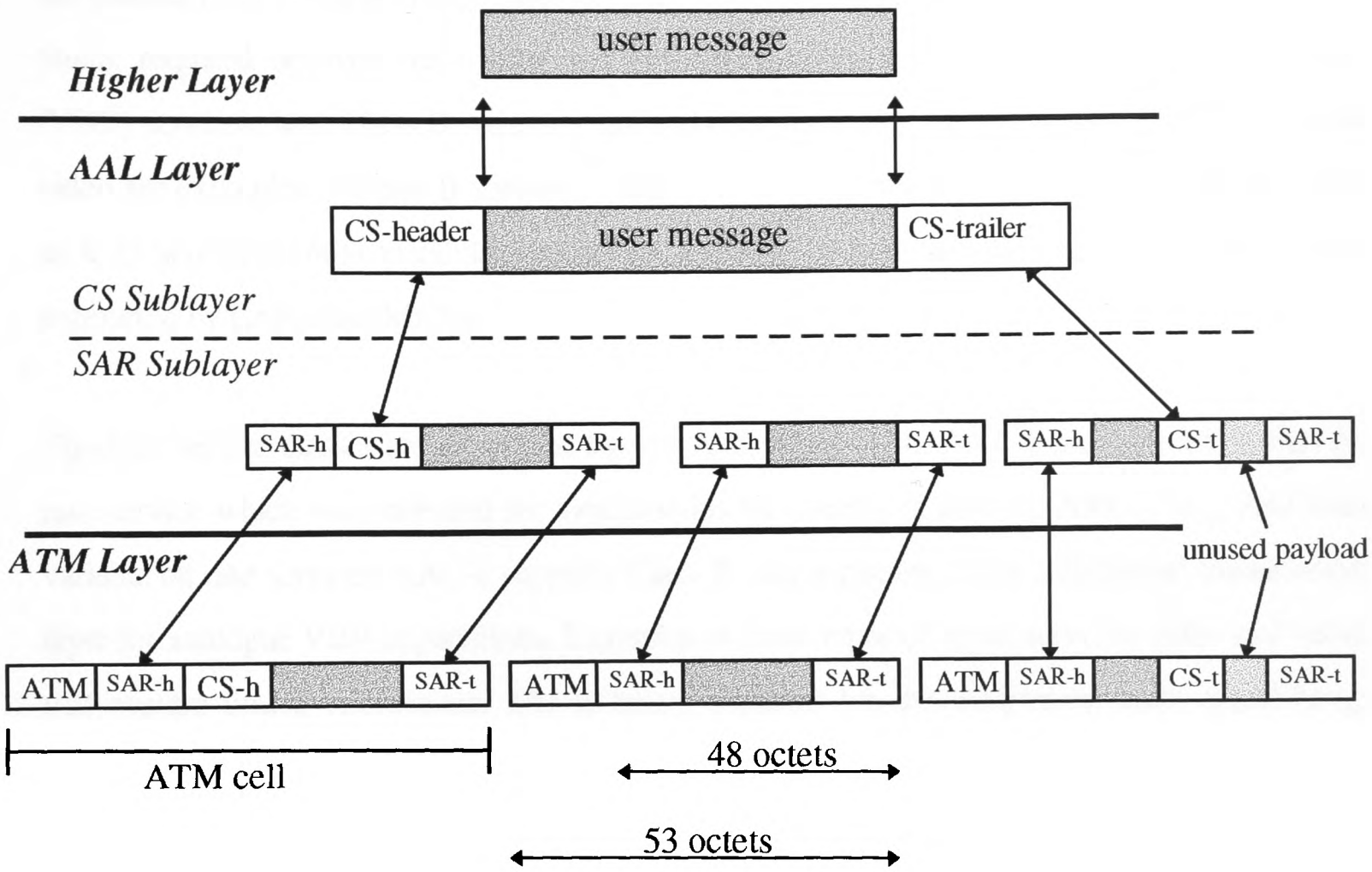



**Figure 2.4 ATM Protocol Reference Model - Sub-layers and functions**

The AAL performs an adaptation function by mapping the user's message or protocol data unit (PDU) in to the information fields of the ATM cells, see Figure 2.5. Cells received from the ATM layer are re-assembled into messages for the higher layers. For variable bit rate services the PDU may not be a multiple of the cell payload, which can result in partially filled cells.




Key

CS Sublayer = Convergence Sublayer
SAR Sublayer = Segmentation and Reassembly Sublayer
ATM = ATM header (5 octets)
SAR-h = SAR header
SAR-t = SAR trailer
CS-h = CS header
CS-t = CS trailer

**Figure 2.5 AAL Segmentation and Re-assembly in CS and SAR Sublayers**

The AAL is sub-divided into two layers, the Convergence sub-layer (CS) and the Segmentation and Re-assembly (SAR) sub-layer. The CS sub-layer performs multiplexing, cell loss detection,

timing recovery and provides the interface to the applications. The SAR sub-layer segments data units into cells for transmission, and reassembles the cell payload delivered by the ATM layer, into a byte stream for the upper layers.

The ITU-T is currently defining a number of AAL specific service classes corresponding to different services, such as real time video. The service access points (SAP) 1 to 4 correspond to the Classes A to D respectively. Class A corresponds to a constant bit rate (CBR) service, with timing required between the source and destination. Class B and C are for variable bit rate (VBR) services, with Class B requiring timing between source and destination. VBR audio and video are examples of Class B services. Class C includes connection oriented data transfer such as X.25 and future high speed data services. Class D is for connectionless services, such as those supported by LANs and MANs.

The four service classes are supported by five AAL types (1-5). AAL-1 provides a constant bit rate service which was intended for synchronous bit streams (Class A). AAL-2 to 5 deal with variable bit rate services. AAL-2 supports Class B, and is planned to be a dedicated transmission layer for analogue VBR applications. Examples of these types of application are video and audio that require timing information. The technical standard for this adaptation layer is still being drafted. AAL-3 (Class C) and AAL-4 (Class D) were combined after it was realised that the initial specifications where very similar. It is now known as AAL-3/4 and can provide connectionless and connection-oriented services. AAL-5 is similar to AAL-3/4 and provides a connectionless VBR service.

Each of the service classes has its own CS protocol which is associated with a SAP. The SAP is used to direct the service data unit (SDU) which is submitted for transfer [HAL96]. There are four types of SAR protocols, each with their own PDU structure, see Figure 2.6.

| SAR PDU | SAR Header | Payload | SAR Trailer |
|---|---|---|---|
| AAL-1 | 1 octet | 47 octets | |
| AAL-2 | 1 octet | 45 octets | 2 octets |
| AAL-3/4 | 2 octet | 44 octets | 2 octets |
| AAL-5 | no header | 48 octets | |

**a) SAR Protocol Data Unit Structure**

| CS PDU | CS Header | Payload | CS Trailer |
|---|---|---|---|
| AAL-3/4 | 4 octet | 1 - 65,535 octets | 4 - 8 octets* |
| AAL-5 | no header | 1 - 65,535 octets + | 8 octets |

* includes 0 - 3 octet Pad
+ includes 0 - 47 octet Pad

**b) CS PDU Structure**

**Figure 2.6 SAR and CS PDU Structures**

An AAL-3/4 protocol data unit (PDU) is received from the higher layers and broken up into payload blocks each 44 octets in length. This means that 4 octets per ATM cell is used for SDU information. It was thought that there was too much complexity and overhead per cell with AAL-1 to 3/4, so a new protocol was devised called SEAL - simple efficient adaption layer [TAN96]. This was eventually adopted by the ATM Forum and became AAL-5, to provide a streamlined, connection orientated transport facility for higher-layer protocols. Connection management is the responsibility of the higher layers. The idea was to reduce processing overhead and to ensure adaptability to existing transport protocols. A message of length 1 up to 65,535 bytes may be passed to the AAL layer, and the full 48 octet payload is available for user data, as the AAL-5 does not have a convergence sub-layer header in each cell, only an 8-octet trailer per message [CLARK96].

### 2.2.3 The ATM Layer

The ATM layer performs relaying functions, with every cell carrying a label which is used for switching. The cells are switched in the network based on the routing information in their

headers. This differs from the N-ISDN User-Network Interface, where several digital channels are offered, that are circuit switched in the network.

The ATM layer assembles the cell header, as shown in Section 2.2.1.2. Transporting an ATM cell requires a connection to be set up, either dynamically or at subscription time. The routing information in the header is not an explicit address, but a label used to switch the cell through the network This consists of a virtual path identifier (VPI) concatenated with a virtual channel identifier (VCI). A multiplexer or switch reads the cell's header, as it arrives at a particular input port. The routing table is used to determine the correct output port and a new label for the cell. The new label is used by the next switching node. The ATM layer uses the services of the Physical layer to transport cells and delivers the cell payload to the upper layers.

### 2.2.4 Physical Layer (PL)

The layer below the ATM layer is the physical layer (PL), which transports valid cells and delivers timing information when required by the upper layer services, if circuit emulation is used. The PL is divided into the Physical Medium (PM) sub-layer and the Transmission Convergence (TC) sub-layer.

The PM sub-layer is responsible for the correct transmission and receipt of bits on the appropriate physical medium. This is dependent on the medium used, but the function of bit timing is common to all. It also includes the insertion and extraction of symbol timing information and electrical-to-optical and optical-to-electrical transformations as required [ONV94].

The TC sub-layer is responsible for generating and maintaining the frame structure of the physical layer. This may mean inserting or suppressing idle cells in the PM sub-layer. This layer also handles header error control (HEC) generation and verification, frame and cell delineation and line coding. The TC transmits cells as a stream of bits to the PM sub-layer. At the receiving end, a stream of bits is received, which must be converted back into cells for the ATM layer.

Two options are specified to transport cells. One option is to use a continuous stream of cells, with no multiplex frame structure imposed and synchronise on a cell by cell basis (ATM). The

other option is to place the cells in a synchronous time-division multiplex (TDM) envelope (SONET and SDH).

According to [ONV94], the original ATM standard had the primary rate of 155.52 Mb/s, which was chosen to be compatible with SONET. SONET (Synchronous Optical NETwork) was introduced when optical fiber networks began replacing the standard analogue telephone trunk lines. Later SDH (Synchronous Digital Hierarchy) was introduced, which differs from SONET only in minor ways. However, nearly all long distance telephone traffic, in the USA uses SONET in the physical layer. It was decided that SONET would be a traditional TDM system, with a single channel containing time slots for the sub-channels. SONET is a synchronous system, controlled by a master clock with an accuracy of 1 part in $10^9$.

There are two levels of ATM connections defined by ITU-T, and these are virtual channels and virtual paths.

**Virtual Channel Connections**

A virtual channel connection (VCC) is the basic type of end-to-end connection. It is not only a logical connection used for routing, but also has traffic usage parameters and quality of service objectives associated with it. Cell sequence integrity is maintained within each VCC. Both the VCI and VPI fields of the cell header are examined for routing purposes.

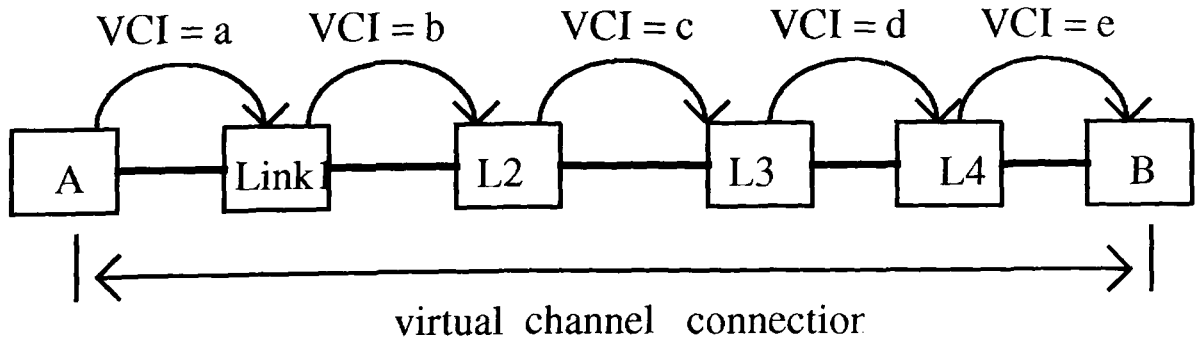



**Figure 2.7 A Virtual Channel Connection, showing VCI labels per link**

At each node in the network, the VCI label in the cell header is changed, in accordance with the routing table entry, as indicated in Figure 2.7. This allows the cell to have a reduced size address field, since the VCI is not a full network address, but only a label used for routing purposes.

### Virtual Path Connections

A virtual path connection is used to transport bundles of virtual channel connections, which are then switched together as a single unit. A virtual path connection is defined in a similar way to a virtual channel connection, except the VPI field in the cell header is used for routing. At a virtual path switch all virtual channel connections with the same VPI are switched together. The VCI field remains unchanged through the virtual path connection, see Figure 2.8.

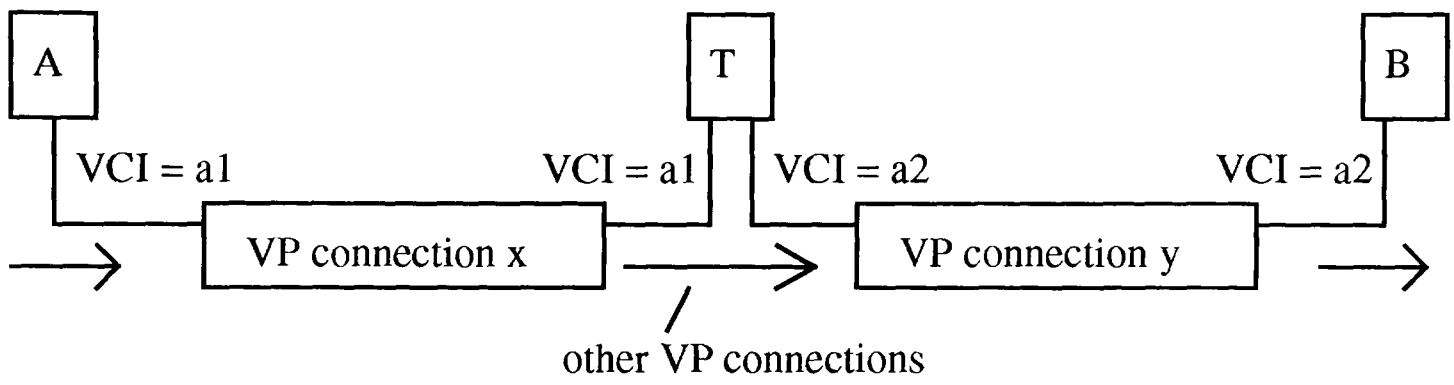



**Figure 2.8 Virtual Path Connection**

A two-level hierarchy is achieved by splitting the routing field in the cell header into the VCI and VPI fields.

### 2.2.5 Performance Issues

At connection set up, a number of parameters relating to the required type of service, bit rate, delay, delay variation, cells error/loss probability are negotiated. The quality of service (QoS) required varies according to the type of traffic that is being transported, ranging from video and data, to multimedia applications. For real time sources, such as speech and video, the overall

delays, cell loss probability and cell delay variation are the main QoS issues, while for data traffic the main QoS issue is low cell loss probability. For data this means error free delivery, while for video this is error free delivery within a specific time frame.

The following terms have been used to define QoS:-

**Peak cell rate** - is defined as the number of cells generated when the source is in the active state. For CBR cells this is the same as the mean rate. The ITU-T also refers to the "Instantaneous Peak Cell Rate" as the reciprocal of the minimum inter-arrival time between two consecutive cells belonging to the same connection [STAM94].

**Mean cell rate** - is the average number of cells generated over a measured time period. The true average cell rate should be calculated over the course of a connection:-

$$\text{Mean cell rate} = \frac{\text{Total number of cells generated}}{\text{duration of connection}} \quad \text{(cells per second)}$$

**Estimated mean cell rate** - is the number of cells generated during a time period $T$, divided by $T$. If the connection time is long, then this may not necessarily be the true mean cell rate, and is only an estimate over a short time period.

$$\text{Estimated mean cell rate} = \frac{\text{No. cells generated in } T}{T} \quad \text{(cells per second)}$$

**Burstiness** - is defined as the ratio of the peak rate to the mean rate:-

$$\text{Burstiness} = \frac{\text{peak rate}}{\text{mean rate}} \quad \text{(no units)}$$

### 2.2.5.1 Delay

Delay issues are mainly applicable to real time services e.g. speech and video. The typical delays that are found in ATM networks are shown in Figure 2.9.

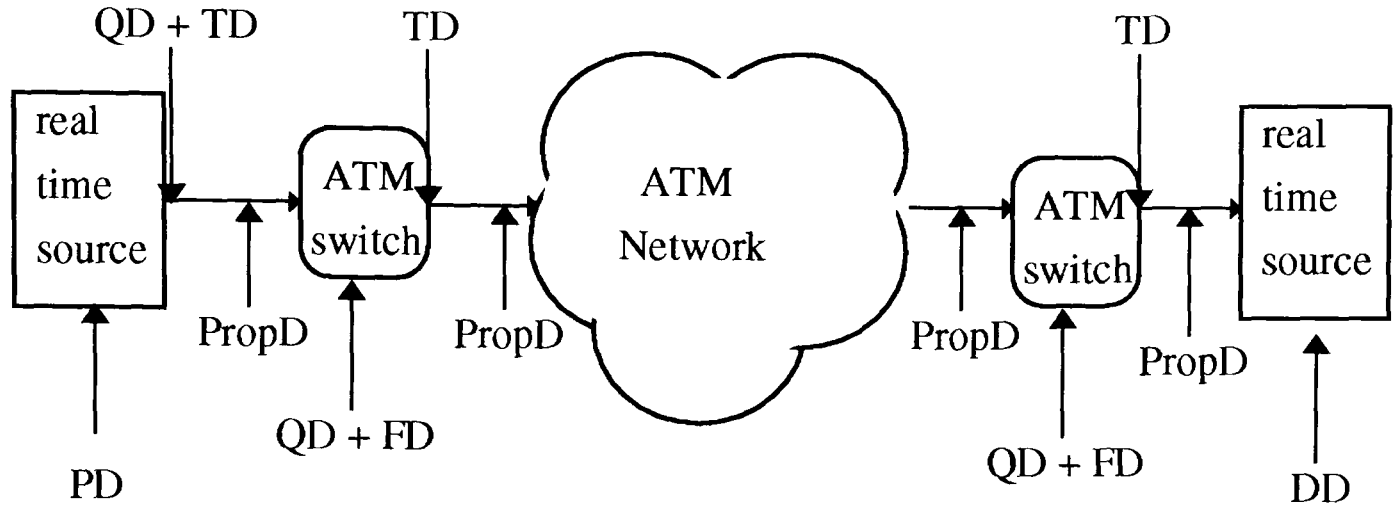



**Figure 2.9 Delay Characteristics of an ATM Network**

**Transmission delay** (TD)

The transmission delay is the time it takes to place a cell on to the transmission medium (also called the service time) and is defined by the formula

$$\text{transmission delay} = \frac{\text{cell size (bits)}}{\text{link speed (bits/s)}}$$

**Propagation delay** (PropD)

The propagation delay is dependent on the distance and independent of the ATM concept. Depending on the transmission medium, this is typically 4 to 5 microseconds per kilometre. For example, if the distance between source and destination is assumed to be 1,000 Km, this gives a propagation delay of 4,000 microseconds.

**Packetisation delay** (PD)

A packetisation delay occurs each time a real time service e.g. speech and video, is converted into cells. Also occurs at boundaries between ATM and non-ATM networks. The packetisation delay (PD) is dependant on the cell length and on the speed with which the source generates bits.

**Switching Delay** - the switch delay comprises a fixed part (FD) and a variable part determined by the queue (QD).

**FD** - The fixed switching delay is the delay imposed by the switch on each cell as it is passed through the switching fabric to the output port. It is dependent on the implementation, but is in the order of tens of cells per switch exchange. For small sized cells and high link speeds, this results in a value between 2 and 32 microseconds per exchange (16 and 256 microseconds for eight consecutive exchanges).

**QD** - ATM switches are routing many ATM cells at any given moment and queues are required to avoid excessive loss of cells, when contention between cells inevitably arises. Delays are invariably introduced by queuing and this will vary, depending on the load of the links inside the network. The behaviour of a particular queue is characterised by a probability density function (pdf) of the queue length describing the statistical behaviour of the queues.

**Depacketisation Delay** (DD)

A depacketisation delay may be added to smooth out the stochastic delay (i.e. delay jitter) introduced by the network. This is particularly relevant for real time traffic, when excessive delay jitter can cause a loss of synchronisation within the bit stream. This does however, represent an additional delay at the receiver. If the cell delay in the network is longer than the depacketisation delay, then the cell will arrive too late and will be lost. The additional delay is determined by the sum of the queuing delays. The total jitter to be removed ranges from 50 to 800 microseconds depending on speed and cell size.

The delay indicated is actually the worst case (i.e. maximum) delay through all queues in the network. Therefore, the depacketisation delay will not appear in the formula for the total delay. These delay parameters can then be combined to calculate the total end-to-end delay.

$$ATM(D) = \sum_i TD_i + \sum_i PropD_i + \sum_j FD_j + \sum_j QD_j + PD$$

where

i = transmission link
j = ATM switch

### 2.2.5.2 Cell Delay Variation

Cells enter the network at regular intervals, especially in the case of a CBR source such as speech. As the individual cells pass through multiplexers and switches, the inter-cell spacing changes, due to some cells being delayed in queues longer than others. This is known as cell delay variation (CDV), or jitter, which can cause problems for decoders. Multiplexers can also cause clumping or dispersion of cells [JAD95] which affects the CDV. Real time cells may arrive too late to be included in the decoding process, [KEY94]. CDV also causes a major problem for clock recovery and synchronisation. Synchronisation is based on reproducing the encoder's system clock and using it to present video and audio data timed according to it. For an MPEG video stream, this manifests itself at the decoder as either picture break-up and audible gaps or may even cause loss of lip synchronisation which is noticeable to the viewer. A typical MPEG decoder can tolerate only one to four milliseconds of jitter before these types of problems are introduced [RUIU96a].

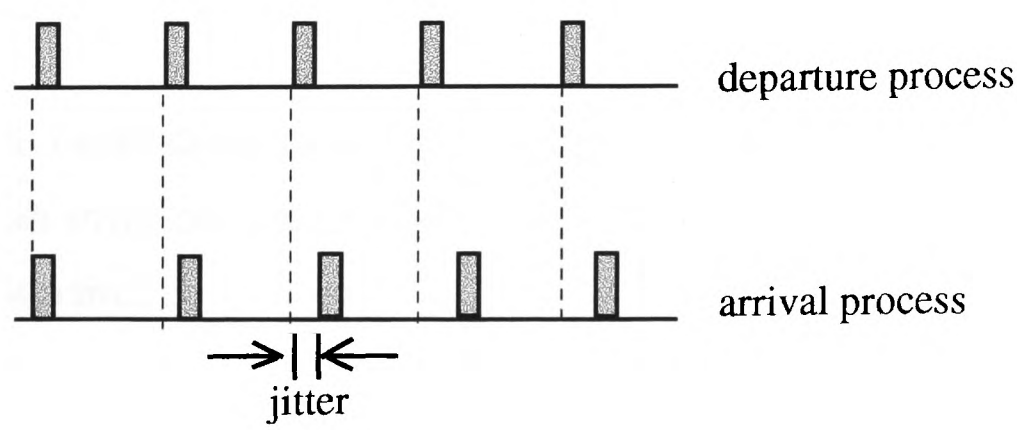



**Figure 2.10 A CBR Source Exhibiting CDV**

Figure 2.10 from [MIT94] shows the effect of CDV on a stream of CBR ATM cells as they pass through multiplexers and switches.

## 2.3. Traffic Models

At this time, there are very few ATM networks actually in operation, so it is not possible to collect statistical data from a fully functioning network about the types of traffic using the network and behaviour patterns. It is often difficult to predict the exact nature of the traffic mix that will be found on ATM networks, although it is expected that real time video traffic will dominate. Hence, to evaluate performance accurately, some form of approximation to the actual traffic must be made, and the modelling of traffic sources is the focus of much research. There are many statistical and stochastic traffic models which have been developed for convenient mathematical solutions to queuing theory problems. They are often an over simplification which allow elegant mathematical solutions to complex problems.

By using simulation, the more complex behaviour of the different traffic types can be explicitly modelled. This enables a true insight into the effects of different sources of traffic through multiplexers and switches. It also provides a vehicle to model the effect different traffic types may have upon each other, in terms of end-to-end delays and losses due to buffer over-flows can be investigated. This is exactly the type of behaviour which is difficult to model mathematically.

Multimedia is characterised by continuous traffic [KARA95] such as speech, video, high quality audio and graphical animation, which place greater demands on the network than traditional traffic types, such as file transfer.

Ideally, any traffic model should be described using a limited number of parameters. These tend to be the mean cell arrival rate, the peak arrival rate, the burstiness of the source and the average time the source is active.

### 2.3.1 General Traffic Models

In earlier work on traffic modelling it was often assumed that cell arrival rates fit a Poisson process. However, the type of traffic that requires a bandwidth enforcement strategy, can not be

modelled simply as a Poisson process [KIM92]. [PAX95] agrees that this is an over simplification, which can seriously under estimate the burstiness of, for example, FTP and TELNET traffic, as sampled on a WAN. The statistical behaviour of bursty traffic is far from the Poisson or deterministic model used [GALL89] and it makes analysis very difficult. There is only limited burstiness found in Poisson arrival processes, particularly when such traffic streams are multiplexed together. This can be a problem when testing protocol issues or hardware designs as an under-estimation of the burstiness of the traffic can lead to errors in predicted buffer requirements and delays, particularly for real time cells.

Real network traffic is much burstier than the Poisson and Bernoulli models which are often used to simulate network traffic [XING95]. These models can be quite limited in their burstiness, especially if they are multiplexed together. Poisson arrival processes are used for two classes of traffic, [HART91], which are both independently distributed. The first class of traffic is loss sensitive and defined as data and signalling services, while the second class is loss tolerable and includes speech, image and video services. The low priority cells from this class can be discarded without significant impairment to the services.

A popular model for generalised traffic is a two-state Markov Modulated Deterministic Process (MMDP), which is characterised by an active (ON) state and a silent (OFF) state. An attempt is made to define burstiness [SOLE94] using this traffic model and a burst stream approach is used for traffic modelling. In the work done on the interdeparture processes from an ATM network, [WANG93], real time traffic is modelled as N identical streams, which are modelled as Switched Determined Processes (SDP), and non-real time traffic is modelled as aggregated to form a Bernoulli process with batch arrivals. A Markov Modulated Poisson Process (MMPP) model with parameters that successfully approximate the average delay does not work well when used to analyse cell loss [COSM94].

A two state Markovian representation [ORS95] is used to model a traffic source. The duration of each burst is exponentially distributed with mean 1/a ms. ATM cells are emitted with a constant inter-arrival time T ms, where T = 1/peak bit rate. The silence period is also exponentially distributed, with a mean of 1/b ms. The parameters a and b, are varied to represent either packetised speech, still picture or an interactive data service.

Traffic is often characterised by three descriptors, i.e. the peak rate (R), the mean rate (m) and the burst length (b) [TED93]. The triple (R, m, b) is then used to map a complex user onto a simplified exponential ON/OFF model by interpreting the parameter b, which defines the average amount of data (in bits) generated during a burst period. The average length of the idle period is adjusted to maintain the defined mean rate (m).

Traffic at a multiplexer is often characterised using generalised traffic models (2-state Markov chain) and differentiating between two categories of traffic by varying the mean active and silence periods, and in some cases the peak rate [CALL92, BONO92]. Most previous research on priority packet discarding is limited to Poisson/Bernoulli arrival assumptions [KIM96a]. They use an 8-state MMPP model as the input to a multiplexer with a threshold-based priority scheme. The performance of an ATM multiplexer is studied using a discrete time Markov chain to model two groups of bursty sources [CALL92]. The difference between the two groups is achieved by varying the mean ON / OFF times, and giving each group a different peak rate.

Testing multiplexer scheduling policies is often done using these generalised traffic sources. The loss probabilities for two Markov Modulated Arrival streams are used [KIM96a] to investigate a queuing system that multiplexes heterogeneous traffic streams onto a single finite buffer. The differences between the two streams are introduced by varying the appropriate parameters. A two-state Markov chain was used [BONO93] to represent ON / OFF sources and to investigate the feasibility of using statistical multiplexing for widely different types of traffic.

An aggregate arrival process is also often used to test the performance of multiplexers. A 4-state MMPP as the input to a statistical multiplexer [YEGE94]. The parameters for the MMPP model are derived from an actual arrival process. A superposition of several Switched Determined Process (SDP) is used and modelled as a batch arrival process, which was the input traffic to an ATM multiplexer [WAN92].

Real time and non-real time traffic are modelled as arrivals [CHANG94], in units of a message, in a frame for each traffic type. This constitutes two batch Poisson processes with different mean rates, which are mutually independent. Each message carries a random number of fixed-length cells with an arbitrarily distributed probability distribution and having a maximum size.

The characteristics of RT and non-RT traffic are significantly different, so a generalised traffic model for all traffic types will not fully capture the fundamental differences between them. The performance of multiplexers and switches is dependant on the nature of the offered traffic and while the exact make up of this traffic is unknown, a reasonable approximation can be made by including a diverse mix of traffic sources. The conclusion drawn by many researchers is that the commonly used simple ON / OFF models used to describe ATM traffic streams are not generally appropriate characterisations of the output traffic behaviour of the multiplexer.

### 2.3.2 Speech Traffic

Speech is the most widely understood traffic source, simply because it has been in existence for such a long time. Uncompressed digitised speech is coded at 64 Kb/s for transmission across the telephone network.

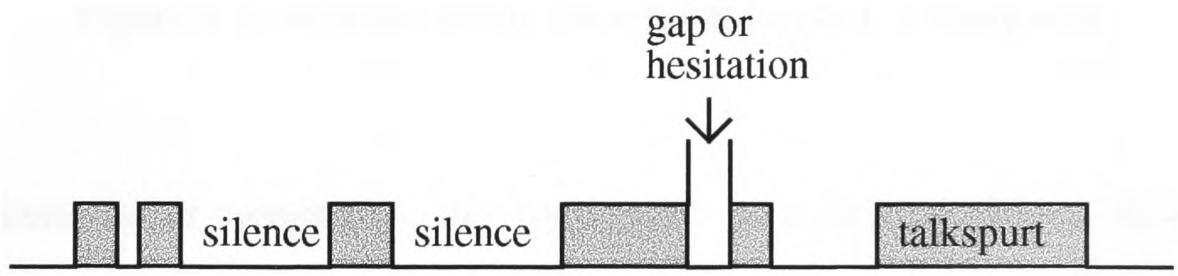



**Figure 2.11 Talkspurts and Silence Periods for Speech**

Speech is made up of talkspurts and silences, also known as On and Off periods respectively, see Figure 2.11. The determination of the average length of both a talkspurt and a silence period varies according to the interpretation of what constitutes a talkspurt. Normal conversation contains many small gaps and hesitations which the speech coder may remove, using either fill-in or hang-over techniques. Fill-in removes silences in the speech samples by discarding any gaps less than the fill-in value. Here longer silence periods and shorter talkspurts are generated. The problem with fill-in is that a delay is imposed on the talkspurt, which is equal to the value of the fill-in, hence hang-over may be the preferred technique. The purpose of hang-over is to bridge short gaps in speech and create fewer, longer talkspurts, see Figure 2.12. This does mean that

the short gaps and hesitations found in normal conversation are included in the coded speech, but this has been found to reduce the impact of cell losses on the perceived speech [GRUB82]. The potential disadvantage is that speech activity is increased using hang-over, as can be seen in Figure 2.12.

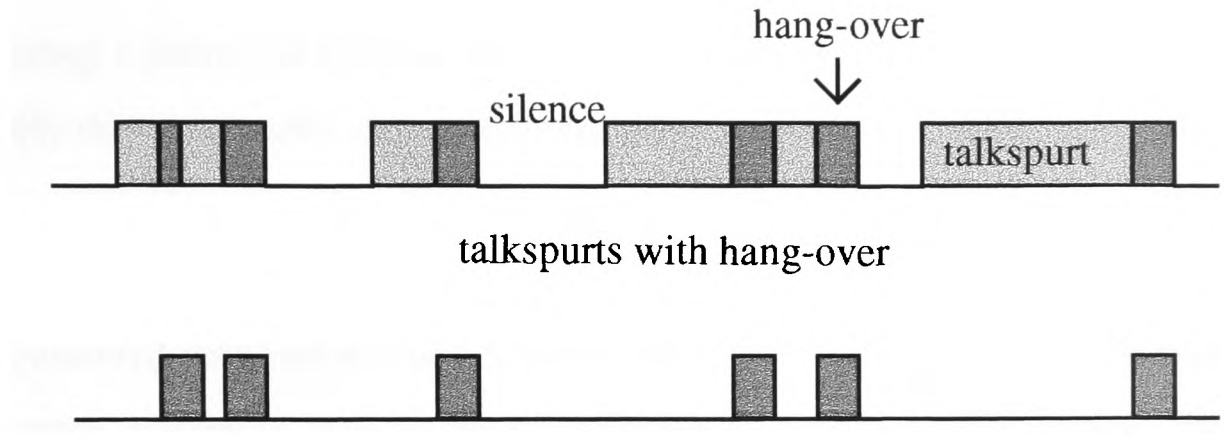



**Figure 2.12 Speech Coding Showing the Effect of Hang-over**

Delay requirements for speech cells vary considerably with different authors. However, most agree that the range of maximum tolerable delay for speech cells is between 100 ms to 400 ms, with 500 ms not being acceptable, [TURN86, SUDA89 and KEY94]. The round-trip delays of greater than 650 ms, common on satellite links, are very noticeable, and annoying, to the users. However, the end-to-end delay limit recommended by ITU-T, for conversation without echo cancellers is 25 ms.

### 2.3.2.1 Characteristics and Coding of Speech

The work on the statistical analysis of speech patterns which was done in the late 1960's, at Bell Laboratories, [BRAD65, BRAD68, BRAD69], where actual telephone conversations were analysed to determine the properties of the speech patterns present on telephone circuits. From analysis of the dynamics of two way, interactive conversations, a more sophisticated six state model was proposed [BRAD69], where it was proposed that a speaker can be in one of six states during a two-way conversation and these were :-

solitary talkspurt (only one party speaking);
double talkspurt (both parties speaking simultaneously) and the original speaker falls silent;
double talkspurt with the original speaker doing the interrupting;
mutual silence;
mutual silence with the called party interrupting;
mutual silence with the original party interrupting

Initially, during a phone call there is a brief interchange between both parties and then the calling party usually dominates, with short occasional interjections by the called party. This dominance may or may not alternate. The dynamics of two-way conversation is quite difficult to model accurately [GRUB82]. Brady proposed that the duration of each of the six states he identified had an exponential distribution. Further work, [BRAD69], indicated that the exponential model for generating talkspurts was a good approximation to the distributions found in real conversations.

Two methods are used for the coding of speech for digital transmission.

**Pulse Code Modulation**

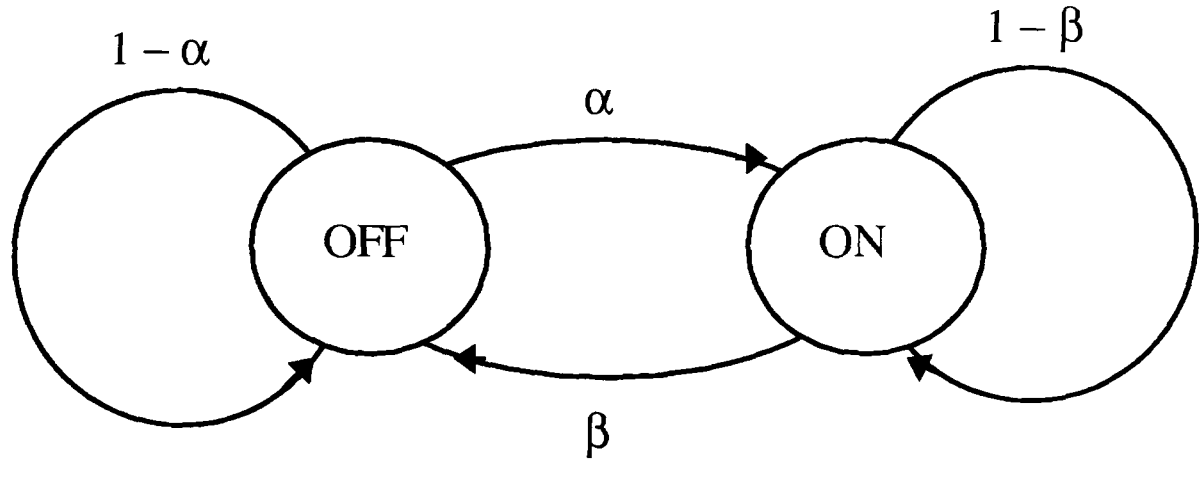
Pulse code modulation (PCM) is a technique for converting analogue speech, into a digital form for transmission across a network. Coded pulse sequences are generated by sampling the signal at time intervals. The sampled signal is then quantized and coded using an analogue to digital converter. At the receiver the original audio input is easily recovered, after conversion back to an analogue signal, using a low-pass filter. PCM is widely used in telephony mainly because the signal generated has a greater noise immunity than other coding methods, which is an advantage when using noisy telephone networks.

**Frequency domain coders**

Frequency domain coders divide speech samples into separate frequency components. Each is then coded separately. The main advantage of this is that the coder can allocate bits dynamically to frequency components where they are most needed [SING90]. The most significant bits are placed in a high priority cell and the least significant bits in a low priority cell. Coding speech in this way allows the low priority cells to be dropped at times of congestion [GERZ91]. Up to 10% of the low priority cells coded in this way can be lost with negligible degradation of the perceived speech [SUDA89, YIN90, ABBAS92], compared to only 0.1% to 2% for ordinary packet speech cells with no priority. Speech coded using these methods is generally more robust and has greater tolerance to losses of low priority cells.

### 2.3.2.2 Speech Models

The most commonly used traffic model for speech is the ON-OFF model, shown in Figure 2.13, [BAE91, GANO95]. Here active periods alternate with silence periods for the duration of the connection. The traffic is modelled as a stream of cells generated during the ON period, with no cells generated during the silence period. These time periods are almost always modelled as either an exponential or a geometric distribution.




transition from OFF to ON occurs with probability α
transition from ON to OFF occurs with probability β

**Figure 2.13 Generalised ON/OFF Speech Model**

There have been many proposals put forward for the mean length of a talkspurt and a silence period. The mean talkspurt and silence periods used [SUDA89], were 170 ms and 410 ms, respectively, for an exponential distribution. Voice packets with delays greater than 50 ms were discarded. The mean talkspurt and silence periods used were 352 ms and 650 ms respectively [ABBAS92], again with an exponential distribution. In a study to compare the effect fill-in and hang-over on speech parameters, it was found that the mean silence period was 808.8 ms for fill-in and 606.3 ms for hang-over and the mean talkspurt duration's were 2.157 seconds and 2.36 seconds, respectively [GRUB82]. The longer talkspurts are caused by the elimination, by these techniques, of the small gaps and hesitations that occur naturally in normal conversations.

Using a coding technique that splits the speech samples into the most significant bits and the least significant bits, two speech samples are required. The most significant bits are grouped into a high priority cell and the remaining, least significant bits, are placed into a low priority cell [GERZ91]. This gives a packetisation delay that is twice the normal for speech (2 x 6 = 12 ms), but two cells are coded together, and sent out one after the other. This allows discarding of low priority cells at times of congestion, while protecting the high priority cells. Loss of the low priority cells impairs the quality of the received speech, but does not cause any perceived loss in intelligibility to the user, [SUDA89, YIN90 and ABBAS92].

### 2.3.3 Video Traffic and Coding Standards

Video is the least understood and the most unpredictable of the traffic sources. The video for transmission across ATM networks will fall mainly into two categories, namely constant bit rate (CBR) as found in video phones and variable bit rate (VBR), which is video conferencing and video-on-demand services e.g. HDTV. Both have the strict delay requirements of real time traffic. Experiments have shown that viewers do notice the loss of lip synchronisation that occurs if there a mis-match greater than 120 ms between the audio and video streams [HUAN96].

The most common standards for audio and video compression coding and multiplexing is that defined by the Moving Pictures Expert Group (MPEG). Two versions are in existence. MPEG-1 supports video coding at bit rates up to about 1.5 Mb/s giving near VHS quality. MPEG-2 was optimised for the digital compression of TV broadcast material and the modelling of video traffic in this work is based on this standard. MPEG-2 produces data at a non-constant rate. Still areas produce few bits and when there is a lot of movement within a frame the output from the video codec increases. Standard definition television produces bit rates from 4 - 9 Mb/s and high definition TV at 15 - 25 Mb/s. Compression is achieved by exploiting spatial and temporal and psycho-visual redundancy.

#### Spatial and Temporal Redundancy

Pixel values are not independent, but are correlated with their neighbours both within the current frame and across frames. Hence, the value of a pixel is predictable given the values of neighbouring pixels.

**Psycho-visual Redundancy**

The human eye has a limited response to fine spatial detail and is less sensitive to detail near edges or scene changes. Consequently, controlled impairments introduced into the decoded picture by the bit rate reduction process should not be visible to viewers.

Both these types of redundancy can be exploited to reduce the bit rate for a frame.

### 2.3.3.1 Characteristics and Coding of Video

MPEG-2 coding is based on the MPEG-1 standard for video compression. There are two key techniques are used by an MPEG codec, and these are the intra-frame Discrete Cosine Transform (DCT) and motion compensated inter-frame prediction [LODG92]. Both are old and tried bit-rate reduction techniques known prior to MPEG which exploit the spatial and temporal redundancy present within a video frame. This can be used to predict the position of a pixel in the next frame.

**Intra-frame DCT Coding**

The first step in the video coding is to convert the picture into the frequency domain using a 2-dimensional discrete cosine transform (DCT). To do this pixels are grouped together into small blocks of the image called macro-blocks. The most common block size is 8x8. A transform is performed on the blocks to produce a block of DCT coefficients. The DCT is close to the 2-dimensional Fourier transform, except that the resulting coefficients are reals. At this stage it is totally reversible, as no information has been lost and no compression has been performed. The DCT produces a series of coefficients in order of increasing frequency. For natural images, the transform tends to concentrate the energy into the low frequency coefficients and many other frequencies are near zero. This non-uniform coefficient distribution is the result of the spatial redundancy present in the original block.

The top left-hand coefficient represents the dc component of the block (actually twice the mean). The dc component is more sensitive and has a greater impact on the picture quality, if damaged. The bottom right-hand quadrant usually has hardly any significant coefficients at all. This highlights the redundancy in the data, which is typical of natural images, and which can be exploited to derive an efficient binary description of this block for transmission. Bit rate

reduction is achieved by not transmitting the near zero components (thresholding) and by quantizing the remaining ones.

In thresholding any coefficients falling below the threshold are discarded, on the assumption that they will not make any significant contribution to the image, e.g. any coefficient smaller than 2 is discarded. The coefficients are then quantized to reduce the number of possible values and to convert some of the least significant bits to zero, while still achieving excellent quality reproduction. The degree of quantization applied to each coefficient is weighted according to its visibility in the resulting image. In practice this often means that high frequency coefficients are more coarsely quantized than low frequency coefficients. However, this process is not reversible at the decoder and hence this is called a "lossy" coding technique.

The information lost in the quantization process is usually the higher frequency coefficients that tend have very small values and hence have little impact on the quality of the final picture. The dc term is represented as a fixed length binary word The remaining coefficients are then Huffman encoded using variable length codes, with the most common coefficients coded using the least number of bits. The number of zero coefficients are also coded by using a special code indicating that the following number of coefficients are all zero. Since up to 50% of the higher frequencies tend to get converted to zero, this reduces the bit rate significantly. The dc and lower frequency coefficients are the most important as they have the greatest impact on the received picture quality. The loss of some of the higher frequency coefficients may result in some loss of fine detail. Since the human eye has a limited response to fine spatial detail and is less sensitive to detail near edges or at scene changes, controlled impairments introduced into the decoded picture by the bit rate reduction process should not be visible to viewers.

**Motion Compensated Inter-frame DCT Coding**

The same method as intra-frame DCT coding is used, except the basic coding block is not made from picture samples, but instead is the error resulting from an attempt to predict the sample values in the block, from the contents of the previous frame. A block from the previous frame is used to predict the current block. This works well for static areas, but for moving areas, there is a need to offset the motion that has occurred by using a shifted block from the previous frame. This is called motion-compensated prediction. A technique called block matching is used to find

the displacement between the current block and its most appropriate match in the previous frame. A block of (2N x 2N) samples from the current frame is compared with every possible location in a larger search area of (2M x 2M) samples from the previous frame. The displacement vector for each block is conveyed to the decoder, along with the DCT-coded prediction

Another technique used is also based on motion-compensation and is called bi-directional motion-compensation. This technique tries to match blocks from both a past and future frames and then uses a mathematical method to generate the current block. This method is called interpolation.

In MPEG-2 the codec decides, on a block by block basis whether to employ motion compensated inter-frame coding or intra-frame coding and a control bit indicates to the decoder which mode to use. More often inter-frame mode is better, but where there is motion within the scene which is erratic, or an unpredictable backgrounds appears, then the intra-frame option results in a lower volume of data for the block. The intra-frame mode is deliberately chosen for a number of blocks in each field, on a periodic basis, as a means of flushing out persistent error effects, since it does not rely on reconstructed previous fields. This is called refreshing and occurs 2 to 3 times per second.

In MPEG-2 three picture types are defined, based on the mode used in each block. (A block is referred to as a picture in MPEG coding).

| | |
|---|---|
| I pictures | - "intra" coded without reference to other pictures<br>- moderate compression is achieved by reducing spatial redundancy, but not temporal redundancy<br>- provides access points in the bit stream where decoding can begin |
| P pictures | - "predicted" - using motion compensation from a past I or P picture<br>- may be used as a reference for further prediction<br>- reduced spatial and temporal redundancy<br>- offers increased compression compared to I pictures |
| B pictures | - bi-directionally predictive<br>- uses both past and future I or P pictures for motion compensation<br>- also offers the highest degree of compression |

The coded data may also include synchronisation words, mode control bits, motion vectors, buffer occupancy information, audio, teletext, text data etc.

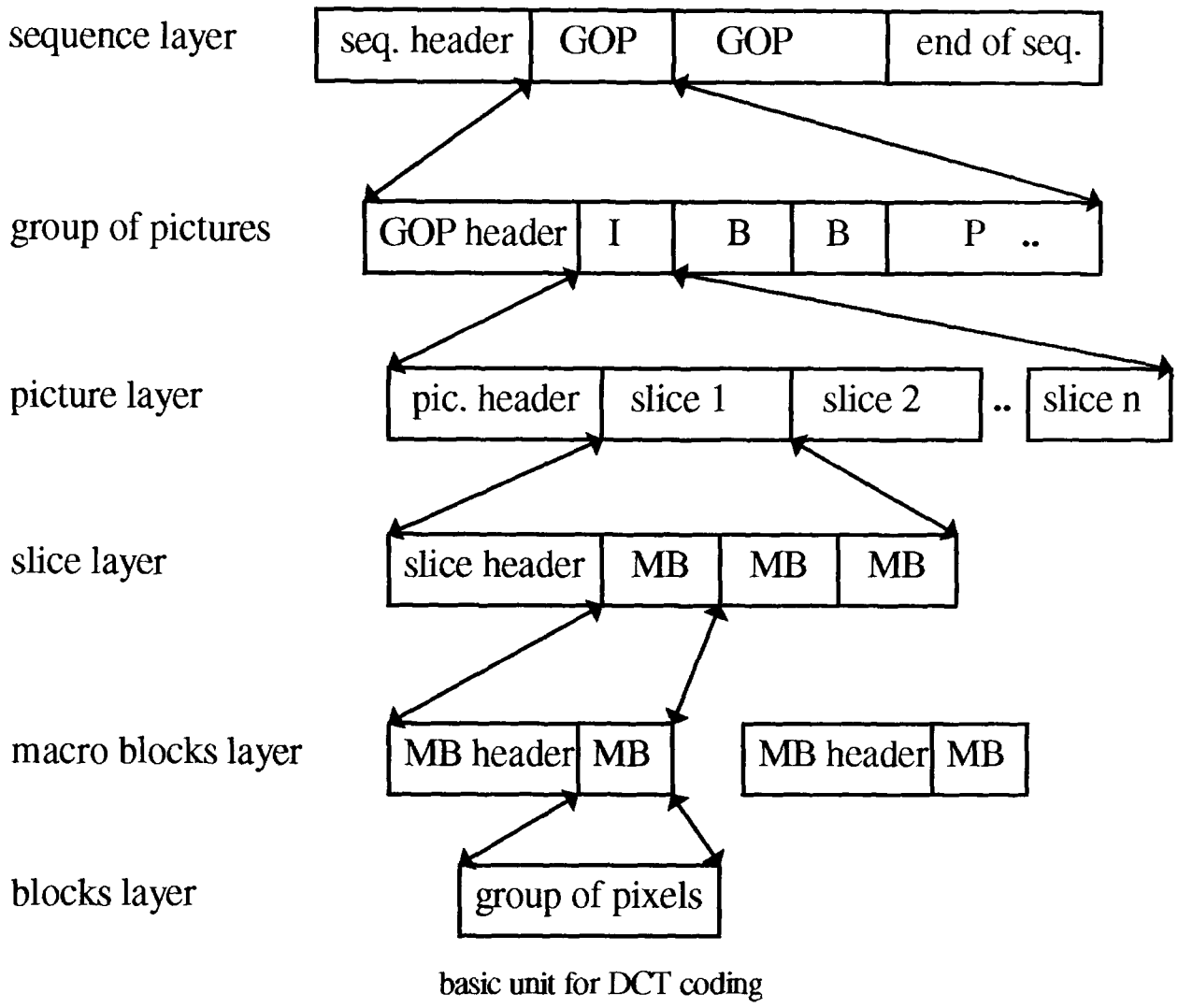



**Figure 2.14 MPEG Sequence Structure**

The MPEG standard consists of six layers (sequence layer, group of pictures, picture layer, slice layer, macro-blocks layer and block layer) as shown in Figure 2.14. Each layer begins with a unique start code. The highest layer is the sequence layer, which consists of a header followed by one or more GOP and ends with an end-of-sequence code. Each GOP may also have a header which will carry video information such as vertical and horizontal picture sizes, pel aspect ratio, picture rate and bit rate etc.

A "Group of Pictures" (GOP) layer has the typical sequence as shown here :-

$I_1\ B_2\ B_3\ B_4\ B_5\ P_6\ B_7\ B_8\ P_9\ B_{10}\ B_{11}\ P_{12}$ ..

but is re-ordered and transmitted as $I_1\ P_6\ B_2\ B_3\ P_9\ B_4\ B_5\ P_{12}\ B_7\ B_8\ P_n\ B_{10}\ B_{11}$ ..

The GOP layer contains at least one I frame and consists of the actual frames (I, B and P). Each picture is divided into slices. Each video frame is made up of a variable number of slices. A slice contains a 40 bit header and one or more macro-blocks. The macro-block is the basic unit of coding for motion compensation and the block layer is the basic unit of coding for the DCT. The lowest independent data unit occurs at the slice layer [IZQUI96], since it does not require data from any other slice for decoding. The slice header contains the position of that slice within the picture and the quantisation scale.

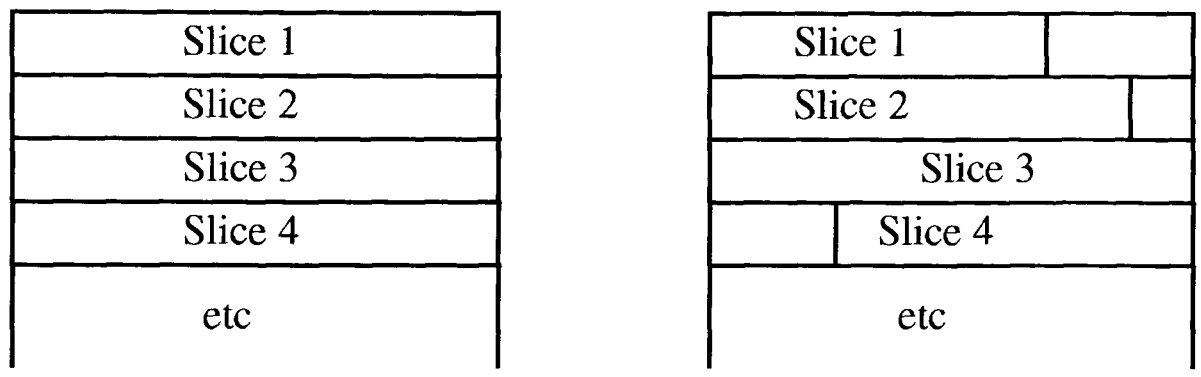



**Figure 2.15 The Arrangement of Video Slices within a Picture**

For VBR video, the number of slices per frame is flexible and determined at the time the video is encoded. It is possible to have one slice per picture, to reduce the overheads per slice, however, for transmission over ATM networks, multiple slices are preferable as errors may occur within slices. The actual number of cells per slice, for the sequences investigated, varied depending on which video clip was analysed. For the three separate video clips investigated by [IZQUI96] the peak number of cells/slice was 40, 57 and 112 respectively.

The use of the AAL-1 constant bit rate adaptation layer maps one 188 byte transport stream packet into 4 ATM cell payloads (47 octets per cell) and is used to transport uncompressed CBR video or audio.

AAL-5 operates with variable length payloads and appends a length indicator and a 32 bit CRC. One or more transport stream packets map into five, eight or more ATM cells [RUIU96]. Although AAL-5 operates with variable length payloads there are still at least four unused octets per cell when used to transport compressed VBR video [ABBAS92, DAG95, RUIU96] and there are currently debates about whether these should be used to carry timing information. It would appear from the literature that AAL-5 will be the preferred protocol for VBR video. It is currently the leading choice for American equipment manufacturers [GRAH96] and according to a recent HP survey the majority of MPEG design groups are planning to support AAL-5 [RUIU96a].

#### 2.3.3.2 Models of Video Traffic

The modelling of video traffic in an ATM network can be carried out either analytically or by simulation. At present there exits no widely accepted model which lends itself to mathematical analysis [CONT95]. The traffic generated by VBR video sources has complex characteristics which can not be effectively described in terms of traditional traffic models. Analytical models tend to require many assumptions [DAG95], and can be too restrictive. However, many simulation models also suffer from the severe disadvantage that the modelling process is based on statistics taken per picture frame. Realistic simulation requires the provision of traffic at the cell level. Most video models are modelled at the picture layer [IZQUI96] using autoregressive Markov process (AR(1) and AR(2)) models. The output of an AR(1) model has a Gaussian steady state distribution and an exponential autocorrelation function [SKEL93].

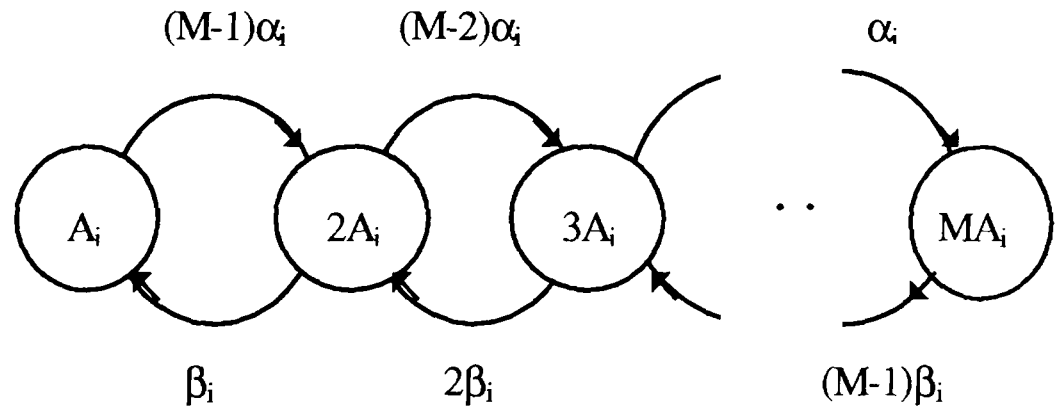
Packet video is modelled in [CHOW94] and each video source is modelled as a users conditional replenishment compression algorithm. Two reference frames are stored, one at the source and the other at the destination. The picture is scanned every 1/30 seconds, and each generated frame consists of 250,000 pixels. The frame is then compared to the reference and the differences are transmitted. A first order AR(1) model is used [CHOW94] but the parameters used were chosen to match the specific video encoding under consideration.

Statistical analysis of video sequences [IZQUI94] has revealed that each sequence has different characteristics which depend on the content of the video clip used. Modelling of an MPEG video stream using an MMPP process does achieve a close approximation to the associated video clip under investigation, but the state transition probability matrix is specific to that particular video sequence. An 8-state model is proposed [IZQUI96] with each state representing a number of cells per slice and the peak number being 40 cells per slice. Each state represents up to 5 cells per slice more than its predecessor ( peak No. of cells / 8). For example,

state 0 = 0 - 5 cells/slice
state 1 = 6 - 10 cells/slice
state 2 = 11 - 15 cells/slice etc.

The compression of video data causes the resulting traffic to be highly variable [CONT95], dependent on the adopted encoding scheme and on the activity within the movie under consideration. A bidimensional Markov chain is used and again the transition probabilities are estimated using an MPEG-1 trace making the resulting output fit the specific video clip under consideration.

A video model is proposed [ABBAS92] with exponentially distributed On and OFF periods. The mean ON period is 2.208 ms and the mean OFF period is 31.125 ms. The video payload is considered to be 44 bytes, with 4 bytes for the AAL overhead.




i = layer (base or second)

**Figure 2.16 10-State Video Model**

Two 10-state models are used to approximate each of a 2-layer video stream for video-conferencing (head and shoulders) [DAG95]. Video is coded as a base-layer with restricted bandwidth and the second layer contains more details and uses smaller quantisation steps. Since there are no scene changes, both layers are independently modelled by a Discrete M-state Markov Chain. Each state in the chain represents a quantized number of cells per frame. The transitional probability between the states and the number of generated cells per frame are derived ($\alpha$ and $\beta$) knowing the statistics of the coded video to be modelled, and again is dependent on the particular clip used. This model is also activated at 33.333 ms frame intervals and cell generation in state K is K times that of state 1. A 10-state chain (M = 10) is used for each layer, as shown in Figure 2.16. The generated cells per frame are evenly distributed within the frame. This distribution follows a modulated Poisson process with interarrivals having a gamma function. However, these Markov chains are unsuitable for performance analysis, since they are activated at video frame intervals. To represent the cell generation of both layers in shorter intervals a more descriptive model is required. The burstiness of the traffic carried by ATM networks is not captured well by Poisson and gamma processes. The performance of these networks may depend on how densely the cells arrive, which is described by the burstiness of the source.

For video with scene changes the behaviour of a 2-layer VBR coder depends on the bandwidth allocated to the base-layer [DAG95]. The average intervals between scene cuts varies between 5 and 9 seconds and the distribution can be approximated by a geometric distribution. If scene change frames are ignored, then the quantized cells per frame, in this case, in both layers, follow a binomial distribution. If this is the case then the base-layer of a particular sequence can be represented by a linear Markov model.

A general model is used in [SKEL93] which can characterise a wide range of sequences independently of parameters such as scene contents and coding algorithm used. Based on the results obtained, it was concluded that the long term correlation found in video streams is an important characteristic, but the actual form of the correlation is not. A histogram approximation is used to model video, which is an aggregate arrival process that can be directly calculated from the histograms of individual sources, using convolution. As a comparison video traffic is also modelled as a Markov Modulated Rate Process (MMRP). An 8-state Markov chain was chosen

for this and the transition rates between the states determined, either from the sequence's statistical parameters or by directly measuring its transition probabilities.

Forward error correction (FEC) can be used to try to protect video data from errors at the decoder, due to lost cells. FEC may be detrimental at higher loads [RILE95]. As the network load increases, the losses are heavily correlated to the traffic increase. There is a case for FEC on the video data most sensitive to error distortion rather than to all the data, with the overhead that this implies. Other ways to control the errors caused by the loss of a video cell need to be found.

### 2.3.4 Data Traffic

Data traffic is often modelled as a constant bit rate traffic source which is very sensitive to cell loss. A protocol data unit (PDU) is passed down from the higher layers to the AAL for segmentation and assembly into ATM cells. If a cell has been dropped or mis-routed there is no way of knowing which cell from the PDU has been lost, as ATM cells have no sequence numbers. This means that the cells which have arrived will be discarded and the whole PDU must be re-transmitted. From the observations of data traffic on a LAN [FOW91] a PDU may contain up to 210 ATM cells. If the cell loss is due to congestion, then re-transmitting the PDU will not be very helpful. A much better approach would be to ensure that data cells do not get routinely dropped, as is considered in this work.

#### 2.3.4.1 Characteristics and Coding of Data

Data networks have been around for some time but in-spite of this, the characteristics of data traffic is not well understood. This may be partially due to the fact that there is no typical data connection, as there is for speech. Data transfers range from large data files which output continuously, to small scale queries which cause short bursts of traffic. It is even difficult to predict the behaviour of a particular type of connection, for example, in client/server computing, because the amount of data and the dynamics of the exchange vary significantly from one application to another.

The data traffic found on a LAN has a high peak-to-mean ratio [FOW91]. For example, the peak bit rate in a 5 second interval is 152 times the mean arrival rate and in a 5 ms interval it can be 715 times the mean, which differs significantly from simple arrival models. A more sophisticated

traffic model is required [FOW91] in which individual users are simulated, which would show the same dispersion behaviour as that observed.

The range of time durations and bit rates of some typical data applications is described by [WEINS87].

#### 2.3.4.2 Models for Data Traffic

Data message arrivals are often assumed to be a Poisson process, with each message decomposed into a number of packets with a general distribution [LI85]. Data was modelled as an ON/OFF source [ABBAS92], with the mean ON period being 125 ms and the mean OFF period being 250 ms using an exponential distribution. Data generated from a single data source is well characterised by a Poisson arrival process [BAE91]. However, analysis of FTP sessions and TELNET packet arrivals over a WAN, [PAX95], shows that this type of traffic is not well modelled using a Poisson process, as the burstiness of the traffic is not captured well by this type of model.

The data model presented [YEGE94] uses a 4-state MMPP model to capture a large number of arrival statistics from a homogeneous system of high-speed ON/OFF sources. This model was then used to generate traffic from a heterogeneous system with two classes of ON/OFF sources.

The most commonly used traffic models are Markov Modulated Poisson Processes (MMPP) and Markov Modulated Bernoulli Processes (MMBP) [ARV95]. These models tend to be used due to their ability to match various traffic statistics and their mathematical tractability. However, little is known about their ability to generate bursty traffic that is equivalent to real traffic.

## 2.4 Multiplexers

### 2.4.1 Multiplexer Gain

The performance of a multiplexer is extremely sensitive to the burstiness of traffic streams with widely differing characteristics. There has been a lot of work done on the effectiveness of statistical multiplexing with heterogeneous traffic streams. The multiplexing gain suffers from the heterogeneity of the sources, which is caused by differences in the peak rates [SMIT94]. It is not

effective to statistically multiplex many sources with widely different characteristics [BONO92 and BAE91]. Statistically multiplexing these types of sources [BONO92] requires the use of large buffers (>500 cells), which is unacceptable to real time traffic because of the resulting queuing delays. The multiplexing gain decreases with bursty sources. Longer burst lengths result in a worse network performance [BAE91], an increase in the cell loss probability and the end-to-end delays experienced. The effect of longer burst lengths causes statistical multiplexing to be less effective and means that fewer active sources can be supported for a given bandwidth. A statistical gain is only possible with heterogeneous traffic when using large buffers with a large number of slow speed CBR sources [BONO93].

Statistical multiplexing results in a finite probability that the sum of the instantaneous rates of the multiplexed connections will exceed the link capacity [MIT94], with resulting cell losses or may even lead to long delays. The conclusion drawn [GERZ91] is that statistical multiplexing may not be appropriate for heterogeneous traffic.

[KARL96] advocates deterministic multiplexing as the natural choice for traffic with long bursts or if the quality of service required by a call is high. He also states that the prime motivation for statistical multiplexing was to allow efficient use of bandwidth. Since this is no longer a scarce resource, alternatives which guarantee quality may be preferred for real time traffic, by the network users. However, deterministic multiplexing, with peak rate allocation of bandwidth for each connection, can not totally guarantee no losses or delays, as concurrent arrivals of cells may result in short term link overload [MIT94].

Few present day applications are suitable for statistical multiplexing. Telephony, facsimile and file transfer do not have enough variability to have a statistical multiplexer gain. High bit rates e.g. video retrieval and HDTV cause high peak rate to link rate ratios at which no statistical multiplexer gain can be achieved [SMIT94].

Highly non-homogeneous traffic streams need separate treatment for each type of traffic. It does not pay to try to "squeeze" out statistical gain when input/ output speed ratios are small and when only a few streams (<20) can be merged [BONO93]. Their work also supports the conclusion that statistical gains are only relevant when using large buffers and that it appears not to be effective to multiplex sources with very different characteristics.

### 2.4.2 Scheduling Policies

Various priority schemes have been proposed to improve the performance of the multiplexer.

A study [CHANG94] investigates a queue length threshold (QLT) scheduling policy for an ATM multiplexer using batch Poisson arrivals. Cells are divided into real time and non-real time traffic. If the non-real time queue length exceeds the threshold, then priority is given to the non-real time queue, otherwise the real time queue has priority.

Dual queues are proposed with limited cyclic service, which protects high priority traffic from low priority traffic overload [HART91]. Low priority cells are defined as cells with their CLP indicator set to 1. These low priority cells are discarded at the first sign of congestion. Buffer management schemes include push-out and partial buffer sharing. The push-out scheme operates when a high priority cell arrives at a full buffer causing a low priority cell to be discarded to make room for it. The main problem here is maintaining the cell sequencing. Partial buffer sharing using adaptive thresholds is costly to implement and can lead to system instability if the thresholds do not adapt fast enough, but the scheme does guarantee a minimum amount of bandwidth for each class of traffic.

The discarding of low priority cells to reduce delays is proposed [GERZ91]. When the delays to speech cells are too great, they are dropped on arrival which can cause gaps in the reconstructed speech. Selective discarding of low priority speech and video cells is advocated [ABBAS92] as a congestion control mechanism and several possible candidates are examined. These are input discarding, output discarding, I/O discarding and push-out discarding. In input discarding a marked cell is discarded if it tries to join a full queue, while output discarding marked cells are dropped at service time if they are beyond a threshold, when the front of the queue is reached. I/O discarding is a mixture of both with different thresholds. Push-out discarding is also proposed [HART91], as described above. Priority should be given to video cells due to their sensitivity to delays [ANAG91] .

Using a threshold-based priority packet discarding scheme traffic is split into two streams and priority levels are introduced [KIM96a]. Low priority cells are dropped if necessary and loss sensitive traffic (data) is given priority over speech. Discarding of video cells is also allowed.

Low priority cells are only accepted if the current system occupancy is less than a threshold. The benefit here is that buffer requirements are reduced.

The performance of four scheduling policies are compared for a statistical multiplexer [CHIP89]. Traffic is divided into non-real time and real time, with the non-real time traffic being defined as cell loss sensitive and the real time as delay sensitive. The scheduling policies are first in first out (FIFO), priority for real time cells, minimum laxity threshold (MLT) and queue length threshold (QLT).

- FIFO - all cells are served strictly in the order in which they arrive.
- Priority service for real time cells - the real time cells always have priority over the non-real time cells. Service for the real time cells is improved, but the delays for the non-real time cells are increased.
- MLT - each real time cell has an associated deadline (laxity) after which there is no point in transmitting the cell, so it is deleted. If the minimum laxity is below a pre-defined threshold for the real time queue, then that queue has priority otherwise priority goes to the non-real time queue.
- QLT - defines a threshold for non-real time cells and if the number of cells exceeds that threshold, then priority is given to the non-real time queue, otherwise the real time queue has priority.

To test these scheduling policies the multiplexer is modelled as a discrete Markov Chain for FIFO, MLT and QLT, and as a M/G/1/$\gamma$ queue for priority service. This study has shown that both MLT and QLT give improved performance over the others, but QLT is easier to implement.

A performance evaluation of ATM scheduling policies has been carried out using Petri Nets [HAV94]. Four policies were investigated and these were :-

- FIFO, without priority
- FIFO-PR, with pre-emptive priority
- Threshold priority
- Exhaustive threshold priority

In FIFO without priority, the single queue is served without discrimination between real time and non-real time cells. Whereas, for FIFO-PR, there are separate queues for real time and non-real

time traffic and real time cells are given absolute non-pre-emptive priority over non-real time cells. For the threshold priority scheme, a threshold is determined for the real time queue. Both queues are serviced alternately, until the real time queue length is greater than the threshold. Then the real time queue receives priority service until the queue length falls below the threshold, at which time the server returns to alternate service. Various values for the threshold are examined. A threshold of zero gives absolute non-pre-emptive priority to the real time queue. A variation of this is exhaustive priority service. Here, once the server begins priority service, it continues to serve the real time queue until there are no more cells waiting.

These scheduling policies were the subject of a simulation study [GAN95]. Here, a single queue with FIFO scheduling was compared to alternating cyclic service and the threshold priority scheme as described above [HAV94]. It was found that separate queues for the real time and non-real time traffic reduces the delays to real time traffic, with the best service occurring when a threshold priority scheme was used.

The conclusion drawn is that separate treatment is required for widely differing traffic sources. Large buffers introduce unnecessary delays and may increase CDV on individual channels. This can be a serious problem for some traffic sources, particularly speech, since the decoders are particularly sensitive to jitter, which can be made worse by being mixed with other types of traffic.

It is generally agreed that splitting the buffer in a multiplexer can benefit both streams of traffic. The differences are in the definition of that split. Splitting the buffers into real time and non-real time traffic and introducing some kind of priority service for one of the queues is also popular. Since speech is a CBR source, it should have a separate queue at the multiplexer [ARN95] and that all other traffic should be queued in a "bursty" queue. Output from the bursty queue in this case is strictly controlled. Dividing the traffic into prioritised classes allows use of traffic conditioning to smooth bursty sources, to protect speech cells in the network.

## 2.5 ATM Switches

An ATM switch operates at high speed so as to absorb the variations of the different types of traffic, particularly the burstiness of the video traffic, while keeping the quality of service constant. An ideal switch is one which can route all cells from their input port to the required output port without losses [WAN93]. The high speed operation within the switching fabric requires that the processing of cells is mainly hardware based. A switching fabric can either be blocking or non-blocking. Blocking means that the switch is unable to provide simultaneous independent paths between arbitrary pairs of input and output ports. A non-blocking switching fabric may also suffer from congestion if two cells arrive independently, at different input ports, destined for the same output port. Contention will then occur for that particular output port. This type of congestion is unavoidable, since cells can arrive at any time and the cells need to gain access to an output port. Since only one cell can be served, additional cells will be queued and inevitably delayed.

When a connection is established in an ATM network, routing tables are set up at each switching node. ATM cells are then transported across the network using the routing information obtained from the cell headers and the routing tables. Each connection has a unique identifier and an output link identifier, which is used to route the cell to the correct output port within the switch. The routing tables provide the mapping between the input and output ports for each connection.

ATM switches can be classified according to the position of the buffers within the switching fabric. Switches use input queuing, output queuing or shared queuing, or a mixture of these, within the switch. The switching fabric is generally operating a number of times faster than the input lines [HLUC88].

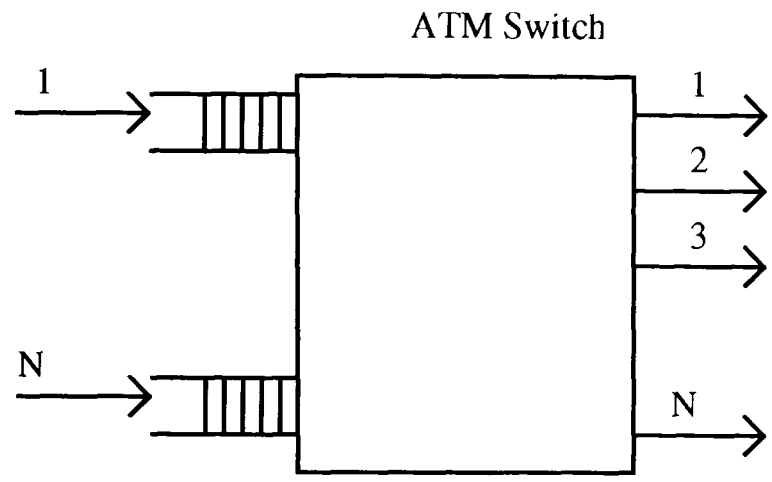



**Figure 2.17 Input Queuing at ATM Switches**

In input queuing the arriving cells are queued at the input port and are then switched across to the destination output port for transmission, [HLUC88, LIEW90, AMBR92 and BADR94], see Figure 2.18. Only one cell may access a particular output port at a given moment. If more than one cell wants to use the same output port, then contention occurs as only one may gain access and the other must wait. The cell waiting at the head of the queue may also be blocking the cells behind it from accessing other, possibly idle, output ports. This is called head-of-line (HOL) blocking, which does cause a reduction in the throughput for a large number of input and output ports, according to many sources [GUPT93, WIDJ93, PATTA93 and BADR94]. To solve the problem of HOL blocking, contention resolution schemes have been proposed [CHEN94], which relax the strict FIFO discipline, but this causes an increase in the switching fabric complexity. One variation of input queuing uses a switching fabric that operates a number of times faster than the input lines. Mini-slots may also be used to divide up each cell time slot and this effectively reduces the offered load by the speed-up factor. It is possible to clear more packets in each time slot, but does require buffers at the output ports. Another strategy used is to increase the number of output ports to allow more than one cell to be served from each input queue [LIEW90].

Output queuing appears to be the preferred method with ATM switch vendors [JAIN96], see Figure 2.18. Here, arriving cells are directed to the correct output port, using the routing information contained in the cell headers. The cells are then queued at the output port to await transmission in a strictly FIFO order. The advantage here is that cells arriving at the same input port, for different destinations, do not block one another. Also if all the input ports contain cells

for the same output port, they can all be routed to that port in one time slot. Only one cell will be transmitted at each output port, with the others waiting in the queue. Long delays can occur in queues, if many cells want to use the same output port.

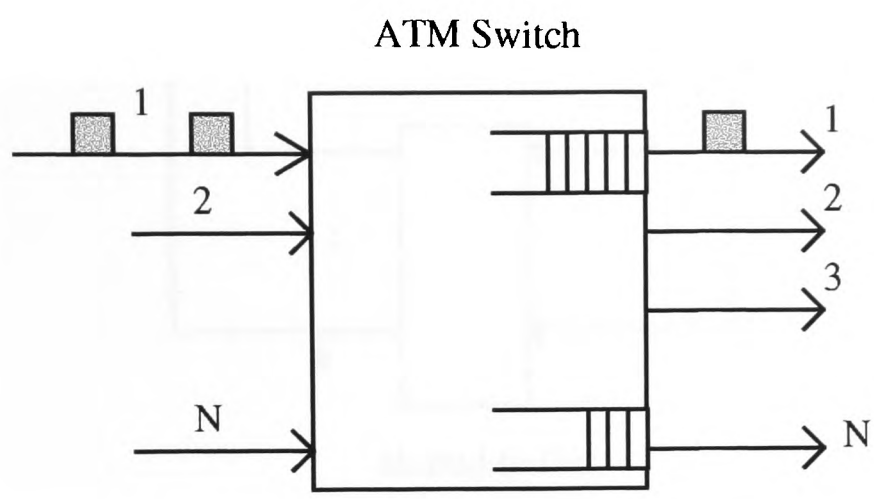



**Figure 2.18 Output Queuing at ATM Switches**

Another alternative is to use completely shared buffers. A single buffer space is shared by all input ports for writing and all output ports for reading. This is usually implemented as logical queues which are associated with each output port [AMBRA92]. This achieves optimal throughput and delay performance [HLUC88], and smaller buffers are required overall. Buffer sharing [LIEW90] is an effective way of dealing with bursty traffic. Two examples of switches which use this type of buffering are the Starlite and Prelude switches, see Figure 2.19.

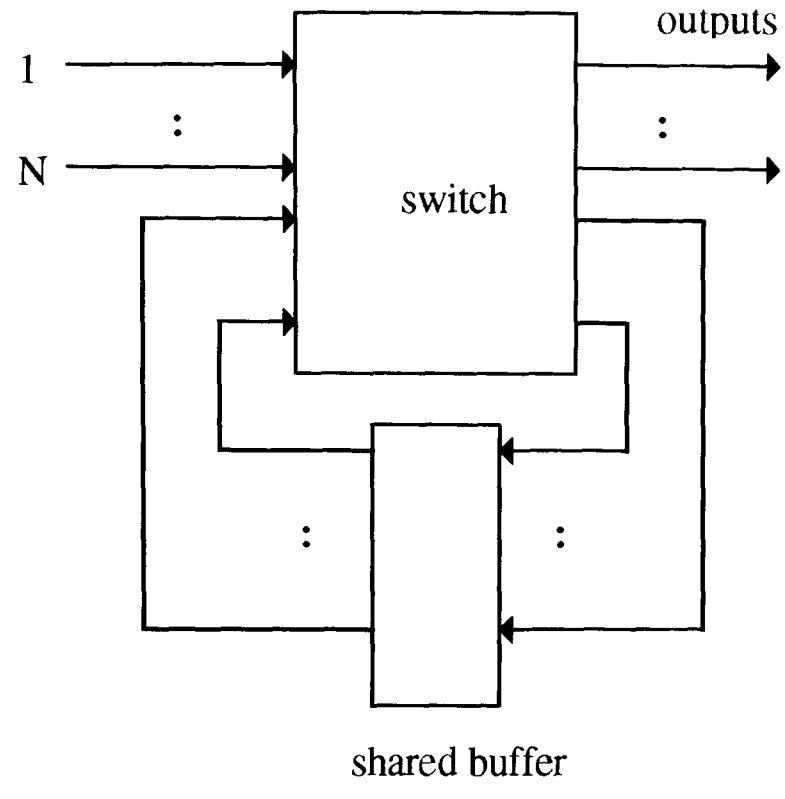



**Figure 2.19 Shared Buffer - Starlite Digital Switch with Trap**

In space division switch architectures, [WAN93, WENG91] multiple cells can be switched from the different input ports to output ports, concurrently. A dedicated path is established through the switching fabric for each cell, which allows control to be distributed within the switch. One type of space division architecture is the Banyan network, which is based on a matrix topology. A Banyan network is self-routing [PATTA93] and has the property that there exists a unique path from any input port to a particular output port, see Figure 2.20. Internal blocking can occur when two cells try to access the same link between two stages, which can reduce the throughput. To reduce the effects of internal blocking, three methods can be used and these are internal speed-up, internal/input buffering and pre-sorter networks. These types of switches require buffers to be placed where potential conflicts may occur [WAN93]. Banyan networks are scaleable, as the modules can be used to build larger networks without modification to the algorithms used. To make space division switch architectures internally non-blocking, all possible combinations need to be realised. Unfortunately, this means losing the self-routing [ONV94]. A Banyan network is non-blocking if the cells to be switched are sorted before being switched [PATTA93]. An example of this is the Batcher-Banyan switch [TAN96]. The Batcher switch pre-sorts the cells as they arrive at the input port so that the Banyan switch can transport them without blocking.

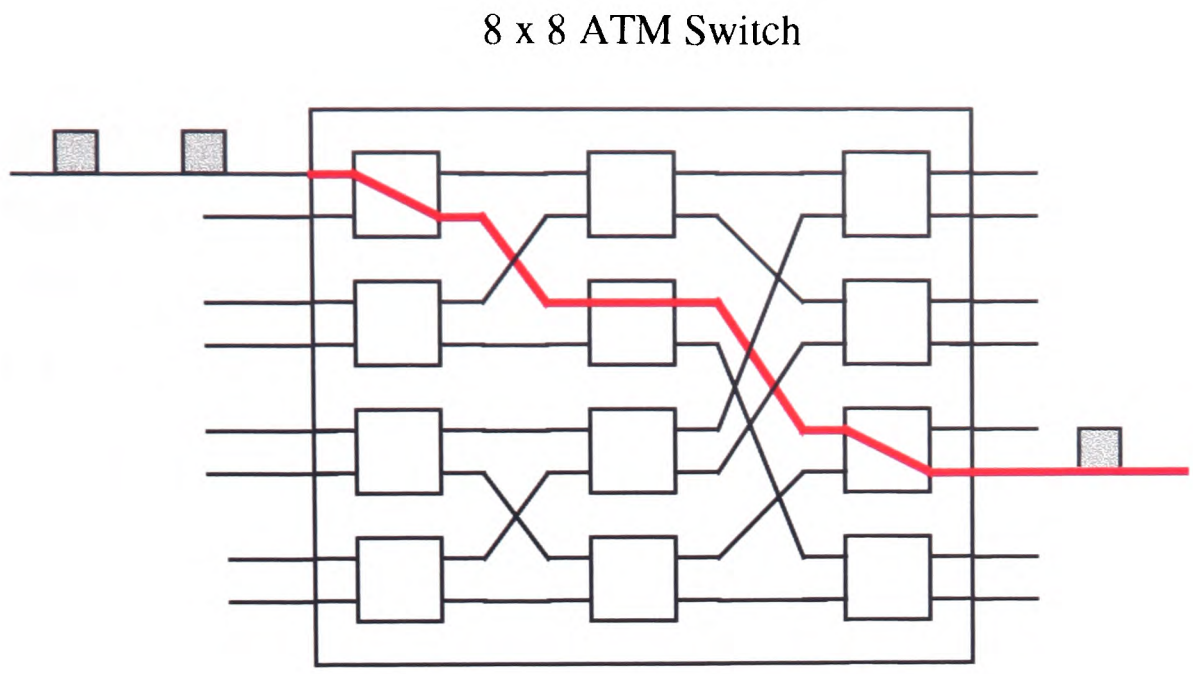



**Figure 2.20 Banyan-type Switch**

It appears from the literature that a mixture of input and output queuing, with some internal buffering to help resolve conflicts, can provide non-blocking switching fabrics which will help eliminate cell losses due to collisions.

### 2.5.1 Models of ATM Switches

When modelling an ATM switch with the purpose of analysing the output processes from that switch, the actual architecture of the switch is not the focus of the work. [WAN93] observed that in these cases, it is better to model a non-blocking generic switch, so that the switching strategy used will not influence the performance. A simulated switch may be assumed to be a perfect output queuing switch, and hence will have no effect on the arrival of cells [FOW91]. This avoids linking the analysis with a particular switch architecture. This is a reasonable assumption for switches that operate faster than the transport facilities that connect them.

## 2.6 Preventative Congestion Control

Congestion control is an important aspect of an ATM network. On a single ATM link operating at 155.52 Mb/s, 350,000 cells per second may be arriving at a switch. A switch may have 100 input lines, and cells must be switched across the switching fabric in real time (2.7 micro

seconds) to the correct output port. If congestion occurs at a switch within the network, low priority cells are initially discarded to protect high priority cells, and although this is not part of the ATM standard, [JAIN96] states that many switch vendors are implementing it. ATM cells are not numbered, and so many cells must be re-sent to replace one missing cell (data traffic). The loss of a some cells of a burst may result in the loss of the information content of the whole burst [DAIL96]. A cell deleted from a higher layer protocol data unit will require the whole data unit to be re-transmitted [NITTO92, PLATT94] and this can be counter-productive if the loss is due to congestion and could even lead to congestion collapse.

Previous generation networks handled congestion by telling "upstream" nodes to throttle back or stop transmitting or by diverting packets round congestion "hot spots" via alternative routes, using feedback schemes. Lost packets could easily be replaced by requesting the missing packet. Since ATM rates are very high, existing packet network error control strategies such as ARQ and go-back-N are not feasible.

Careful control and policing at the User to Network Interface (UNI) is essential. By not allowing traffic which would cause an overload onto the network, and by monitoring the connections that are already active, congestion can be kept to a minimum.

There are two aspects to preventative congestion control. These are the call admission control (CAC) and user parameter control (UPC). CAC and UPC are discussed in more detail below.

When a call requests access to the network, the CAC checks that there are sufficient network resources available for that call. A contract is negotiated between the originating system and the network. The contract will have QoS characteristics associated with it, such as mean and peak rates. Providing the user stays within the negotiated limits then the network should function smoothly. However, once users gain access to the network they could, in theory, transmit at any rate, as cells are transported transparently, so there needs to be some form of monitoring for the duration of the call. Since there is no link by link flow control, the only point of restriction is the access to the network at the UNI. The network then monitors all connections for contract violations. This is called source policing.

### 2.6.1 Call Admission Control

It is necessary to control access to the network. This is done by only allowing those calls for which there are sufficient network resources. Admission control is invoked for each link between the originating and terminating points of an ATM connection [RASM91]. The new connection will only be allowed if it is accepted for each link on a particular route.

Peak rate allocation for users gives strong performance guarantees [TURN92], and it is easy for users to understand and simple to implement. The draw-back is the poor use of bandwidth with bursts of traffic. The best solution for calls requiring loss free connections is peak rate allocation [DAIL96].

An acceptance algorithm [WALL90] for heterogeneous traffic makes the decision to accept a call based on both the peak and the mean rate of the call and of the existing connections as well. The algorithm ensures that the cell loss probability of each virtual connection does not exceed a pre-defined limit.

A pacing mechanism is proposed [KARL96] which controls the departure instances of the reserved cell-stream and the reserved buffers which are guaranteed never to overflow. There are two kinds of connection defined and these are, one which reserves capacity and the other which is described as a best effort and has no bandwidth allocation. The pacing function is performed per link and not per VC.

A CAC algorithm proposed [DAIL96] checks if the throughput requirement can be satisfied, when a call requests admission. It also checks if the QoS of the existing calls can be maintained. If either fails, then the call is rejected. There are three different algorithms, one for each of three classes of traffic. Type A traffic require loss free connections, type B traffic are burst scale and type C are cell scale traffic. For a typical mix of 500 sources, the CAC algorithm takes less than 5 ms. It provides a wide variety of guarantees to users through the coexistence of these three types of service. Another CAC proposed [LAET95] maintains two types of connection (high priority or low priority). The rate of the individual connections plus the new connection must not exceed the total link capacity allocated to that type of connection.

A CAC and a UPC is proposed, [MAS96 and YAMA92], which ensures that during the information transfer phase, connections which exceed their specified traffic rates do not deteriorate the QoS of other conforming users.

### 2.6.2 Policing Strategies

The ATM layer assigns all cells the same priority at queues i.e. first in first out (FIFO). This may not be the most appropriate discipline for video and telephony applications [GERZ91], where a delayed cell may arrive too late to be included in the reconstruction. Alternative queuing disciplines such as oldest-cell-first, which defines the priority of a cell as the time spent in the network, or selectively discarding cells to reduce congestion may be more appropriate. However, the difficulties of tracking the time spent in the network for each cell, given that the cell header is very small and the increased complexity in switches and multiplexers makes this impractical.

Windowing has been the favoured method with the previous generation networks. This method was ideally suited to fixed bandwidth transmissions, using numbered packets. Lost or corrupted packets could easily be replaced by asking for a repeat transmission. A copy of the packets within a particular window were held until an acknowledgement was received that they had been received correctly. If an error had occurred, then either a single packet was requested again or the entire window was re-transmitted. With the longer propagation delays found in ATM networks, windowing is no longer an appropriate technique.

A basic congestion control strategy should consist of a buffered leaky bucket and spacer [GUN93], operating at the source of each origin - destination pair. The burst length of the source affects the buffer requirements, even at low utilisation [BADR94] .

The leaky bucket allows cells to enter the network at a certain rate, called the leak rate. If cells arrive at a faster rate than specified, they are penalised, by either being discarded or by being tagged as low priority cells, which may subsequently be discarded should congestion occur in the network. There are two main implementations and these are the virtual scheduling algorithm and the continuous state leaky bucket, which are both equivalent, according to [KEY94 and WAL93], and both of which are discussed in the following sections.

The main argument against any flow enforcement is that a large time constant is needed to avoid excessive cell loss from a statistical source that is generating cells close to its mean rate, [DITTM91 and RATH91]. The large time constant also implies inefficient control of the traffic flow. Optimum dimensioning and the effectiveness of the policing mechanism depends heavily on the characteristics of the traffic sources and their QoS requirements [RATH91]. Policing close to the mean bit rate requires unrealistically long sampling periods.

Conventional statistical bandwidth allocation methods based on the leaky bucket cannot guarantee the QoS [YAMA92].

Positioning of the congestion control function is also important for the performance received. Policing may take place at the customer's premises according to [JAD95]. It was proposed [NEIST90, RATH91] that the leaky bucket should be as close to the source as possible. [YAMA92] advocates that cells must pass through the leaky bucket before entering the multiplexer and gaining access to an ATM network and [GALL89] also places the bandwidth enforcement devices before the multiplexer.

The standard multiplexer to leaky bucket configuration causes a greater CDV between successive cells [JAD95], as they are queued at the multiplexer and get "clumped" together. Dimensioning and the effectiveness of the policing mechanism depends heavily on the source traffic characteristics and on the QoS requirements. Deterministic rule-based traffic descriptors offer advantages over statistical based ones.

There have been some studies to compare various windowing techniques with the leaky bucket method. One such study, [MOLL94], compares the moving window, jumping window and the stepping window with a leaky bucket mechanism. The conclusion is that windowing techniques would fail when policing the peak rate of a source, as they would either not catch all illegal cells, or they would discard legal ones. The performance of the leaky bucket, jumping window, the triggered window, the moving window and the exponentially weighted moving average (EWMA) mechanism are compared [RATH91]. The jumping window, moving window and EWMA mechanism with the leaky bucket were also compared [RAD96]. Both concluded that the leaky bucket and the EWMA mechanisms are the most promising. Another study [DITTM91] also compares a leaky bucket with a rectangular sliding, triangular sliding and the

EWMA mechanisms. The conclusion drawn here was that, contrary to popular belief, the rectangular sliding window is competitive with respect to cost-effective implementations. However, the main drawback is its lack of flexibility, in that the window length is fixed during implementation. This is not a constraint found in the leaky bucket algorithm, which is therefore more adaptable.

The most important policing strategies include the virtual scheduling algorithm, the continuous state leaky bucket, avalanche tagging and multiple virtual leaky buckets, which are discussed in the following sections.

#### 2.6.2.1 The Virtual Scheduling Algorithm

The virtual scheduling algorithm [KEY94, CAS94 and KIM96b] uses an estimation of the expected arrival time of the next cell to determine if the cell is conforming or not. The theoretical arrival time (TAT) for the next cell is calculated, based on the declared peak bit rate. The expected arrival time of the next cell is then incremented by the TAT each time a conforming cell arrives. There is a threshold limit for early arrivals, which means that if a cell arrives earlier than expected, but within the threshold limit then the cell is conforming, otherwise it is not and would be discarded.

#### 2.6.2.2 Continuous State Leaky Bucket

The continuous state leaky bucket, [CAS94 and KEY94] consists of a counter, which is incremented by arriving cells, and which is decremented periodically. The counter is not allowed to become negative. If the arrival rate is greater than the decrement rate, then counter will quickly reach the threshold value, and all cells arriving subsequently are either deleted or tagged as suitable for deleting, until the counter value falls below the threshold. Variations use tokens instead of a counter. The difference between the two methods is that a cell must find a token waiting in the token pool before it can transmit and the tokens are periodically refreshed at an appropriate rate, (see description of the token leaky bucket).

When a long burst of cells is generated at near peak rate the threshold limit may be violated. Between bursts the rate falls back to below the mean and the overall average rate is satisfied. This means that bursty applications can gain access to the peak rate for brief periods, providing

the average rate remains below the threshold. Even when a traffic source is conforming, cells may be discarded if a long burst of cells arrives.

**Token Leaky Bucket**

A cell arriving at a token leaky bucket can only be transmitted if there is a token present in the token pool, otherwise it must wait in a finite buffer until a token becomes available. The token pool has a maximum size (M), also called the bucket depth, and the tokens are periodically refreshed at a steady rate. Tokens are stored up to the capacity of the token pool and any tokens generated when the token pool is full are lost. A diagram of the token leaky bucket is illustrated in Figure 2.21.

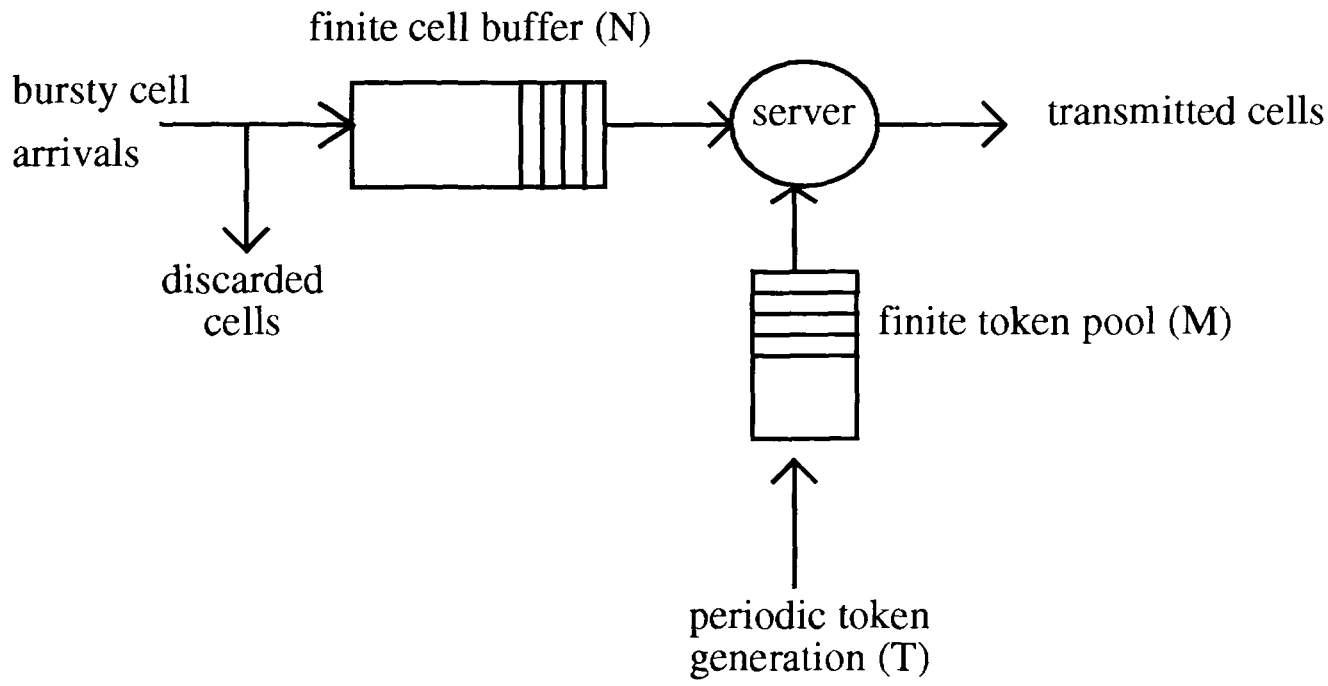



**Figure 2.21 The Token Leaky Bucket**

The peak rate may be controlled by dropping cells with inter-arrival times smaller that the minimum allowed. A cell arriving when there are no tokens available, must wait in the buffer until the next token generation period. Cells arriving if the input buffer is full are blocked and lost. In this case tokens are generated at the mean arrival rate. It was found that as the size of the token pool was increased, so the performance improved. There was a trade-off between the delay experienced and the cell loss probability [KIM92] .

Another buffer management strategy proposed [ONV94] is that a cell arriving at a full buffer is deleted if it is a low priority cell, or if it is a high priority cell, then it may "push-out" a low priority cell that is already queuing. If there are no low priority cells in the buffer, then the cell is deleted.

Another variation of the token leaky bucket [CHAN94], uses two token pools, identified as red and green tokens, which are generated independently, see Figure 2.22. If the number of cells waiting in the buffer is less than a threshold, then a green token is used to transmit the cell. If the buffer size is greater than the threshold, then a red token is used and these cells are marked as low priority. A cell may only use a single colour token. Cells arriving at a full buffer are rejected.

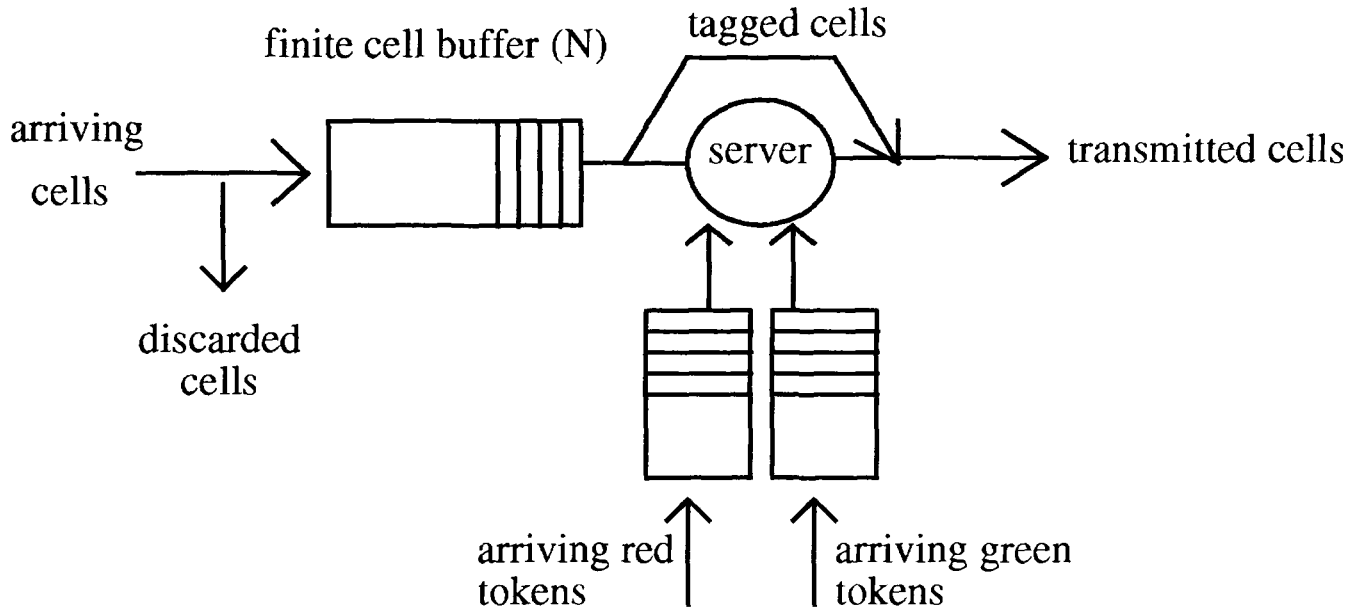


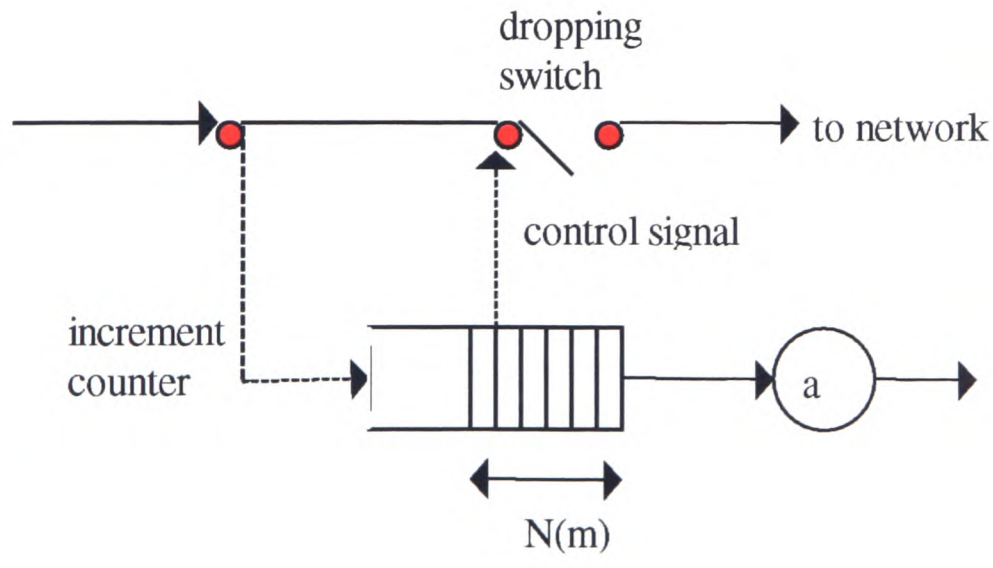

**Figure 2.22 Token Leaky Bucket showing Red and Green Tokens**

A tagging leaky bucket and a queuing leaky bucket were compared using simulation [TED93]. The tagging leaky bucket marks the cell as a low priority cell if no token is available. The cell is then allowed onto the network and will see a high loss probability within the network. In the queuing leaky bucket, cells arriving and finding no token are queued until one becomes available. A tagging leaky bucket which can react dynamically to the changing traffic conditions and protects the network from congestion is proposed.

**Virtual Leaky Bucket**

The virtual leaky bucket (VLB) behaves as a virtual FIFO buffer, but it does not store cells and nor does impose any delay on them, see Figure 2.23. This is essentially the same as the continuous state leaky bucket, except that the counter is decremented in discrete steps. Each time a cell arrives at the leaky bucket, the counter is incremented [JAD95]. Cells are allowed onto the network if the counter is below a threshold. The counter is decremented periodically. If the counter stays below the threshold, then the source is considered to be behaving correctly, and the cells pass though without being penalised. If the counter reaches the threshold, then the cells are either discarded or they are marked as low priority (tagged) and allowed onto the network.

A method is proposed for setting the parameters of the leaky bucket to reduce jitter for CBR services [NIEST90]. Decrementing the counter may be in single increments, but for greater flexibility, larger numbers improve performance, particularly for high bit rate sources. [NIEST90] states that it is quite valid to police the peak rate only in the case of VBR sources, but this does not take advantage of statistical multiplexing. A larger threshold value for video traffic is required which allows longer bursts of cells at the peak rate. For VBR video, a peak rate burst could last up to 10 seconds when there is a lot of movement in a scene and the leaky bucket must allow for this. This does mean that any violations in the overall mean bit rate can not be detected for at least 10 seconds, which may be too long.




a = leaky rate
N = threshold level

**Figure 2.23 Virtual Leaky Bucket**

A virtual leaky bucket which drops violating cells is proposed, see Figure 2.23, [BUTT91]. The conclusion drawn is that the traffic source characteristics directly affect the selected leaky bucket parameters. It was also found that the leaky bucket easily controls the peak rate, but the burst length is not so easily controlled.

### 2.6.2.3 Avalanche Tagging [NITTO92]

Avalanche tagging is an extension of the virtual leaky bucket. As with the virtual leaky bucket, the avalanche tagging algorithm uses a counter, which is set to zero and subsequently incremented by one each time a cell passes, until it reaches a threshold value. A decrement function periodically decrements the counter until it reaches zero. If a cell arrives when the counter has reached the threshold it is tagged and all following cells for that burst are also tagged. Although the counter is not incremented while these cells pass, it continues to be decreased at a constant rate. Tagged cells have a higher probability of being discarded within the network.

The avalanche tagging algorithm is dimensioned based on the following assumptions. The link bandwidth is 150 Mb/s and there are a fixed number of sources allowed, each with a peak rate of 10 Mb/s and a burstiness of 5. The mean rates of the sources were varied from 2 Mb/s to 4.29 Mb/s. This ensures that at the higher values, the intervention of the monitoring mechanism is certain. The mean burst length is 100 but the avalanche tagging mechanism is dimensioned using twice the burst length as the threshold value. Buffer dimension is fixed at 50 cells, to guarantee a maximum delay ($t_{max.}$) of 141 μs

The performance of a virtual leaky bucket and avalanche tagging were compared. For each policing method, two types of simulation were run, one set using a multiplexer with a threshold scheme and the other a multiplexer with a push-out mechanism. Sources were given different mean rates to make them exceed established limits, and cause the multiplexer to overfill and tagged cells to be discarded. Threshold values tried for the multiplexer with threshold scheme were 15, 10, 5 and 3. For threshold = 15, untagged cells were also dropped, this meant that the threshold was set too high. A threshold of 3 was found to be too low and the performance was

the same as for the virtual leaky bucket. This method actually deletes the same number of cells as the virtual leaky bucket, but the losses are compacted into fewer data units.

When a link is over utilised the multiplexer drops cells, both tagged and untagged, at the same rate as the virtual leaky bucket. However, the proportion of tagged cells dropped increased for the avalanche tagging algorithm, and so fewer untagged cells were dropped. The number of intact data units was greater when compared to the virtual leaky bucket, thereby increasing the throughput. The conclusion reached was that avalanche tagging meets the needs of loss sensitive sources, when looked at from the point of view of intact data units arriving at the destination as the number of re-transmissions is decreased.

#### 2.6.2.4 Multiple Leaky Buckets

The problem with using a single leaky bucket to police VBR sources is that of dimensioning the parameters [BUTT91 and JAD95]. Using the mean bit rate requires that the traffic generated must be measured over a long period, so as not to discard cells generated during long peak rate bursts, but this also means that the policing function may be slow to react to contract violations.

A dual leaky bucket was proposed [NIEST90], for VBR connections, which can be used to shape these types of traffic sources. The first leaky bucket has a leak rate the same as the peak rate of the source, while the second is dimensioned close to the mean rate. If either of the leaky buckets has their counter at the threshold value then the cell is discarded. Problems occur if the VBR source generates long bursts at the peak rate, as can be the case with VBR video when violent movement may last anywhere between 1 and 10 seconds. If peaks of 10 seconds are allowed, then the mean rate will not be detected until at least 10 seconds. A dual leaky bucket policing a source, with a peak rate of 10 Mb/s and a mean rate of 2 Mb/s, requires a threshold value for the mean rate policer of 200,000.

A dual leaky bucket is proposed [YAMA92] to police a variable bit rate source. The first leaky bucket monitors the peak rate, with a bucket depth of 1, while a second leaky bucket monitors the mean rate. They conclude that conventional UPC techniques based on the leaky bucket can not guarantee the QoS requested. A dual leaky bucket is advocated [JAD95], see Figure 2.24. The leak rate is dimensioned using the peak cell rate divided by 0.95. This allows a stream with a

constant cell rate which is 5% higher than the peak rate to pass through the policer. This is used to control CDV. The peak rate leaky bucket has a small buffer and the leak rate is close to the peak rate, while the mean rate has a large bucket size and a leak rate slightly higher than the mean. However, this is only effective for packet voice, still picture video and ON/OFF traffic with constant ON periods and uniform or geometrical distributed OFF periods that do not significantly change for the duration of the call.

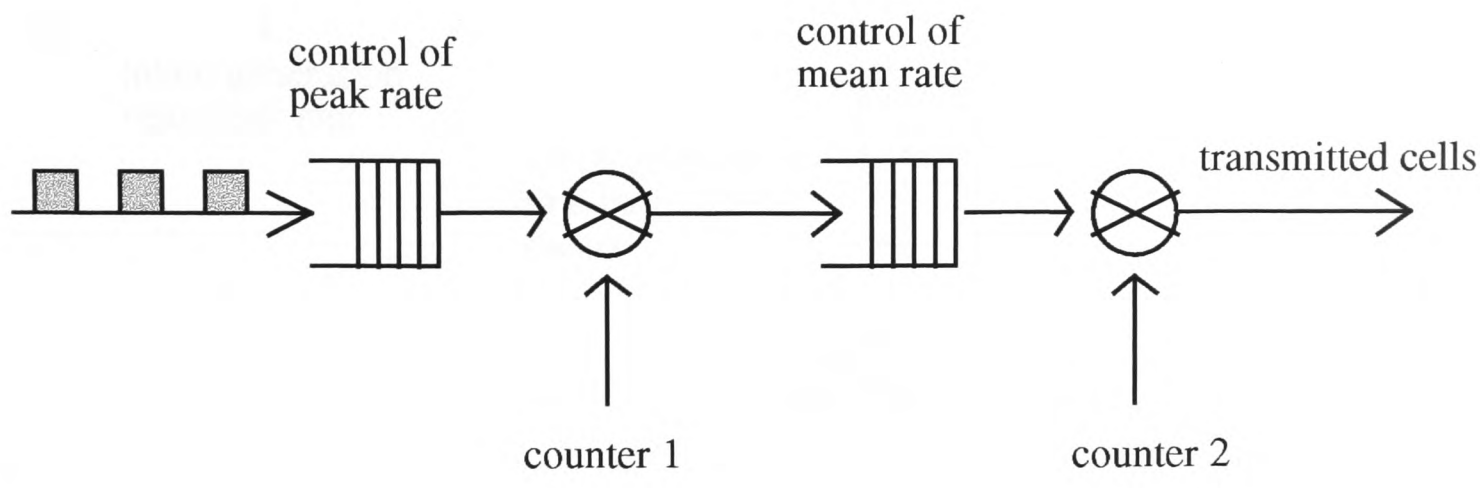



**Figure 2.24 Dual Leaky Bucket**

The performance of the triple leaky bucket was analysed [ORS95], see Figure 2.25. Control of the peak rate was achieved by setting the token generation rate close to the peak rate. The actual token generation rate chosen was (0.98 * peak rate), to eliminate the need for a buffer and to give a fast reaction time to large increases in the mean traffic rate. It was found that the mean rate was not effectively controlled by setting the token rate close to the mean rate, as cells arriving during a peak rate burst required a large buffer to prevent conforming cells from being discarded. Time sensitive traffic can not be delayed by using a smoothing buffer to space the cells. A long reaction time is needed to accurately control the mean rate, however, for fast reaction times a short estimation period is needed. The mean rate was controlled by using two token generators, which were dimensioned using a refresh rate higher than the mean cell arrival rate. This gives a slower reaction time, but provides tight control over the mean rate. The results show that the token generation rate needs to be slightly higher than the mean bit rate.

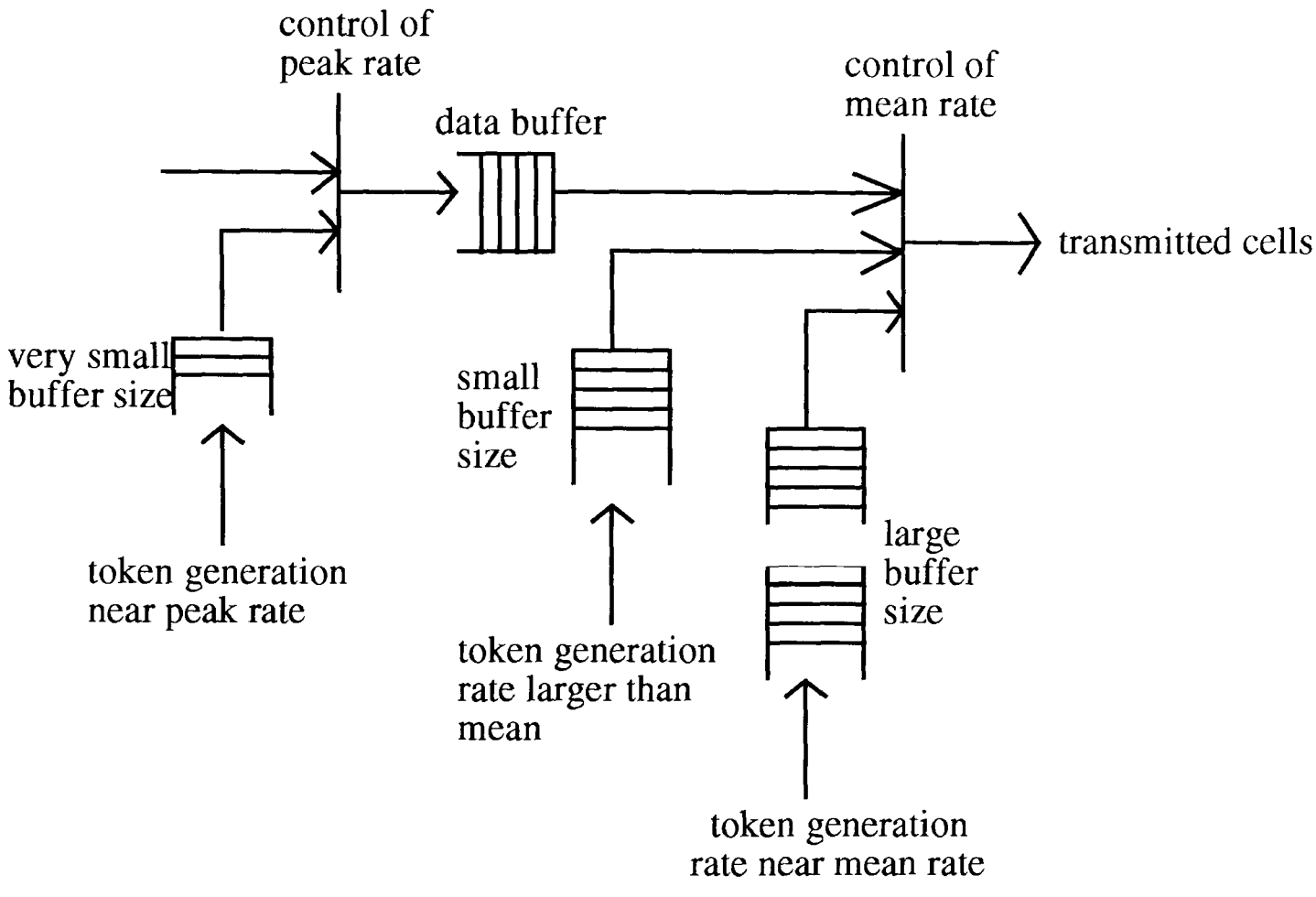



**Figure 2.25 Triple Leaky Buckets**

The configuration of multiple leaky buckets is very complex. Tight control over the mean rate or a fast reaction time to violations can be provided, but not both at the same time.

## 2.7 Summary

It is clear from the above discussion that congestion control in ATM networks is a very complex issue and is a subject of considerable interest. The next chapter defines the scope of the present work. The system model and details of the proposed leaky bucket are also presented.

# Chapter 3 - Present Work:
# System, Simulation Model and Implementation

The aim of this work is the investigation of the performance of an ATM network, mainly the UNI, with realistic multimedia traffic. At the present time there are few ATM networks around and measurements to determine the characteristics of the carried traffic are not possible. Analytical models often require many assumptions and may be too restrictive. These are not tractable for heterogeneous, multiple sources. Simulation modelling imposes fewer limitations and is a better choice for evaluating the performance of these networks and this is the approach which has been taken in this work.

## 3.1 The System Model

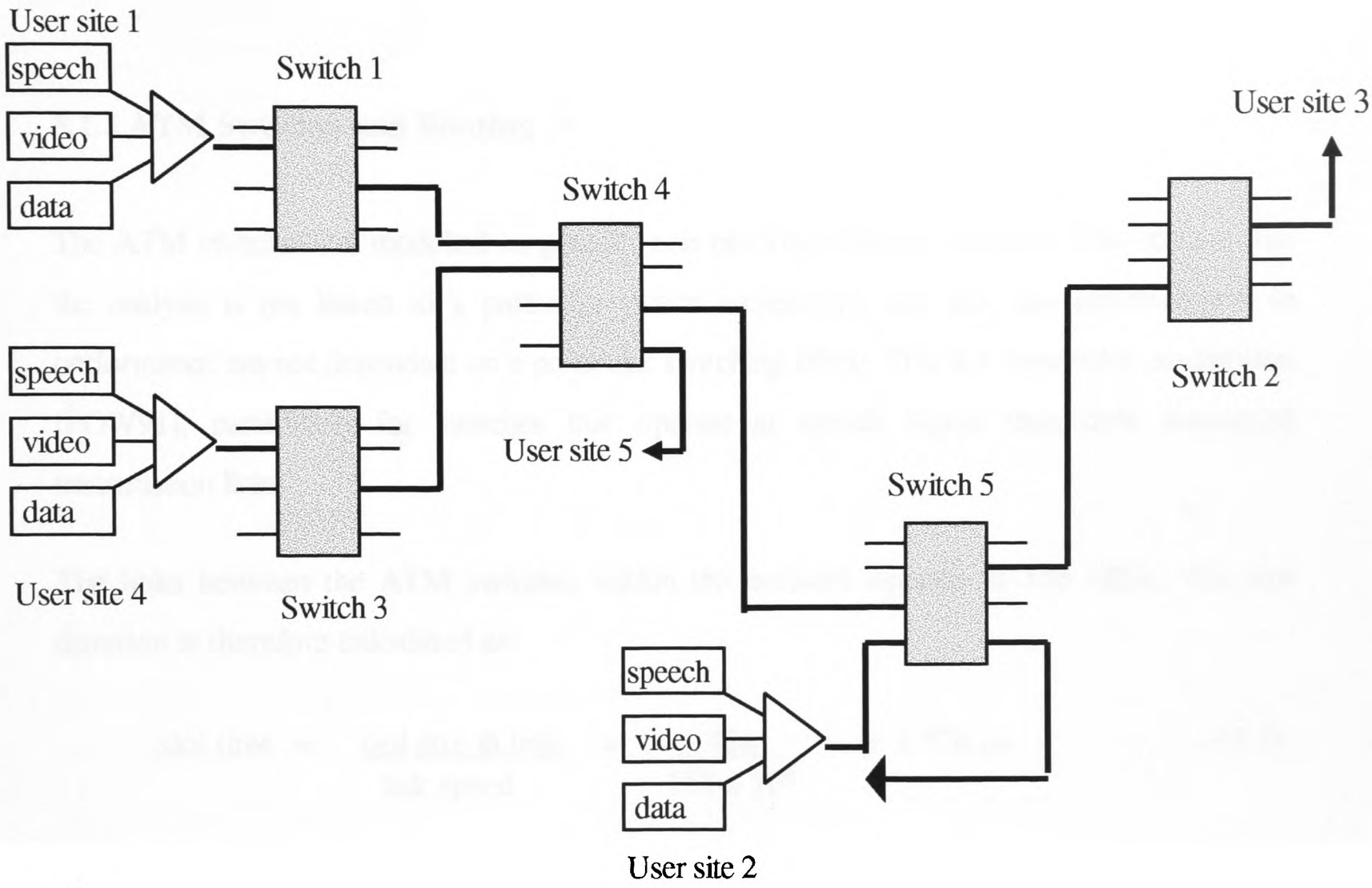
A prototype ATM network model was initially developed, which consisted of two ATM switches [GAN95]. This model was extended, after consultation with BT at Martlesham, to include five ATM switches. It was felt that the merging of the different traffic streams within the network was an important aspect of the simulation study.

The system modelled in this work is a five node ATM network with associated user sites. Each user site generates different types of traffic, which are multiplexed onto the network. The cells of the various traffic types cross the network to a destination user site. Performance statistics are collected at the cell level and where applicable at burst level. A diagram of the network is shown in Figure 3.1.

The layout of the network was designed to allow a large number of cells to pass through the maximum number of switches, in this case 4 switches. At each of the intermediate switches a percentage of the cells leave the network.

User sites 1 and 4 each access their own local ATM switch. These two traffic streams, destined for user sites 2 and 3, merge at switch 4, while the cells destined for user site 5

leave the network. At switch 5 the cells from user site 2 join the traffic stream and merge with the cell stream from user sites 1 and 4. These cells are all destined for user site 3, and only pass through two switches before reaching their destination. The performance of the cells from user sites 1 and 4 which are destined for user site 3, and the cells from user site 2, which are all destined for user site 3, are used to monitor the performance of the network




**Key**

= multiplexer

**Figure 3.1 ATM Network**

In this work the access link from the multiplexer to the ATM network transmits at 45 Mb/s, [FOW91 and TED93], while the network links operate at 155 Mb/s. All traffic gains access to the network through the multiplexer, which has a service time of 9.422 μs, determined by the size of the ATM cell and the speed of the access link. All time during the simulations is measured relative to the ATM slot duration, which is 2.7263 μs, see Table 3.1.

| | | |
|---|---|---|
| User Site Access Link | 45 Mb/s | |
| Network Links | 155 Mb/s | |
| ATM slot duration | $(424/155*10^6)$ | = 2.726 µs |
| Service time at ATM switch | 2.726 µs | ≡ 1 slot |
| Service time at the multiplexer | $(424/45*10^6)$ | = 9.42 µs (3.456 slots) |

**Table 3.1 Typical Values used by the Simulation Model**

### 3.1.1 ATM Switches and Routing

The ATM switches are modelled as generic, non-blocking Banyan switches. This ensures that the analysis is not linked to a particular switch architecture and that any improvements in performance are not dependant on a particular switching fabric. This is a reasonable assumption [FOW91], particularly for switches that operate at speeds higher than their associated transmission links.

The links between the ATM switches within the network operate at 155 Mb/s. The slot duration is therefore calculated as:

$$\text{slot time} = \frac{\text{cell size in bits}}{\text{link speed}} = \frac{424}{155 \times 10^6} = 2.726\ \mu s \qquad \text{----(3.1)}$$

Routing tables are used to route the cells through the network. The ATM standards specifying how routing should be determined are not yet complete. The ATM equipment manufacturers will have to design their own techniques at present [CLARK96]. Each switch in the network must have knowledge of the locations of the other switches to enable VC and VP connections to be set up.

Both routing tables (fixed and dynamic) used in this work are stored centrally and accessed by all switches, for economy of run-time memory. The network routing table (Table 3.2) allows each switch to have knowledge of it's immediate downstream neighbour, so the correct output port at the current switch and the correct input port at the destination switch are always used. It also provides knowledge of any attached user sites which may be

present. The network routing table used here is purely for programming convenience and makes no attempt to model an actual routing table which might be found in an ATM network. In reality each switch would have its own copy of the network routing table

| Table Entry | Function |
|---|---|
| switch | identifier for the switch accessing the table |
| attachedswitch | the number of the next switch in the network |
| OPport | the output port to use to pass the cell to the attached switch |
| IPport | the input port that the cell arrived at |
| usersite | an address for an attached user site (blank if there is no attached user site) |
| sitenumber | site number of an attached user site (0 if there is no attached user site) |

**Table 3.2 Network Routing Table**

A dynamic routing table, (Table 3.3) would also be present at each switch in an actual ATM network. The dynamic routing table allows VCs to be switched at high speed through the network by eliminating the need to match complex addresses in large routing tables. Within a particular switch in an ATM network, the entries are unique for each connection. Since the table in the simulation is global each entry must be unambiguous. The dynamic routing table is loaded at initialisation and does not change throughout the simulation. For a real network, the entries in the table would change as connections are set up and cleared.

The dynamic routing table contains the following fields:-

| Table entry | Function |
|---|---|
| Input port | the input port that the cell arrived at |
| Output port | the output port that the cell needs to be routed to |
| OldVCI | the current VCI label |
| NewVCI | the new VCI label to use |
| last switch | a boolean variable indicates when this is the last switch and then the cell must be passed to the user site manager |

**Table 3.3 Dynamic Routing Table**

Routing for ATM cells is carried out at each switch using the cell header information. A cell entering an ATM switch has its VCI label read. The VCI label is changed based on the information within the dynamic routing table held within the switch. The cell is then routed to the correct output port queue.

In this work, the VCI label is also changed as the cell enters the switch. The dynamic routing table is searched until a match is found for the current VCI label and the input port that the cell arrived at. The new VCI label is placed into the cell header and the cell is routed through the switch to the correct output port, as indicated by the dynamic routing table. If a match is not found, then the cell is assumed to be mis-routed and is counted and discarded.

At the output port the cell at the head of the queue has its VCI label matched to the entry in the dynamic routing table to ascertain if the cell is to be passed to the next switch, or to the destination site. If this is the last switch then the destination site number is found in the network routing table and the cell is passed to that site. If this is not the last switch, then the number of the next switch and the input port to use at that switch are obtained by matching an entry in the network routing table. The cell is then passed to the next switch.

Since both routing tables are global, each VCI label must be unique within the network and hence each entry in both tables must also be unique.

There is an associated fixed switching delay as the cell traverses the switching fabric and the VCI label is changed to the new value. The propagation delay between switches and the multiplexer and a switch are not included in the delay statistics. When the simulation model was under development there were no values available and since the delay is a constant for each link in the network it was felt that this omission was in no way detrimental to the overall results.

### 3.1.2 User Sites

The concept of a user site is used as the source for all the traffic entering the ATM network. Each user site generates traffic which passes through the policing function before being

queued at the multiplexer. The multiplexer then passes cells to the network for transmission to the destination user site. Traffic destined for a particular user site is received, from the network, and passed to the multiplexer at the destination user site.

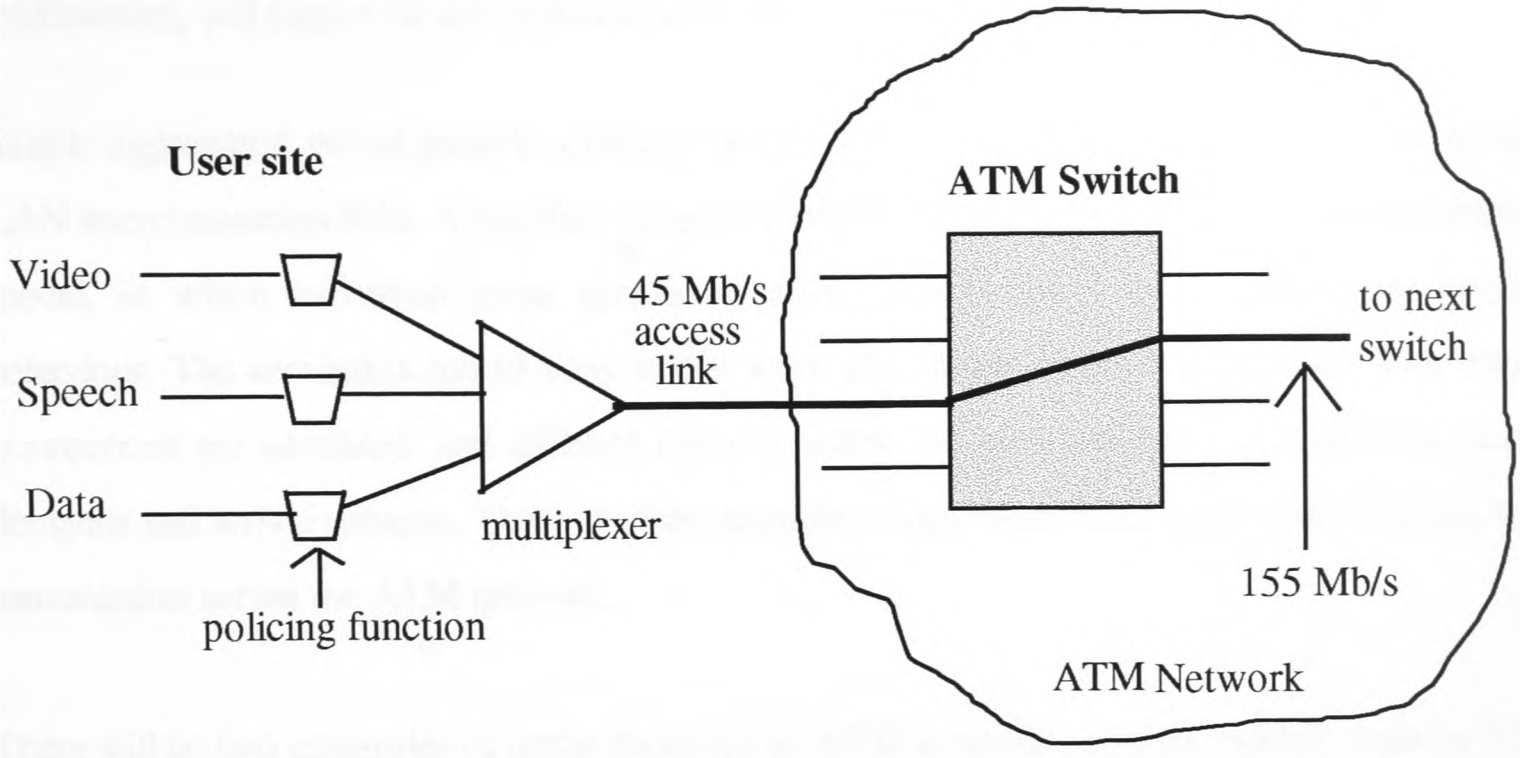



**Figure 3.2 A Typical User Site and Local Switch**

A user site typically generates speech, video and a variety of data traffic, as shown in Figure 3.2. A call is set up when a source becomes active and the connection is closed when it finishes. A call is accepted if the combined bit rate of the new connection and those already active does not exceed the total capacity of the access link.

All traffic must first pass through the policing function before gaining access to the multiplexer queue. Much previous work, as indicated in Section 2.3, has assumed that all traffic sources may be treated the same and are queued in a single FIFO buffer. This work is intended to show that traffic sources with different characteristics and arrival patterns achieve better QoS when treated with different policing and multiplexing strategies.

### 3.1.3 Traffic Sources

The traffic source models which have been included in this work, are an attempt to represent realistically the type of traffic that will ultimately be found on future ATM networks. The different traffic types have very different QoS requirements and it is expected that the interaction and interference which will occur between these traffic types, as they pass through switches and multiplexers, will impact on the performance of the network as a whole.

Simple aggregated arrival models seriously under estimate the peak demands found on actual LAN interconnection links. It has been suggested [FOW91] that a more sophisticated simulation model, in which individual users generated traffic, would show more realistic dispersion behaviour. The simulation model used in this work also incorporates this approach. Individual connections are simulated. and different types of traffic are generated each with different peak demands and arrival patterns. These are then multiplexed together into a single traffic stream for transmission across the ATM network.

There will be two categories of traffic found on an ATM network and these are RT and non-RT traffic. The sources of RT traffic are mainly speech and compressed video, while the non-RT traffic is data traffic. Since each of these categories has different loss requirements and time constraints, it is important to include instances of each in any study of network performance, particularly the performance as seen by the end-user.

#### 3.1.3.1 Speech

In this work, only one half of the telephone conversation has been modelled. Speech is modelled as a two state (ON / OFF) model. The silence and talkspurt periods are determined using an exponential distribution with mean values of 1.67 and 1.34 seconds, respectively, [BRAD65]. It has been found [BRAD69], that for speech, an exponential distribution fits the talkspurt well, but is a less good fit for silence periods. However, if more than 25 speech calls are multiplexed together, the approximation becomes closer to real traffic sources, according to [ONV94]. The duration of the call is generated using an exponential distribution, with a mean of 3 minutes, [RAMA91].

It is assumed that speech is coded using a frequency plane coding technique. Cells are coded with the most significant bits in a high priority cell and the least significant bits in another cell which is marked as a low priority. Two cells are generated one after another, with twice the packetisation delay for speech (12 ms).

The number of cells in a talkspurt (N) is calculated by multiplying the talkspurt duration (T) by the speech rate (64 Kb/s) and then dividing the result by the number of bits corresponding to the payload.

$$N = \frac{T * 64{,}000}{\text{cell payload}} \qquad \text{----(3.2)}$$

A small scale PBX is assumed at each user site. The maximum number of calls (100) corresponds to the total number of lines available at the PBX. Telephone calls start up at various times during the simulation. A new telephone call is accepted if there is a free line available. From the analysis of actual telephone calls, the mean inter-arrival time between calls in a PBX has been found to be 20 seconds. However, the mean inter-arrival time used in this work has been set to a much lower value (0.003 seconds), to ensure that the number of phone calls is maintained as close to the maximum as possible.

The request for bandwidth to the CAC function is half the coding rate for speech. This is a reasonable approximation, since it is known that telephone calls are made up of more than 50% silence [BRAD65].

### 3.1.3.2 Video

Video is a relatively new RT traffic source, previously restricted to low bit rate video-phones, which produced poor quality pictures due to bandwidth limitations imposed by transmission across telephone lines. It is expected that video traffic will become more wide-spread as applications are implemented to take advantage of the increased bandwidth which will become available with the introduction of broadband networks. With the introduction of home video on demand, video could become the main traffic on ATM networks.

In the work done by [IZQUI94], on the statistical characterisation of MPEG video, an 8 state Markov Chain was successfully used to obtain a close approximation to the actual output generated by an MPEG encoder. The most important characteristic of MPEG video coded as ATM cells is the number of cells per slice. This will vary from 0 to some peak value, P over the duration of the video clip. By quantizing the range 0 to P into 8 equal size ranges, and assigning each to a distinct state of Markov chain, one can construct a tractable mathematical model. The Markov Chain transition probability matrix was obtained from the particular video clip analysed, as each clip had different characteristics and generated different patterns of I, B and P frames. This meant that the method used was specific to the video clip under consideration.

The 8 state model was adopted here with modifications, to model the bursts of video data which result from sudden scene changes, or violent action within scenes (worst case scenario). High action and many scene changes generate large bursts of cells at the peak rate. For this reason the video model has been biased in favour of the peak bit rate, to explicitly model this behaviour. The peak rate (P) is a model parameter. The 8 states (j = 1, 2, ..8) are characterised by bit rates $r_j$ given by :-

$$r_j = \frac{P}{j} \qquad \text{----(3.3)}$$

$$\text{mean bit rate} = m = {}^{1}/_{s} \sum_{j=1}^{s} r_j \qquad \text{----(3.4)}$$

Where (s) is the total number of states.

The mean burst size (b) is calculated by assuming that the codec outputs at the mean bit rate (m) for the duration of the video refresh period of 30 frames per second (0.0333 seconds) with a cell payload of 44 octets.

$$\text{burst size} = b = \frac{m * \text{frame refresh period}}{\text{cell pay load}} \qquad \text{-----(3.5)}$$

$$= \frac{(m * 0.0333)}{(44 * 8)} \qquad \text{-----(3.6)}$$

Bursts of video cells are generated using an exponential distribution with the mean burst length as given by equation (3.6).

The state of each new burst of video cells is randomly chosen (between 1 and 8), at the start of the burst. The state, say j, is used to determine the current bit rate of the burst. A state of 1 causes a burst to be generated at the peak rate. The current bit rate (r) is easily calculated using (3.3).

The packetisation delay, in slots, can then be calculated for the current bit rate (r), using the network speed (155.52 Mb/s), the cell payload of 44 octets and a cell size of 53 octets.

$$\text{pd (in slots)} = \frac{\text{(cell payload)}}{\text{(r * slot duration)}} \qquad \text{------(3.7)}$$

Using the previously calculated mean video rate (m) from (3.3), a new rate is calculated called the effective rate, which lies between the peak and mean rates. The effective rate is used to request bandwidth from the CAC function and to dimension the leaky bucket.

$$\text{effective rate} = m * ( 1 - \text{Log}_{10} (m / P) ) \qquad \text{-----(3.8)}$$

This allows an over-dimensioning factor [ARV95] to be introduced to the mean bit rate, since the mean rate alone is not adequate for the CAC function. Allocating at the mean rate could cause the link to become over-loaded, with a resulting buffer over-flow and loss of cells. Allocating at the peak rate, for VBR sources makes inefficient use of bandwidth.

As stated in Section 2.3.3.1, each picture is divided into a number of slices. Each slice consists of a slice header and one or more macro blocks. The number of slices per frame is variable for VBR video [IZQUI96] and the peak number of ATM cells per slice varies from 40 to 112, for the video clips analysed and up to 150. Since the slice layer is the lowest independent data unit for MPEG video, it seems reasonable to use this as the

resynchronisation point for the stream of ATM cells. By discarding a single ATM cell within a slice, there is a high probability that the following cells within that slice will be discarded on arrival at the destination as useless, at the very least. They may also cause the codec to desynchronise with resulting errors within the decoded picture, which can persist for quite some time. Once a cell is discarded the cells from the rest of that slice are also dropped. The start of the next slice terminates the clipping function.

Each burst is divided up into a number of slices. The number of cells in a slice is determined by selecting a random integer between 5 and 100. The start of each slice has a bit set in the GFC field of the cell header, to indicate that it is the first cell of a new slice. If the tail end clipping (TEC) function has been activated at the leaky bucket, then the arrival of the first cell of a new slice, will cause it to reset and stop discarding cells.

### 3.1.3.3 Data

Data traffic has very strict loss requirements, but is less sensitive to delay than the RT traffic previously mentioned. A single, high level protocol data unit (PDU) may require many ATM cells to transport it. It has been reported by [FOW91], that a variable length packet PDU could require up to 210 cells to transport it across an ATM network. The loss of even a single cell would require the whole PDU to be re-transmitted, as the cells arriving at the destination would be discarded on arrival. If the cell loss was caused by congestion, then re-transmitting the PDU could exacerbate the problem.

The data sources used in this work are all considered non-RT and constant bit rate traffic sources. There are five different data types included in the model [GAN94]. These are voice-band data, videotext/teletex, telemetry, facsimile and transaction time sharing. Each one has a minimum and maximum bit rate and time duration. The mean bit rate and the connection time durations are selected from within the range indicated by the data type (Table 3.4) [WEINS87].

| | | Bit Rate | | | Time Duration | |
|---|---|---|---|---|---|---|
| | Source | Min | Max | | Min | Max |
| data 1 | Voiceband data | 300 b/s | 30 Kb/s | | 55 sec | 3000 sec |
| data 2 | Videotext/teletex | 600 b/s | 90 Kb/s | | 600 sec | 2000 sec |
| data 3 | Telemetry | 2 b/s | 200 b/s | | 1 sec | 60 sec |
| | | | | | | |
| data 4 | Facsimile * | 3 Kb/s | 3 Mb/s | | 3 sec | 1000 sec |
| data 5 | Transaction Time Sharing * | 60 b/s | 6 Kb/s | | 20 sec | 3600 sec |

* the minimum and maximum time duration's are not linear - as the bit rate increases, the time duration decreases

**Table 3.4 Data Traffic Parameters**

Data traffic is modelled as data messages. A time duration and a bit rate are chosen at random within the appropriate limits.

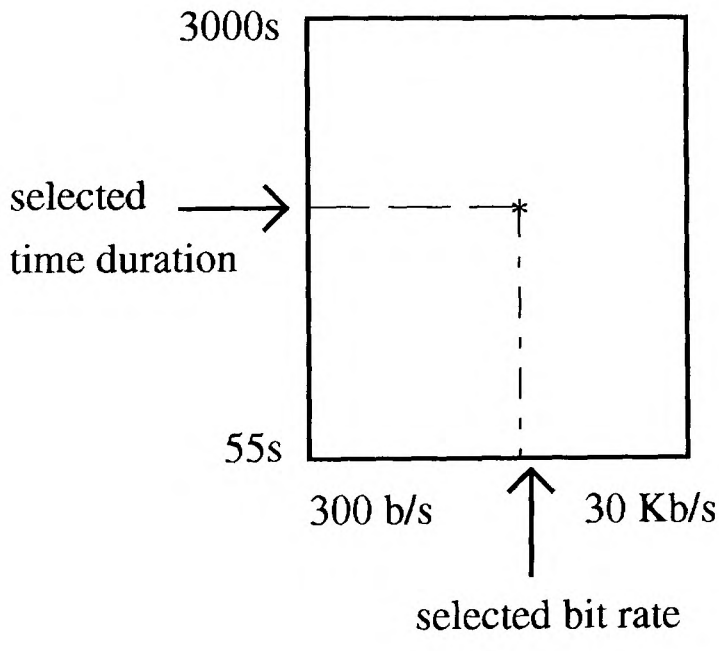



**Figure 3.3 Voiceband Data Model**

The data model for voiceband data is shown in Figure 3.3. The minimum time duration for this model is 55 seconds and the maximum time duration is 3000 seconds. The range of bit rates is between 300 b/s and 30 Kb/s.

The transaction time sharing data model is shown in Figure 3.4. For this model, the bit rate is randomly selected from within the range (60 b/s to 6 Kb/s). The minimum time duration is fixed at 20 seconds, but the maximum time duration depends on the bit rate selected. The maximum bit rate is calculated, according to the bit rate, and a time duration is then randomly selected between the minimum and maximum times. This gives the bit rate and the time duration for this call.

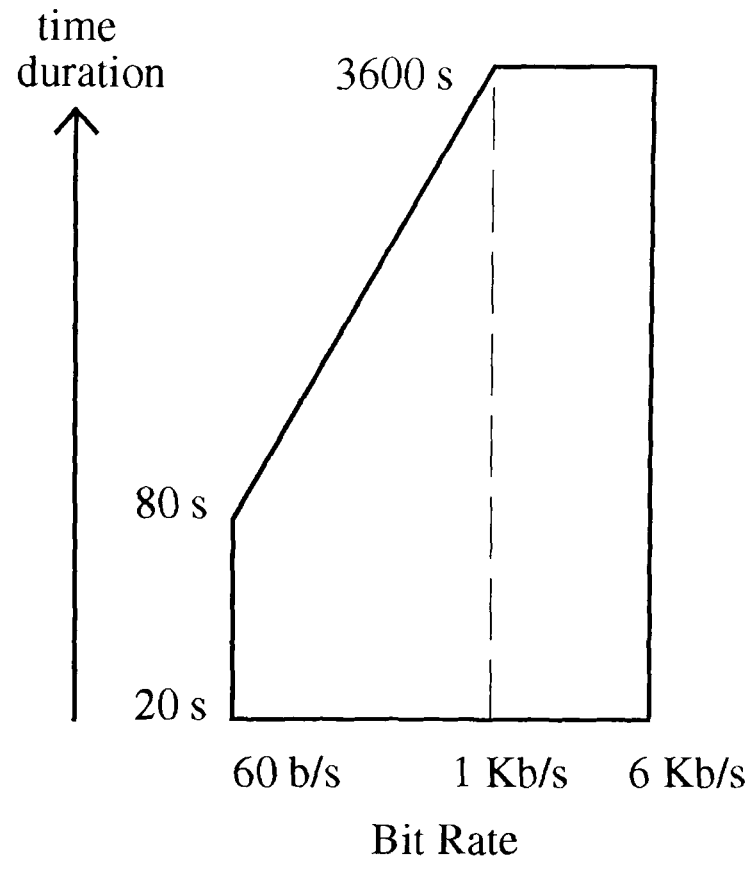



**Figure 3.4 Transaction Time-Sharing Data Model**

The facsimile data model is shown in Figure 3.5. As the bit rate increases, so the time duration range decreases. The bit rate is randomly selected from with the range indicated. The minimum and maximum time, in seconds, is then calculated, which corresponds to that bit rate. A time duration for the data connection is then randomly selected between the minimum and maximum.

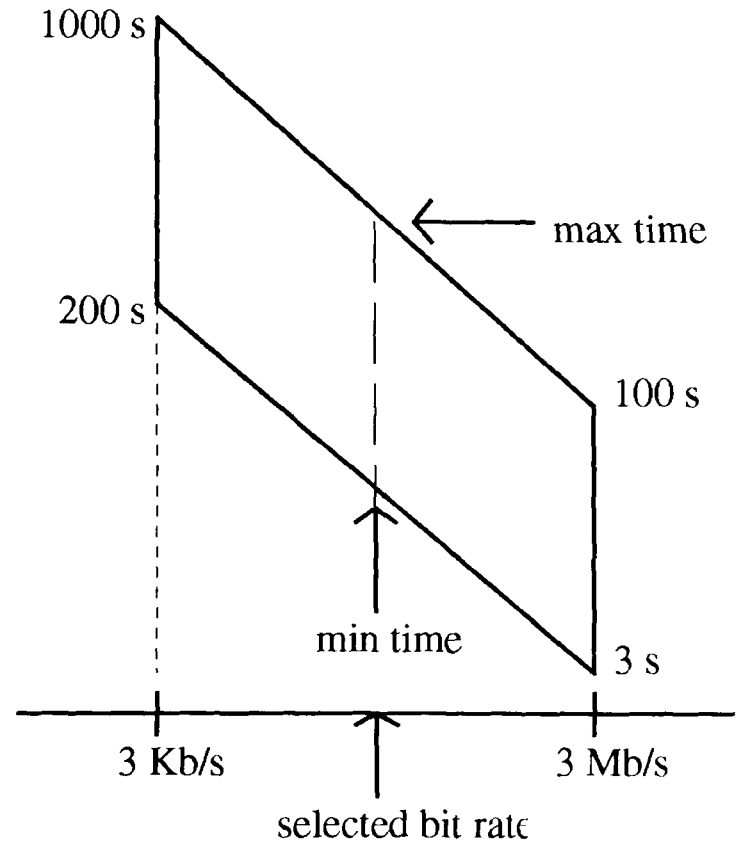



**Figure 3.5 Facsimile Data Model**

Given the time duration (t) and the bit rate (r), the file size (f), in cells, can be calculated.

$$f = \frac{(r * t)}{\text{cellpayload}} \qquad \text{-----(3.9)}$$

The time duration is converted to slotted time by dividing it by the length of an ATM slot (2.7μs).

$$t_{slotted} = \frac{t}{2.7E-6} \qquad \text{-----(3.10)}$$

The packetisation delay for each data cell is then calculated.

$$pd = \frac{t_{slotted}}{f} \qquad \text{-----(3.11)}$$

Since data traffic is a constant bit rate source, the packetisation delay which is calculated in (3.11) is used to space out the individual cells appropriately, see Figure 3.6.

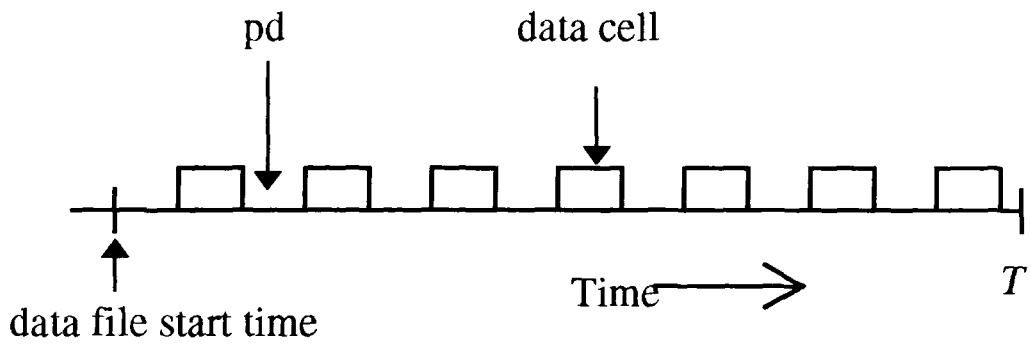



**Figure 3.6 Packetisation Delay for Data Sources**

### 3.1.4 Multiplexer

The performance of the multiplexer depends on the service strategy and the queuing method used. A single queue with a FIFO service strategy will cause significant delays to RT cells.

especially if they join an already large queue of cells. It is acknowledged that other service strategies (oldest cell first, MLT, QLT) can reduce the delays experienced by individual cells, but at the expense of increased complexity at the multiplexer. Using separate queues for the two types of traffic with cyclic service is an alternative.

Giving the RT cells some degree of priority significantly reduces the delays experienced by these cells. Using two thresholds to monitor the RT queue, dynamic queue management can be achieved. When the RT queue length exceeds the first threshold (RT1), the server switches over from cyclic service and gives non-pre-emptive priority service to the RT queue, until the queue length falls below the threshold. Previous work, [GAN95], has indicated that exhaustive priority service (i.e. serve the priority queue until it is empty) for the RT queue, has no significant impact on the delays experienced by RT cells, and so it is not included in this work. When the queue length falls below the $RT_1$ threshold, the server returns to cyclic service.

If the queue length exceeds the second threshold (RT2), then all low priority cells removed from the queue are discarded, until the queue length again falls below the threshold. The low priority cells will be mainly speech cells which have been labelled as low priority by the speech coder. This enables the server to effectively serve at least two cells at each service time and significantly reduce the delays to high priority RT cells.

There are also two thresholds associated with the non-RT queue at the multiplexer (T1 and $T_2$), which are used to prevent the buffer in the multiplexer from overflowing, by throttling back the data traffic sources. If the multiplexer non-RT queue reaches the first threshold $T_1$, this causes the leaky bucket to reduce the flow of cells to the multiplexer, by increasing the service time at the buffer. When the threshold reaches the second threshold ($T_2$), any non-RT cells arriving at the multiplexer are blocked at the source and is assumed to be re-transmitted.

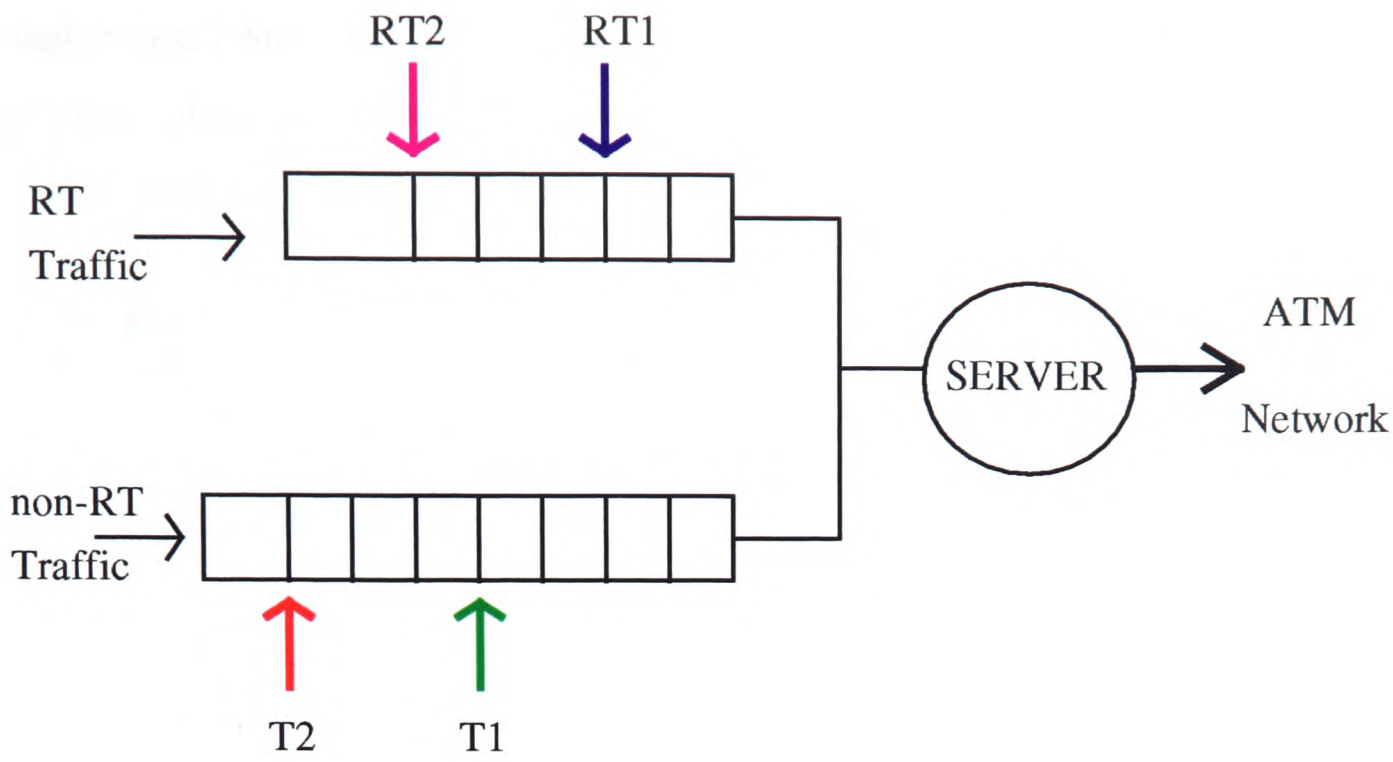



**Figure 3.7 Cyclic Server at Multiplexer with Thresholds**

A zero switch-over time for the cyclic server is assumed. A diagram of the cyclic server is shown in Figure 3.7, indicating the thresholds (not to scale).

### 3.1.5 Algorithms

This work is based on the algorithms (leaky bucket and avalanche tagging) proposed by Niestegge [NIEST90], and Di Nitto et al. [NITTO92].

#### Dimensioning the Algorithms

The parameters of the leaky bucket are negotiated when a call is set up. As cells pass through the leaky bucket a counter is incremented. When the counter reaches a threshold value, any subsequent cells will be either discarded or tagged. The counter is periodically decremented and cells which arrive at a steady rate will pass through without penalty. It is the dimensioning of the parameters which is critical to the operation of the leaky bucket.

The algorithm proposed includes a factor ($\delta$) which allows for CDV within the cell stream. [NIEST90] estimated that $\delta$ is approximately 70 service times when the utilisation of the server is 85%. This gives a value of $\delta = 70 * 2.726\ \mu s \approx 0.20$ ms, which is assumed for links with

rates higher than 1 Mb/s. For links with bit rates less than 1 Mb/s then $\delta$ = 20 ms is used. These are the values which have been use in this work. For the lower rate (< 1 Mb/s) a decrement value of 1 is used, while for the higher bit rates a decrement step of 16 is used.

Let S = threshold for the leaky bucket counter
d = decrement level for the counter
T = refresh interval (how frequently the counter is decremented)
r = bit rate of the source to be policed
b = mean burst length (in cells)

The bit rate is used to determine the decrement period for each connection as it starts up and this is calculated as follows:-

$$T = (d / r) * \text{cell payload} \qquad \text{----(3.12)}$$

The threshold for the leaky bucket to start tagging or discarding cells is found using the formula:-

$$S = 1 + d + (\delta * r) / \text{cell payload} \qquad \text{----(3.13)}$$

A threshold (S), a decrement (d) and a decrement frequency (1/T) are all negotiated at set-up. The counter (Z) is initialised to 0.

1) A cell arrives - if counter (Z) is below the threshold then the counter is incremented
else the counter remains unchanged
INC counter: (Z := Z + 1 if Z < S)
do not INC counter: (Z := Z if Z = S) - discard cell

2) At periodic intervals of T (1/T is decrement frequency), the count is decremented by (d) or is set to zero

$$Z := \max. \{Z - d, 0\}$$

The leaky bucket allows the following bit rate :-

$$W = (d/T) * \text{data field} \qquad \text{bits / sec} \qquad \text{----(3.14)}$$

(data field = 48 * 8)

The user can transmit at W bits per second, and no higher, or the counter will quickly reach the threshold value (S). If this rate is exceeded, the remaining cells arriving would be discarded. The parameters d and T cannot be independent of one another.

A decrement of one is adequate for lower bit rate connections such as speech. By selecting a decrement greater than one, the length of the decrement period (T) is increased and the algorithm operates more slowly which may be advantageous for high bit rate connections.

The equations 3.12 to 3.14 have been used to dimension the leaky bucket for data and speech. For video, the mean and twice the mean burst size has been used for the decrement value (d). Avalanche tagging ([NITTO92], see Section 2.6.2.3) has been used for video traffic, to compact the number of tagged cells into the smallest number of data units. However, violating cells are dropped rather than being tagged.

### 3.1.6 Parameters

There are a number of parameters which have been used throughout the simulations. Section 3.1.6.1 shows the parameters that are fixed for all the simulations, while section 3.1.6.2 lists the parameters that vary over different runs.

The fixed parameters are either defined as constants or are coded into the simulation model, and so are the same for all the simulations. The variable parameters are loaded from a batch file at run time.

### 3.1.6.1 Fixed Parameters

- The ATM Network Parameters

| Constant Name | Function |
|---|---|
| capacity | the capacity of the network links (155.52 Mb/s) |
| slot | a slot duration depends on transmission speed of the line |
| switchingtime | the time to switch each cell across the switching fabric is one slot duratic (2.7 μs) |
| servicetime | the service time for each cell is also one slot duration |
| cellcapacity | the capacity of the network links in cells per second (365,566 cells / sec) |
| cellsize | the size of an ATM cell in bits (424 bits) |
| cellpayload | the size of an ATM cell payload (384 bits) |
| **Parameter Name** | **Function** |
| numofswitches | the number of switches present in the ATM network |
| numberofports | the number of ports at each switch |
| dynamicroutingtable | array containing the dynamic routing information |
| networkroutingtable | array containing the network routing information |

- The User Site Parameters

| **Constant Name** | **Function** |
|---|---|
| linkcapacity | capacity of the access link to the ATM network (45 Mb/s) |
| linkinslots | capacity of the access link in slots ( 106,132 cells / sec) |
| capacity | the capacity of the network links (155.52 Mb/s) |
| capcityinslots | the capacity of the network links in slots (365,566 cells / sec) |
| slot | duration of an ATM cell slot on the network links (2.7 μs) |
| aslot | absolute time - reference slot (1.0) for the simulation |
| serviceTime | the service time at the multiplexer depends on the access link speed and is measured as a number of ATM slots (3.456 slots) |
| cellsize | size of standard ATM cell (53 octets) |
| speech coding rate | 64Kb/s |
| Talkspurt | the duration of the mean talkspurt period (1.34 seconds) |
| Silence | the duration of the mean silence period (1.67 seconds) |
| phonecallduration | the mean telephone call duration (3 minutes) |
| speechpayload | speech payload (47 octets) |
| speechpacketisation | time taken to fill one ATM cell with coded speech (8ms) |
| videocellpayload | video payload (44 octets, 352 bits) |
| **Parameter Name** | **Function** |
| numberofsites | the total number of user sites |
| numberofswitches | the number of switches in the ATM network |
| realtimeQ | the identifier for the array index for the RT queue |
| outgoingMUXQ | the identifier for the array index for the non-RT queue |
| incomingMUXQ | the identifier for the array index for incoming cells |

### 3.1.6.2 Variable Parameters

Variable parameters are input at run time from a batch file.

| Parameters for Main Module | Function |
|---|---|
| runlen | length of simulation run (200 seconds) |
| report | reporting interval (20 seconds) |
| randomseed | random seed used by all modules and objects for a single run |
| $RT_1$ | queue-length threshold for RT priority service |
| $RT_2$ | queue-length threshold for RT LP cell discarding |
| $T_1$ | queue-length threshold for non-RT throttle-back |
| $T_2$ | queue-length threshold for blocking non-RT cells |
| **Parameters for each user site** | **Function** |
| maxvideosources | maximum number of video sources allowed |
| peakrate | peak bit rate for video sources |
| maxNodatafiles | maximum number of data sources allowed |
| lam | inter-arrival time between data files |
| maxallowedcalls | maximum number of speech sources allowed |
| lam | inter-arrival time between phone calls (re-used) |

## 3.2 Description of the Simulation Model

The simulation model used for these experiments has been built using an object oriented, event driven simulation language called MODSIM (CACI Inc.). The object oriented approach to modelling the system components is ideally suited for this purpose. Traffic sources, multiplexers and ATM switches are all modelled as objects which have clearly defined interface definitions, that are accessible to other objects and internal implementation details, which are not. Objects have two types of methods; "tell" methods, which run asynchronously to the calling method, and "ask" methods which are similar to procedures or functions in other programming languages.

The simulation runs in slotted time, i.e. a unit of time is equivalent to the service time for an ATM cell on a link operating at 155.52 Mb/s, which has a duration of 2.726 μs. All time is then measured relative to this time frame.

The simulation model is written in a modular format. The Main module runs first and calls the other modules to begin the simulation. The other modules are the usersite module, the atmnw module and the types module, which holds the definitions that are required by the other two modules, see Figure 3.8.

The usersite module contains the user site manager object, which generates an array of user site objects. Each user site object has traffic sources associated with it, which generate the ATM cells, and also a multiplexer to send to and receive ATM cells from the ATM network.

The atmnw module contains an ATM network object, which creates an array of ATM switch objects, which form the ATM network. ATM cells are passed from the usersite module to the atmnw module. They then cross the ATM network and are passed back to the destination user site in the usersite module.

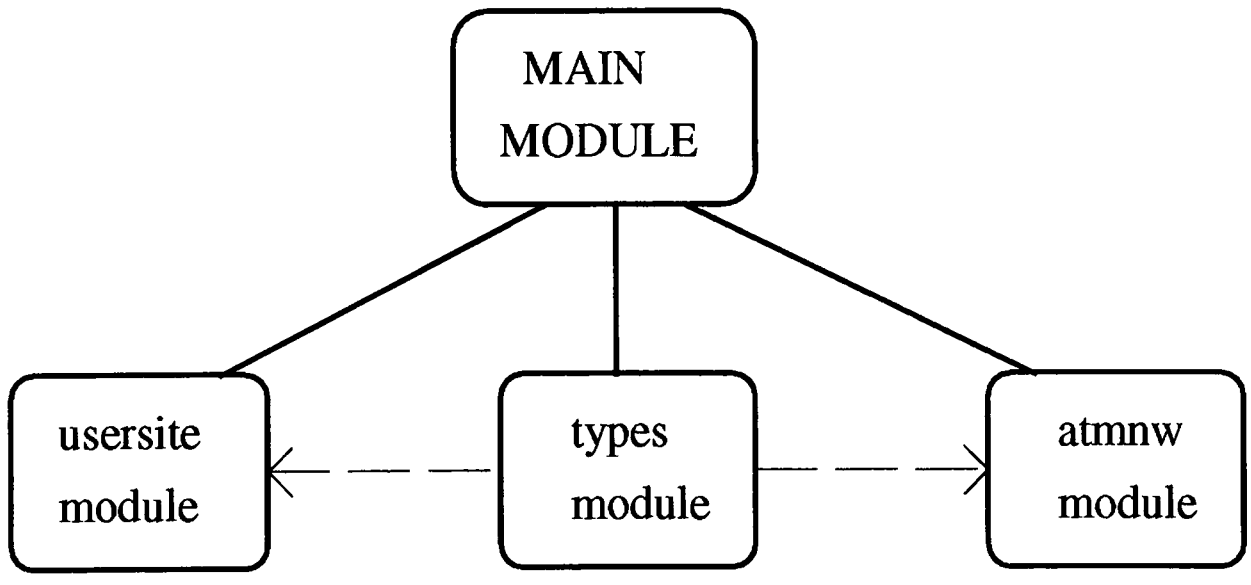



**Figure 3.8 Simulation Program Modules**

The types module does not contain any objects and is only present for the other modules to reference the definition of an ATM cell and the type definitions for the various fields within the ATM cell. The following definitions are found in the types module.

| Type Definition | Function |
|---|---|
| sourceType | identifies the originating source type - can be either video, speech, data, data1, data2, data3, data4 or data5 |
| cell | ATM cell type declaration - defined as a record with the following fields |
| msgstartTime | start of the message time stamp e.g. start of phone call |
| cellstartTime | start time for each individual cell - used to calculate delays |
| source | sourceType - video, speech, data etc. - used to identify the different types of cells |
| VCIlabel | the address label used for routing purposes |
| origin | site number of the sender |
| CLP | cell priority indicator (0 = high priority, 1 = low priority) |
| GFCfield | user defined flow control field at the UNI - used to reset the TEC function |
| VCid | identifier for the virtual channel in use - e.g. if the source is speech, then it contains the line number used by the current call |
| cellcount | holds the cell number currently being output - used for testing purposes |
| prev, next | pointers to previous and next records |

**Table 3.5 ATM Cell Definition**

### 3.2.1 Objects and Relationships

Each module has a number of objects associated with it and these are detailed in the following sections 3.2.1.1 and 3.2.1.2.

#### 3.2.1.1 User Site Module

A user site manager creates and initialises an array of user site objects. During initialisation, the user site manager reads the global run time parameters from an input file. These are, the random seed to use and thresholds values for the multiplexer queues.

A user site manager generates and initialises the required number of user site objects. All the site addresses which will be required by the user sites are also initialised by this object. The number of user sites and the random number seed are input at run time, see Figure 3.9, below.

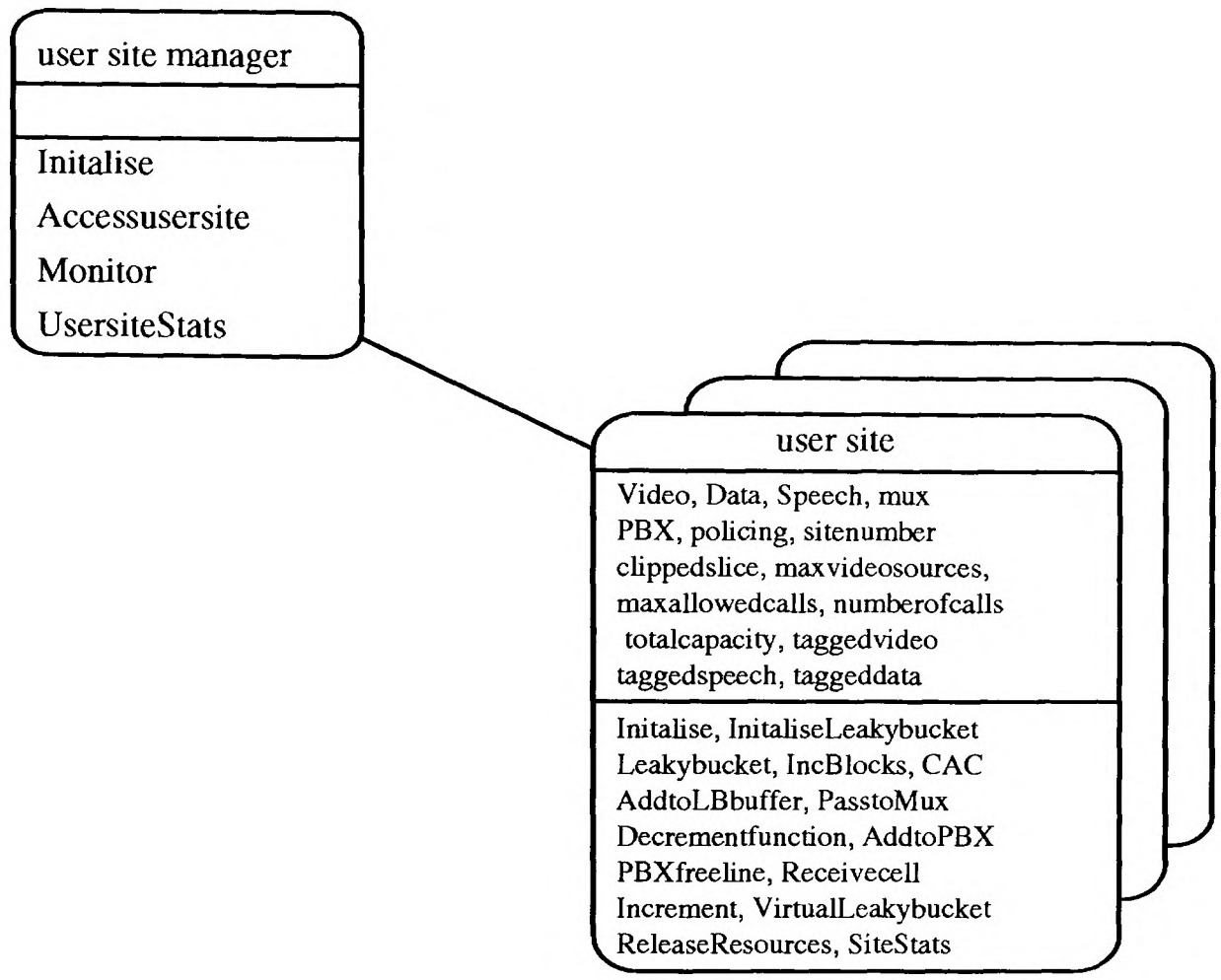



**Figure 3.9 User Site Module Overview**

### User Site Manager Object

| Object | Function |
|---|---|
| User site manager object | Creates an array of user sites. Receives cells from the network and passes them to the correct user site's multiplexer. |

**User Site Object**

| Object | Field Name | Field Definition |
|---|---|---|
| User site object | Video (object) | instance of a video object - one per user site |
| | Speech (object) | a speech object - one per user site |
| | Data (object) | a data object - one per user site |
| | mux (object) | a multiplexer object - one per user site |
| | PBX (array of objects) | an array for phone call objects - one per user site |
| | sitenumber | the number of this site |
| | numberofcalls | the number of phone calls currently in progress |
| | maxallowedcalls | the maximum number of phone calls allowed |
| | totalcapacity | the link capacity currently in use |
| | policing | the array for holding policing parameters for each connection |
| | taggingvideo | collecting statistics on video cells tagged by the leaky bucket |
| | taggingspeech | speech cells tagged by the leaky bucket |
| | taggingdata | data cells tagged by the leaky bucket |
| | speechTotal | counter for the total number of speech cells generated |
| | videoTotal | counter for the total number of video cells generated |
| | dataXTotal * | counters for each of the different types of data cells generated |
| | * where (X = 1 to 6) | |

**Function of Object**

The traffic source objects are created by those user sites which generate traffic. The speech object also creates a PBX, through which all telephone calls at that site are allocated output lines. Not all user sites generate ATM cells, but all user sites have an associated multiplexer object

At each site, the traffic sources are created which generate ATM cells. The current allocated link capacity is stored in totalcapacity. The call is allowed if the combined capacity of the current calls plus the new call, is less than the total link capacity, otherwise the call is blocked. The leaky bucket polices each source and discards or tags cells which are not conforming. The conforming cells are passed to the ATM network via the multiplexer.

The statistical variables that are used to collect information for analysis are initialised at each user site.

**User Site Object - Associated Objects**

Each user site object has the following objects associated with it (see also Figure 3.10):-

| Object | Function |
|---|---|
| Source object | Generic traffic source object, used to initialise the individual traffic source objects. |
| Video object | Video cells are generated by each of the independent video sources. |
| Speech object | Telephone calls are generated through a private branch exchange (PBX). |
| Phonecall object | Generates speech traffic through a PBX, using telephone line numbers. |
| Data object | Generates data traffic. |
| Multiplexer object | Receives ATM cells from all traffic types and combines them onto a single link to access the ATM network. |

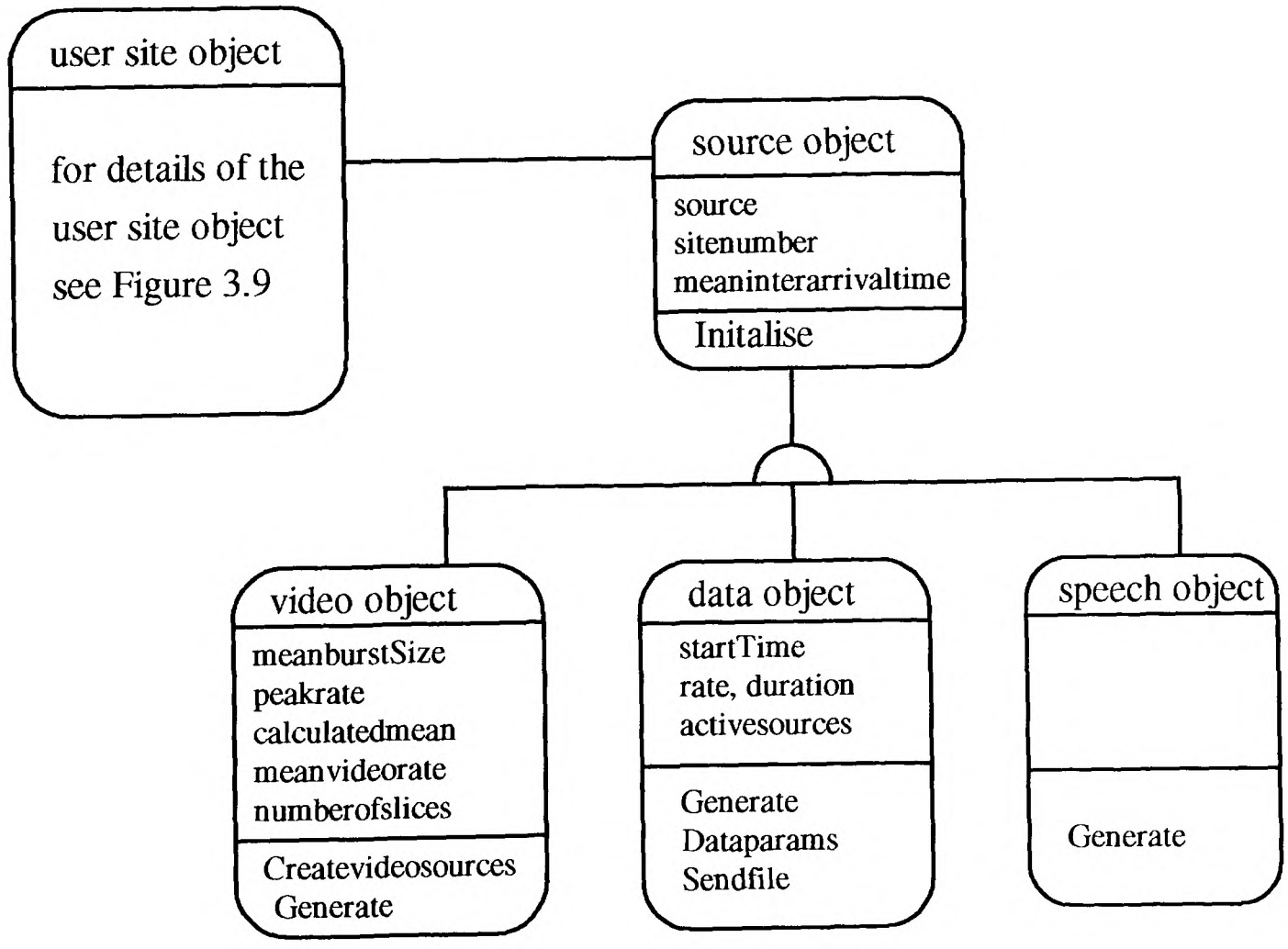



**Figure 3.10 User Site Associated Traffic Objects**

**Source Object**

| Object | Field Name | Field Definition |
|---|---|---|
| SourceObj | source | The type of traffic source for this instance of the source object. |
| | site number | The number of the associated user site. |
| | meaninterarrivaltime | The mean inter-arrival time for this source. |
| **Overview of Object Function** | | |
| The different traffic source objects all inherit a common source object. The source type, the site number and the mean inter-arrival time between calls are common to all, while the variables required by the different types are initialised in the individual instances. | | |

An instance of a source object can be either a video object, a speech object or a data object - see Figure 3.10.

**Speech Object**

The speech object inherits the source object's fields and the method initialise. It also creates the phone call object, see Figure 3.11

| Object | Field Name | Field Definition |
|---|---|---|
| speechsourceObj | | |
| **Overview of Object Function** | | |
| Maintains the number of phone call objects for the duration of the simulation by creating a new one each time a call ends. A new phone call object is created for each new telephone call that is made, providing a free line can be found in the PBX. | | |

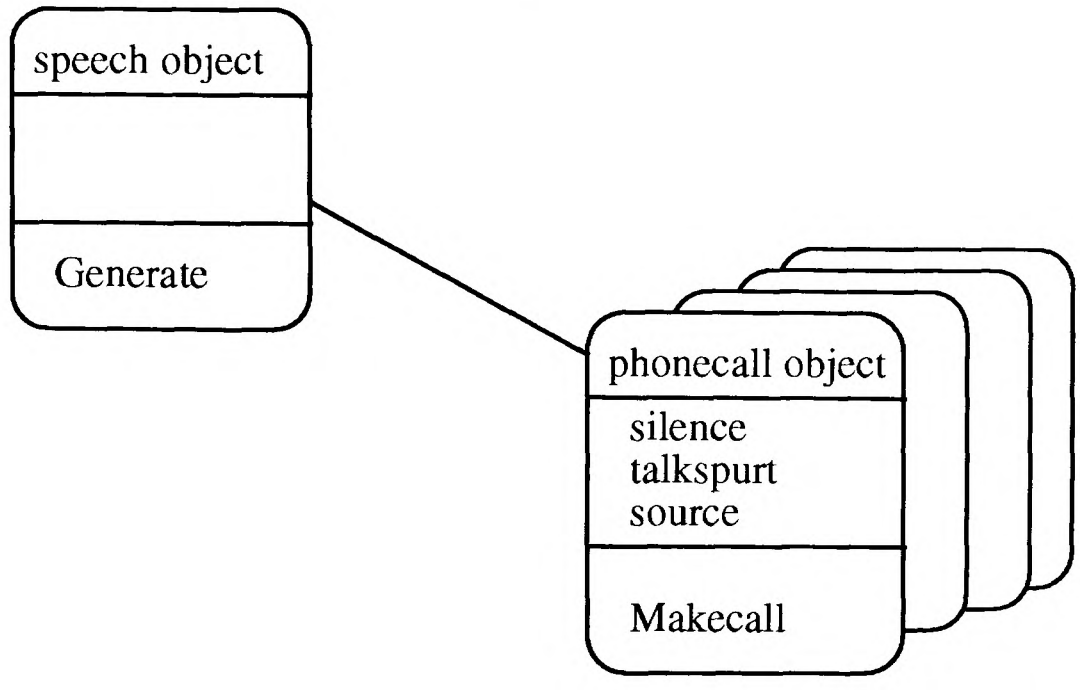



**Figure 3.11 Speech Traffic Source Object and Associated Phone call Objects**

**Phone Call Object**

| Object | Field Name | Function |
|---|---|---|
| phonecallObj | silence | Stores the length of the current silence period in slots |
| | talkspurt | Stores the length of the current talkspurt, in seconds. |
| | source | The source identifier for this object is "speech". |

**Overview of Object Function**

Phonecall objects are generated by the speech source object.

The length of each phone call is determined using an exponential distribution, with a mean of 3 minutes. A destination site is chosen at random and the first VCI address label corresponding to that destination is obtained from the address book.

Each call begins with a silence period and then talkspurt and silence periods alternate for the duration of the call. The duration of a talkspurt is chosen using an exponential distribution, with a mean of 1.34 seconds. The silence period is also selected from an exponential distribution, but with a mean duration of 1.67 seconds. Speech cells are only output during a talkspurt.

**Video Object**

The video object inherits the source object's fields and the method initialise.

| Object | Field Name | Field Definition |
|---|---|---|
| videosourceObj | meanburstSize | The mean burst size is calculated using the peak bit rate. |
| | peakrate | The peak rate for this instance of a video source is input from a batch file at run time. |
| | calculatedmean | The effective bit rate is calculated at run time, which lies between the peak and the mean rates, see Section 3.1.3.2. |
| | meanvideorate | The actual mean rate during the simulation is stored. |
| **Overview of Object Function** | | |
| When the video object is initialised, the required number of video sources and the peak bit rate for those sources are passed as parameters from the associated user site.<br><br>Bursts of video cells are generated. Each burst has a bit rate determined by the current burstiness factor and the peak bit rate, see Section 3.1.3.2, eqn. (3.3). Cells are output for the duration of the burst with a packetisation delay appropriate to the current bit rate. | | |

**Data Object**

The data object inherits the source object's fields and the method initialise.

| Object | Field Name | Field Definition |
|---|---|---|
| datasourceObj | rate | The bit rate of the current data source. |
| | duration | The time duration for the current data source. |
| | activesources | The number of data sources currently active. |
| | startTime | Time stamp for the call duration statistics |
| **Overview of Object Function** | | |
| There are five types of data source. The type is selected and the cells are output with a packetisation delay appropriate to the time duration and bit rate selected. The file size generated is divided into cells and an appropriate packetisation delay is calculated, eqn. (3.12). The cells are then output for the time selected, with a packetisation delay between each. | | |

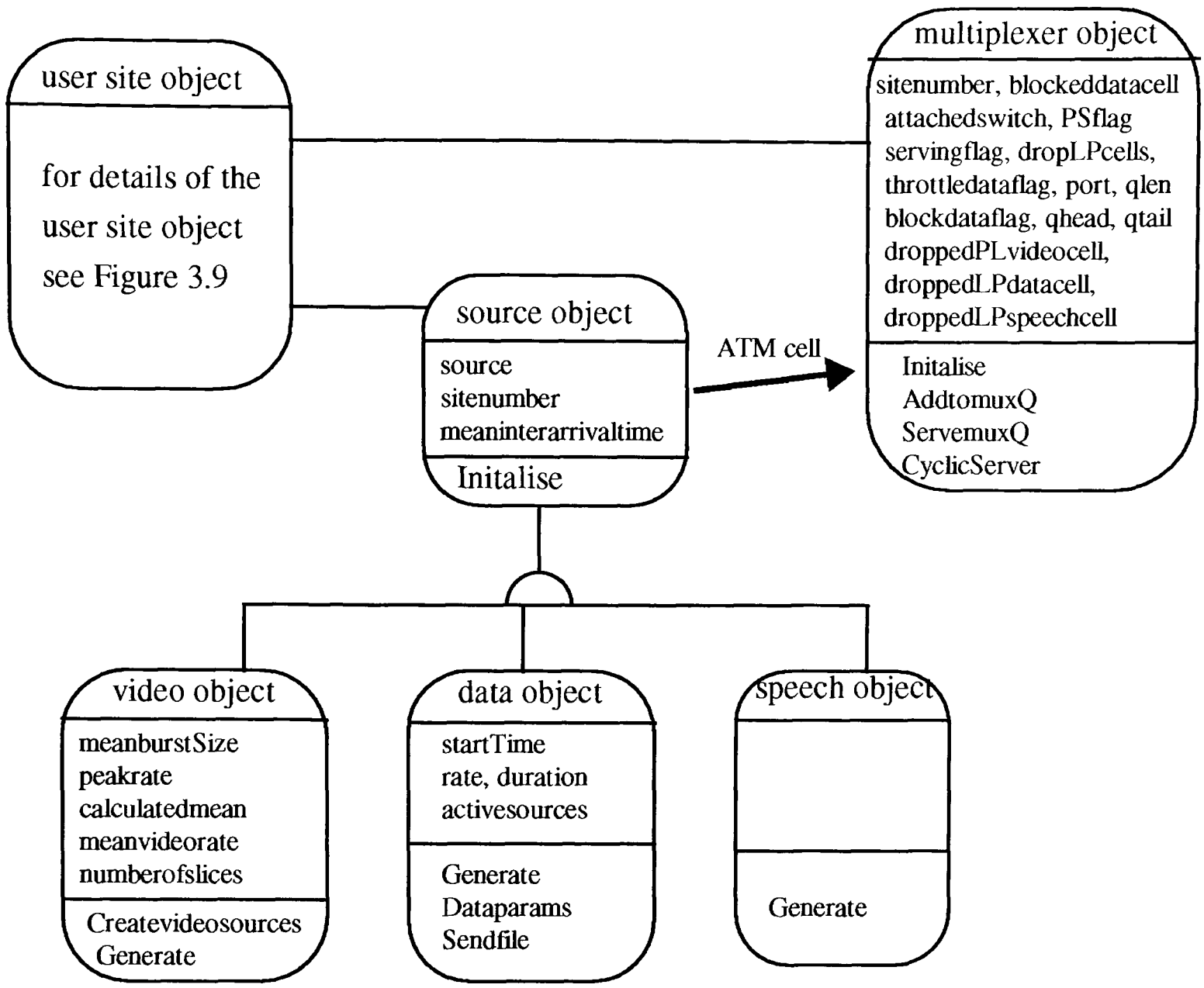



**Figure 3.12 User Site Object showing Traffic Source Objects and Multiplexer Object**

**Multiplexer Object**

The multiplexer object receives ATM cells from the traffic source objects, and passes them to the ATM network module, see Figure 3.12

| Object | Field Name | Field Definition |
|---|---|---|
| muxObj | sitenumber | The number of the user site associated with this multiplexer object. |
| | attachedswitch | The number of the ATM switch local to this multiplexer. |
| | port | The number of the input port to use at the attached switch. |
| | servingflag | This flag indicates whether the server is busy or idle. |
| | PSflag | Flag for the RT queue, which is used to tell the server to give priority service to the RT queue. The flag is set when the queue length reaches the threshold $RT_1$. |
| | dropLPcells | A flag associated with the RT queue threshold $RT_2$. It tells the server to drop all low priority cells as they reach the head of the queue. |
| | throttledataflag | Flag associated with threshold $T_1$ used to slow data cells arriving at the multiplexer. |
| | blockdataflag | Flag which is set if the non-RT queue grows too large, which tells the data sources to stop sending cells to the multiplexer. Associated with threshold $T_2$. |
| | droppedLPvidcell | Counter for the number of low priority video cells dropped at the multiplexer. |
| | droppedLPspeechcell | Counter for the number of low priority speech cells dropped at the multiplexer. |
| | droppedLPdatacell | Counter for the number of low priority data cells dropped at the multiplexer. |
| | qhead, qtail, qlen | Variables used for queue handling at the multiplexer. |
| | resetqlen | Queue length counter which is used for collecting statistics and is reset after each report. |

**Overview of Object Function**

The number of the user site, the attached switch and the input port that the multiplexer must use to access the ATM network are input parameters for the initialisation method.

The multiplexer has three queues: one for arriving cells from the network and two for outgoing cells. The two outgoing queues are for RT and non-RT cells respectively. Each server has a flag to indicate if it is busy or not.

The server serves the two outgoing queues alternately, until the RT queue length exceeds the threshold ($RT_1$). This causes the PSflag to be set and the server then begins priority service for the RT queue until the queue length falls below the threshold.

If the RT queue length exceeds the second threshold ($RT_2$), then the dropLPcells flag is set. The server then discards each low priority cell it removes from the queue, until the queue length falls below the threshold again.

If the non-RT queue exceeds the threshold ($T_1$), then the throttledataflag is set and the data sources must reduce the rate that they are sending cells to the multiplexer. If the non-RT queue length exceeds the threshold $T_2$ then data cells arriving at the multiplexer are blocked and will be dropped as they try to enter the queue.

### 3.2.1.2 ATM Network Module

The ATM network object creates an array of ATM switches, see Figure 3.13.

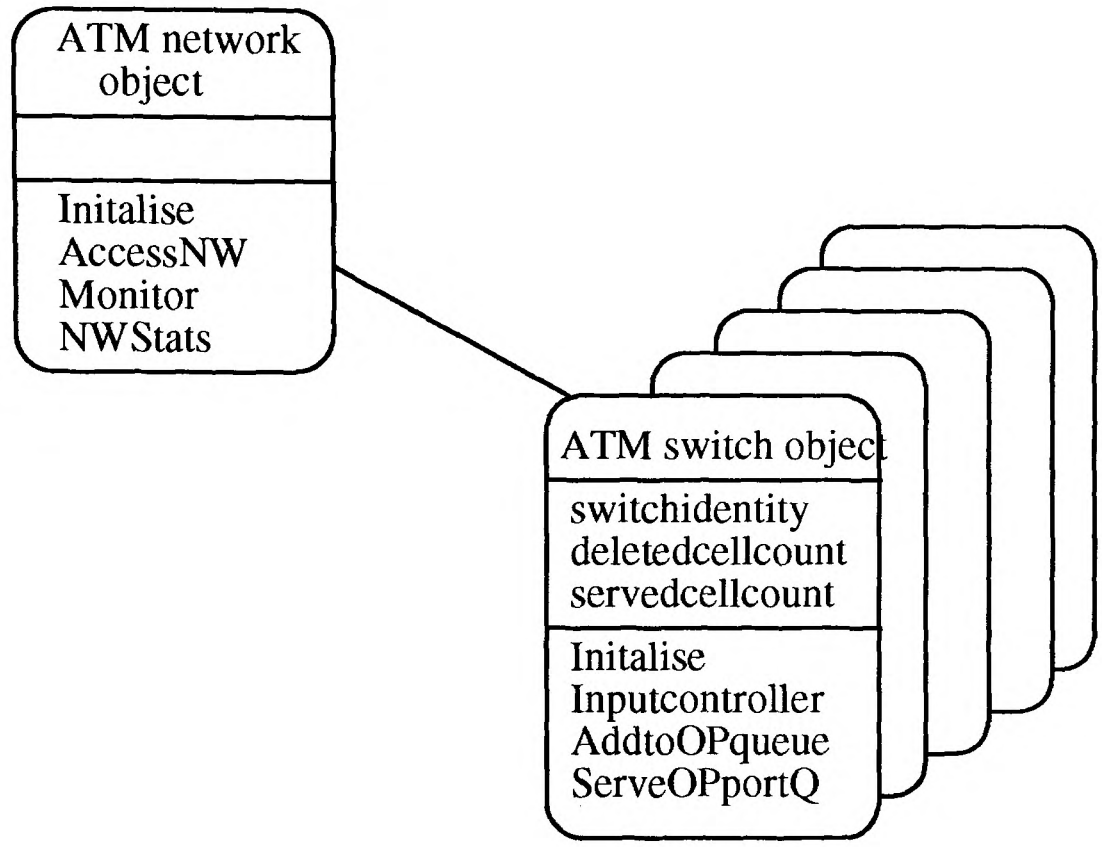



**Figure 3.13 ATM Network Object and Associated ATM Switch Objects**

**ATM Network Object**

| Object | Field Name | Field Definition |
|---|---|---|
| ATMnetworkObj | | |
| **Overview of Object Function** | | |
| The network manager object creates and initialises an array of ATM switch objects that form the network, and the variables used to collect statistics about the network. The network routing and the dynamic routing tables are also loaded centrally by this object | | |

**ATM Switch Object**

| Object | Field Name | Field Definition |
|---|---|---|
| ATMswitchObj | switchidentity | Identification number for each switch, used to access the ATM switch in an array of switches. |
| | deletedcellcount | Counter for mis-routed cells that have no corresponding entry in the routing tables, and so are discarded. |
| | servedcellcount | Counter for the number of ATM cells passing through each output port. |
| | outputQhead | Marker for the head of each of the output port queues. |
| | outputQtail | Marker for the tail of each of the output port queues. |
| | qlen | An array of queue length counters for each of the output ports. |
| | resetqlen | An array of queue length counters for each of the output ports which is used for statistical output. These are reset to zero after each report. |
| | flag | An array of flags which indicates if the server at a particular output port is serving or idle. |

**Overview of Object Function**

Each ATM switch initialises the output port queues, flags and statistical variables which are associated with each output port. Each output port has a server and a flag, which indicates if the server is busy or idle. Any mis-routed cells also need to be counted, using a statistical variable. These are cells that have VCI address labels that can not be resolved and so can not be correctly routed.

Cells are passed from switch to switch, using the network routing table and the dynamic routing table. At the last switch, the cell is passed to the user site manager, which passes it to the correct destination user site.

### 3.2.2 Methods and Fields for Objects

For each of the objects discussed in Section 3.2.1, the associated methods and fields for those objects, are laid out in the tables below.

### 3.2.2.1 User Site Methods

**Methods for User Site Manager Object**

| Method | Field Name | Function of Method |
|---|---|---|
| Initialise | randomseed<br>Numofsites | The input parameters, (randomseed and Numofsites) are the integer value for the random number generator and the total number of user sites allowed, respectively. An address book is created, which is accessed by all user sites. An array of user sites is created and initialised. Each usersite is called to initialise. |
| Accessusersite | ATMcell<br>siteNo | Input parameters are the ATM cell from the network and the site number of the destination user site. The cell is then passed to the correct user site multiplexer queue. |
| Monitor | runlength<br>interval | Input parameters are the length of the simulation run (in seconds) and reporting interval for statistics (in seconds). The seconds are converted to slotted time. At the end of each reporting interval, the statistics are requested from each user site and from the user site manager. When the simulation run is completed, a flag is set and all objects terminate. |
| UsersiteStats | | The current simulation time is converted back into RT (seconds) and output at the start of each statistical report. The number of cells generated by each user site, the queue lengths at the multiplexer and the delays encountered are output to a file. All statistical variables are then reset. |

**The Methods used by User Site Objects**

| Method | Field Name | Function of Method |
|---|---|---|
| Initialise | numberofthissite | The input parameter to this method is the identifier for this user site (numberofthissite). If the user site generates traffic it creates the traffic source objects. All sites create a multiplexer object. Each user site also knows which ATM switch it is attached to and the input port that it must use to access that switch. The run-time parameters are read in from a file. These are the inter-arrival time for each source, the maximum number of each source allowed and the peak bit rate for video sources. This method also initialises the PBX for telephone calls and the leaky bucket. Each source which outputs traffic onto the network is then called to generate traffic. |
| InitialiseLeakybucket | ID<br>sourcetype | Each new traffic source that starts up must initialise a record in the policing array. The source ID and sourcetype |

| | | |
|---|---|---|
| | rate<br>meanburst | are used as the index for the policing array. For video the bit rate and the mean burst length are used to calculate the decrement period for the leaky bucket. |
| Leakybucket | ATMcell | All non-RT ATM cells are passed to the buffered leaky bucket and placed in the queue. The VCI label and the source type is read from the cell header. The decrement function is called if it has not been activated and a flag set to indicate that it is active. The cell counter is compared with the threshold value. If the counter is less than the threshold then the cell is counted and passed to the multiplexer. Otherwise the cell is tagged or discarded before being passed to the multiplexer. |
| VirtualLeakybucket | ATMcell | All RT ATM cells are passed to the virtual leaky bucket. The VCI label and the source type are read from the cell header. The decrement function is called if it has not been activated and a flag set to indicate that it is active. The cell counter is compared with the threshold value. If the counter is less than the threshold then the cell is counted and passed to the multiplexer. Otherwise the cell is tagged and passed to the multiplexer or discarded. |
| PasstoMux | ATMcell | Used to route cells from different policing functions to the correct multiplexer queue. If a RT cell is passed to this method then it is routed to the RT queue. If it is a data cell then it is routed to the non-RT queue at the multiplexer. |
| Decrementfunction | ID<br>sourcetype | The input parameters, ID and sourcetype, are both identifiers for the policing array. Using the parameters stored in the policing array, this method periodically decrements the leaky bucket cell counter, to keep it below the threshold for conforming sources. This method runs until the leaky bucket method terminates. |
| Increment | source | The source type of a cell is used to count the number of cells generated of each type, at each user site. |
| IncBlocks | block | The size of the block of cells for each video burst is stored for statistical counting. |
| AddtoPBX | linenumber | The line number of each new phone is the input parameter for this method. A new phone call object is created and the call status of the phone line is changed to engaged, to indicate that the line is in use. |
| PBXfreeline | linenumber<br>callTimestamp | The line number of the completed phone call is used as the index for the PBX array. The phone call is disposed of and the status of the line is changed to free. The callTimestamp |

| | | |
|---|---|---|
| | | is used to calculate the duration of the call. |
| CAC | newrate<br>source<br>ID<br>accepted | The call admission control function at the user site is called to determine if there is sufficient available bandwidth to allow the call. The bit rate, the source type of the new connection and an identifier used for accessing the policing array, are passed as input parameters. For speech, the bandwidth requested is half the coding rate.<br><br>If the current allocated bandwidth + new call is less than the total capacity of the link then the new call is accepted and the current capacity of the outgoing link is updated to include the new call. If the link capacity is exceeded then the call is rejected. The parameter "accepted" is returned to the calling traffic source with value either True or False. |
| ReleaseResources | rate<br>source<br>ID | The bit rate, an identifier and the source type are the input parameters for this method, which releases the leaky bucket and updates the current used capacity. The identifier and the source type are used to index the policing array. The current link capacity is then reduced by the bit rate of the terminated connection. |
| Receivecell | receivedATMcell | All ATM cells arriving at a user site are counted from each destination separately, for statistical purposes. For all cells arriving at user site 3, the end-to-end delays are also calculated. These are stored for RT and non-RT cells and also independently for each of the different types of cell. The last cell for each video burst holds the start time of that burst and so the burst duration can be calculated. |
| SiteStats | | This method outputs the statistics for the traffic sources and the multiplexer. Cell delays for each traffic source, the multiplexer queue lengths, the number of cells still in the queue and the number of cells dropped by the multiplexer are output to a file. The number of cells tagged or deleted by the policing function are also output. |

### 3.2.2.2 Methods used by Source Objects

**The Method used by All Source Objects**

| Method | Field Name | Function of Method |
|---|---|---|
| Initialise | lam<br>s<br>numberofthissite | The mean inter-arrival time (lam), the source type for this source (s) and an identifier for this user site (numberofthisite) are input parameters. The mean inter-arrival time is converted to slotted time. |

All sources use the generic Source object for initialisation.

**The Method used by Speech Objects**

| Method | Field Name | Function of Method |
|---|---|---|
| Generate | | The time to wait before generating the next phone call has an exponential distribution and the mean inter-arrival time is entered at the start of the simulation from an input file. If the current number of active calls is less than the maximum allowed, then the array is searched for a free line in the PBX. The CAC function is called to request bandwidth and if the call is allowed, then the method AddtoPBX is called to request that the PBX creates the phone call object and change the status of that line to engaged. The phone call object is then called to generate speech cells. This method loops continuously for the duration of the simulation, to maintain the number of active phone calls as close to the maximum as possible. |

The PBX is modelled as an array of phone call objects, each with a status flag attached, which indicates if the line is free or engaged.

**The Methods used by Phonecall Objects**

| Method | Field Name | Function of Method |
|---|---|---|
| Makecall | linenumber<br>sitenumber | This method generates the speech cells and outputs them to the policing function.<br><br>The line number for each call and the originating site number are input parameters to this method.<br><br>The call length is generated using an exponential distribution with a mean of 3 minutes. A user site destination is determined for this call and the corresponding VCI label is found from the address book. The time that this call will terminate is also calculated and used as the terminator for the loop that generates speech cells. The length of the next talkspurt is selected from an exponential distribution, with a mean of 1.34 seconds and is converted into cells. Speech cells are then transmitted in pairs with a packetisation delay equivalent to 12 ms between each pair, until the talkspurt is completed. The first speech cell has its CLP bit set to indicate a high priority cell and the second cell has its CLP bit set to indicate a low priority cell.<br><br>Each cell is counted and then passed to the policing function. A silence period is then generated using an exponential distribution with a mean silence period of 1.67 seconds. No speech cells are generated during a silence period. |
| | | Talkspurt and silence periods alternate until the phone call ends. When the call ends, the reserved link capacity is released. The PBX releases the line by changing the status of a flag to indicate a free line and the phone call object is disposed of. |

**The Methods used by Video Objects**

| Method | Field Name | Function of Method |
|---|---|---|
| Createvideosources | videosources<br>rate<br>count | The number of video sources (videosources) and the peak bit rate (rate) for the video sources are input as parameters. The mean bit rate is calculated and is used to calculate the expected mean burst size. The number of video sources indicated are then activated. |
| Generate | ID | Using the video mean bit rate, the effective bit rate is calculated and is used to request bandwidth using the CAC function for each video connection. If the call is accepted, then the leaky bucket is initialised, using either effective bit rate or the peak bit rate. A destination site is selected and the VCI label for that destination is found in the address book<br><br>The first burst is output at the peak bit rate. The size of each burst is obtained using an exponential distribution, with the mean corresponding to the mean burst size. The current burst size is then rounded up to the nearest integer, and used as the counter for a loop which outputs the appropriate number of video cells.<br><br>Each burst is divided into a number of slices. The number of ATM cells in a particular slice is a random number between 5 and 100. The first cell of each slice has it's GFC field set to 1, to indicate the start of a slice. This tells the policing function that this is the start of the next resynchronisation point for the video stream. If the policing function is discarding video cells then the receipt of a new slice will cause it to reset and stop dropping cells.<br><br>As cells are assembled for transmission, there is a packetisation delay between each, appropriate to the current cell rate. As each burst may have a different cell rate, this packetisation delay must be calculated on a per burst basis. The statistics for the burst size is updated and the simulation time for the start of this burst is also stored.<br><br>Using the current bit rate, the burst of video cells are output with the relevant packetisation delay between each. As each video cell is generated, the ATM cell header fields are filled (VCI label, source type, this site number (origin), destination site number). The current simulation time is entered into the cell start time field (cellstartTime). This time stamp is used to calculate the various delays experienced by individual cells.<br><br>The last cell for the burst is time stamped with the burst start |

| | | |
|---|---|---|
| | | time. A non-zero message start time field in a received video cell also indicates that it is the end of a burst, and will initiate collection of the burst statistics.<br><br>Each cell is counted and then passed to the policing function. At the end of the burst, the next burst is created by selecting the burstiness (1 - 8) and using it to calculate the current bit rate. The packetisation delay for that bit rate is again calculated and the next burst is then generated, until the end of the simulation. |

**The Methods used by Data Objects**

| Method | Field Name | Function of Method |
|---|---|---|
| Generate | maxNodatafiles | This method runs continuously for the duration of the simulation, maintaining the number of active data files as close as possible to the maximum, until the simulation ends.<br><br>The maximum number of data files allowed is entered as a parameter by the calling method. Initially, the method begins by generating one of each type of data file.<br><br>The delay between files is selected from an exponential distribution, using the meaninterarrivaltime as the mean for the distribution.<br><br>If the number of active data sources is less than the maximum allowed, then the array is searched until a free line is found and the number of active data sources is incremented.<br><br>The data type of the next file is selected randomly (1-5). The method which selects the bit rate and time duration for this file, is called. The CAC method is also called and if the call is accepted the leaky bucket is initialised and the data cells are output by calling the method Sendfile. |
| Dataparams | source | The data type for this file is used to determine the allowed range of bit rates and time duration's for each data type. The rate and the time duration for this data type are then selected from within the range using a uniform distribution. |
| Sendfile | rate<br>duration<br>source<br>dataID | This method generates data cells at the bit rate and for the time duration indicated by the input parameters.<br><br>The bit rate and time duration for this data source, are used to calculate the data message size. This is divided into a number of ATM cells and the packetisation delay between each cell is calculated. The start time is used to time stamp the last cell of the |

| | | |
|---|---|---|
| | | file.<br><br>A destination is selected and the VCI label for that destination is found using the address book. The cells are then output with the appropriate packetisation delay between each one. Each cell is counted and then passed to the leaky bucket. When the call ends the resources are released. |

### 3.2.2.3 Multiplexer Object Methods

**The Methods used by Multiplexer Objects**

| Method | Field Name | Function of Method |
|---|---|---|
| Initialise | switchNo<br>useport<br>numberofthissite | The user site number, an identifier for the attached ATM switch and the number of the input port on that ATM switch are passed to the multiplexer as input parameters. The multiplexer queues and the flags which indicate the current state of the server (busy or idle) are generated here.<br><br>One incoming queue and two outgoing queues are created. There is an outgoing queue for RT and one for non-RT traffic. The flags which are associated with each server, indicating if a particular server is serving or idle, are created. There are also flags to indicate when the RT queue requires priority service, if low priority cells should be dropped. For the non-RT queue flags are created to indicate when to throttle back data sources and to block data sources. |
| AddtomuxQ | ATMcell<br>queueNo | As the different traffic sources generate ATM cells, they are passed to the multiplexer to be added to the queue.<br><br>The ATM cell to be added, and the identifier for the queue to use are input parameters to this method. If the cell is from a RT source, then it is added to queue number 3. If it is from a non-RT source then queue number 2 is used. If the cell is received from the network, then queue number 1 is used.<br><br>The relevant queue length counter, is incremented each time a cell is added to that queue. If the cell is added to an empty queue, then the serving flag for that queue is checked. If the flag indicates that the server is idle, then the server is called and the flag is set to busy. If there are already cells in the queue and the server is busy, then the cell is added to the end of the queue to await service.<br><br>The queue lengths are continuously monitored. The priority service flag (PSflag) is set if the RT queue length exceeds $RT_1$ and the dropLPcells flag is set if it exceeds $RT_2$. The |

| | | |
|---|---|---|
| | | throttledataflag is also set if the non-RT queue exceeds $T_1$ and the blockdataflag if the queue exceeds $T_2$. Each flag is released as the queue length falls below the appropriate threshold. |
| ServemuxQ | queueNo | This server provides service for incoming traffic at the multiplexer.<br><br>The current simulation time is stored each time the multiplexer suspends and this is used to determine the number of idle slots that have elapsed since the multiplexer stopped serving. The multiplexer waits for the start of the next slot to re-synchronise the server. The ATM cell at the head of the incoming queue is removed and the cell is passed to the Receivecell method to update the statistics. |
| CyclicServer | queueNo | When the server is restarted, it must re-synchronise with the start of the next slot. This is done by calculating the number of slots that have passed since the server suspended and then waiting for the start of the next slot.<br><br>The cyclic server serves the RT and non-RT queues alternately, while the queue length is below the thresholds. If the RT queue length exceeds the first threshold, $RT_1$ then the server begins priority service to the RT queue. If the RT queue length exceeds the second threshold, $RT_2$ then the server discards each low priority cell it removes from the queue.<br><br>If the non-RT queue length exceeds the threshold $T_1$ then the data sources are throttled back. If the queue length reaches $T_2$ then any subsequent data cells arriving are discarded. |

### 3.2.2.4 ATM Network Methods

**The Methods used by ATM Network Object**

| Method | Field Name | Function of Method |
|---|---|---|
| Initialise | numofswitches<br><br>numberofsites | The number of switches in the network and the current number of user sites are entered as input parameters and used to initialise the ATM switch objects that form the network. The number of input and output ports at each switch are a fixed parameter. The routing tables are also initialised and the Monitor method is activated. |
| AccessNW | ATMcell<br>switchNo<br>portNo | The ATM cell, the switch number and the input port number are all passed as parameters to this method. The ATM cell is passed to the network from a user site through this method. The cell is then passed to the input controller at the port indicated, at the switch identified by the switch number. |
| Monitor | runlen<br>interval | The simulation run length and the reporting interval are the input parameters for this method. This method has the function of synchronising the reporting of statistics with that of the user-sites and setting a flag to indicate to all objects when the simulation time has expired. All the methods in this module monitor this flag and terminate when the flag is set to True.<br><br>During the simulation, the method waits for the duration of a reporting interval and then calls the network statistics method, within the network manager. The run time and the reporting interval are both user-defined at run time. |
| NWStats | | When the Monitor asks for the statistical reports to be generated on the status of the ATM switches, this method is activated.<br><br>The current simulation time is converted into seconds for report headers and the statistical variables are output to the file. The mean and maximum queue length, the total cells served during the interval, the number of cells still waiting in the queue and the line utilisation for each output port in use, are all output to a file. The statistical variables are then reset at the end of each reporting period. |

**The Method used by ATM Switch Objects**

| Method | Field Name | Function of Method |
|---|---|---|
| Initialise | switchidentifier | A unique switch identifier is passed to each instance of this object which also is the index to the array of switch objects. The output port queues and associated flags are created and initialised. |
| Inputcontroller | ATMcell<br>IPport | When an ATM cell to the network is passed to a switch, the Inputcontroller, at the input port of the switch, receives the cell. The ATM cell and the identifier for the input port to be used are the input parameters for this method.<br><br>The dynamic routing table is searched for a match between the VCI label, extracted from the ATM cell header, and the input port number that the cell arrived at. When a match is found, the new VCI label and the number of the next output port are read from the table. The new label is loaded into the VCI label field of the cell. The correct output port to route the current ATM cell to is also obtained from the routing tables and the cell is passed to the AddtoOPqueue method with the output port as a parameter. Any cells which are not assigned an output port are assumed mis-routed and are discarded and the appropriate counter is incremented.<br><br>A switching delay is imposed before the cell is added to the correct output port. |
| AddtoOPqueue | ATMcell<br>OPport | The ATM cell is passed as an input parameter, along with the identifier for the correct output port.<br><br>The ATM cell is added to the queue at the output port indicated. If the ATM cell is added to an empty queue, then the server is called and the flag set to indicate that the server is now busy. Otherwise, the cell is added to the end of the queue and must wait for service. |
| ServeOPportQ | queue | The next ATM cell at the head of the queue is removed from the output queue. The output controller must look up the routing information for the cell, to determine whether to pass it to a user site or to another switch. The number of either the user site or the switch and the input port to use, must be found in the network routing table<br><br>While there are cells in the queue, the server continues to serve. When the server stops serving, the flag is reset to show that the server is again idle. The utilisation of the server is measured continuously. |

### 3.2.3 ATM Cell

ATM cells have the following fields :-

| | |
|---|---|
| VCI label | carries the address label which is used for routing, and which is changed at each node, as the cell crosses the network |
| CLP | cell loss priority field - indicates a high (0) or low (1) priority cell |
| GFCfield | generic flow control field - used to reset leaky bucket for video traffic |

Cells also carry the following fields not present in actual ATM cells, but used for the purposes of collecting statistics during the simulation:-

| | |
|---|---|
| cell time stamp | enables cell delays to be calculated (time waiting in queues and end-to-end delays) |
| message start time | enables statistics to be collected on the call holding time or in the case of video, the delay experienced by the last cell of a burst |
| source type | allows the different types of cells to be identified for collecting statistics |
| site origin | enables cells from particular sites to be identified for counting at the destination |
| VCid | used to differentiate cells from the same site and of the same type (e.g. for speech cells, the line number is used) |

**Table 3.6 Information Fields in ATM Cell Header**

## 3.3 Implementation

The simulation is written using MODSIM (CACI Inc.) and has been run on a Sun Workstation. The program is split into four modules and these are the Main Module, the Usersite Module, the ATMnetwork Module and a Definitions Module.

The Main Module starts up the program and activates the other modules to begin the simulation. The Usersite Module generates the usersite manager and the ATMnetwork Module generates the ATMnetwork. The definitions module is used hold the definition of the ATM cell which is imported by both the Usersite Module and the ATMnetwork

Module. This removed the cyclic dependencies which occurred and which caused problems for the compiler.

The simulations were run for 200 seconds, using a number of random seeds. This is to ensure that any variation in performance (queue lengths, delays and losses) is due to the policing strategy and the multiplexer access method and not due to any variation in the numbers of cells generated.

### 3.3.1 Validation of the Speech Model

The work by [RAMA91] on delay analysis of a packet voice multiplexer has been used to validate the speech model used in the simulations.

In their work, each call is a CBR source where voice packets are generated deterministically and are serviced at the multiplexer with a constant service time. The arrival of new calls follow a Poisson distribution. The combined arrival stream at the multiplexer is a superposition of many periodic streams from all the active speech sources. The same pattern of speech packet arrivals repeats periodically, see Figure 3.14, until a call ends or a new call is generated, which causes the arrival stream to change. They calculate the probability distribution for the number of speech calls in progress and the mean waiting time for an arbitrarily selected packet.

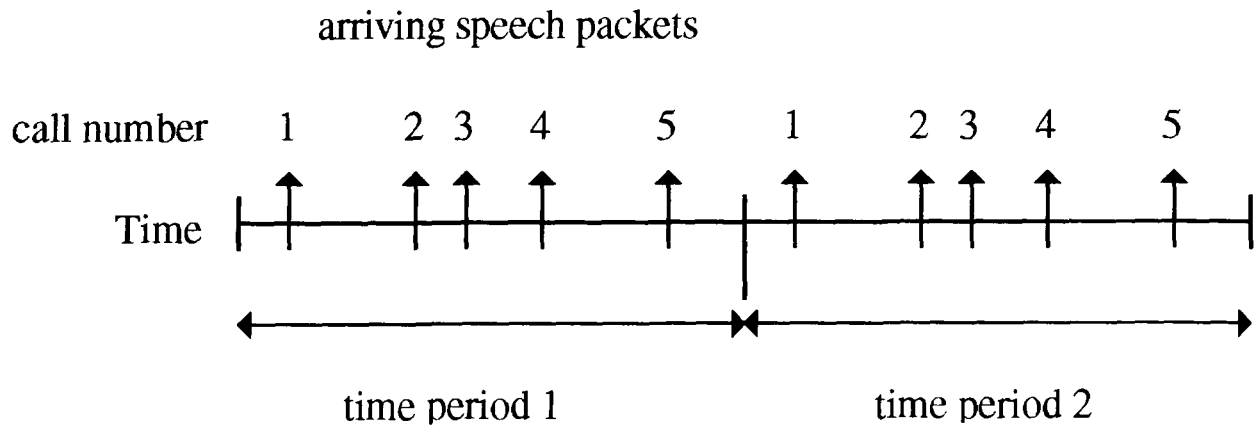



**Figure 3.14 Periodic Speech Packet Arrivals**

The simulation model developed in this work was run with a fixed number of calls in progress. Each call alternates between talkspurt and silence. Thus the probability distribution for the number of calls in talkspurt mode can be calculated using the binomial distribution. This combined with the packet waiting time in [RAMA91] gives the mean waiting time (multiplexer access time) for an arbitrarily selected packet. This was compared with the value estimated from the simulation and a good agreement was obtained.

## 3.4 Proposed Policing Strategy

The proposed policing strategy has two main aims. These are to minimise losses and delays to RT traffic and to protect non-RT traffic from losses. Each of the different classes of traffic defined in Section 3.4.1 has different requirements and uses separate techniques for bandwidth management. Bandwidth control is achieved using dynamic queue management at the multiplexer, as described in Section 3.4.2.

The policing function at the UNI is based on the leaky bucket and has different functions for the three classes of traffic, as described in Section 3.4.3.

### 3.4.1 Traffic Definitions

Three classes of traffic are defined :-

Class 1 (data) traffic needs lossless delivery, while delays, within reason, are less important. The loss of a class 1 cell may cause the remaining cells of the originating PDU to be discarded on arrival at the destination. This would also necessitate the re-transmission of the entire PDU, which may contain a large number of ATM cells.

Class 2 (speech) - represents RT traffic and as such, requires minimal delays. Violating cells are tagged and allowed on to the network. If the traffic in class 2 is speech, then up to 10 % of the low priority cells may be discarded without noticeable loss of audibility to the user.

Class 3 (video) traffic is also RT, with the delay restrictions that this implies, but is much more loss sensitive than class 2 traffic. Since the smallest independent resynchronisation point for

coded MPEG video is the video slice, it seems reasonable to ensure that a damaged slice will never be received. A damaged slice, or a potentially damaged slice, could cause the decoder, at the destination, to become desynchronised. It is more appropriate to delete violating cells in a controlled manner, than to allow tagged cells on to the network, where they might be arbitrarily discarded.

### 3.4.2 Multiplexer Service Strategy

The multiplexer has two outgoing queues. One is reserved exclusively for RT traffic (Classes 2 and 3) and the other is for non-RT traffic (Class 1). Queues are served alternately (cyclic service). Each queue has two thresholds associated with it. If the RT queue length exceeds the first RT threshold, then at the start of the next service instance, the server switches over to give non-pre-emptive priority service to the RT queue. Priority service continues until the queue length drops below the threshold. If the RT queue continues to grow beyond the second RT threshold, then all low priority cells are discarded as they are removed from the queue. Since these are mainly low priority speech cells, there is little detrimental impact on the overall QoS for Class 3 traffic. This may even benefit the Class 3 traffic, as the waiting times within the queues can be shortened, at times of congestion, giving priority to the high priority cells.

The non-RT queue also has two thresholds associated with it. If the non-RT queue length exceeds the first threshold, then the rate at which the cells arrive at the multiplexer is reduced, by doubling the service time at the buffered leaky bucket. If the queue length exceeds the second threshold, then data sources are blocked and any data cells subsequently arriving at the multiplexer are discarded. [DITTM91] advocates using a flow throttling function to enforce a source close to it's mean rate.

The setting of the thresholds is critical to the performance of the multiplexer. Too high and the delays will be too large. Also the system will be slow to react to over-loading and cells may be lost in an uncontrolled manner. The key here is that a cell may be discarded, but only in a controlled way, and this benefits the RT and high priority cells.

[BONO93] states that using large buffers at multiplexers (>500 cells) can improve multiplexing efficiency and will reduce the importance of transient phenomena. Instantaneous arrival rates can be smoothed. However, it also requires the implementation of complex priorities to cope with

intolerable delays for delay sensitive traffic, that such a buffer could impose. A buffer size of 500 cells could impose a delay of nearly 5 ms (actually 4.711 ms) for a link speed of 45 Mb/s. The buffer size for non-RT traffic has been set to 30 cells. This imposes a potential queuing delay of 282 μs or 565 μs if cyclic service is in operation, either of which are acceptable for non-RT traffic.

The thresholds for the RT queue are set low (2, 6). Previous work by [GAN95] has shown that performance for the RT traffic is significantly improved. Priority service for the RT queue begins when more than two RT cells are queued, and dropping low priority cells mechanism begins when more than six cells are queued. These thresholds are deliberately low to minimise the delays for RT traffic. The thresholds for the non-RT queue however, are higher (20, 30), as delays are not as important as accurate delivery.

### 3.4.3 Policing Strategy

The positioning of the policing function is an important factor in the overall performance. [RATH91] states that the policing function should be placed as close as possible to the traffic source. Previous work by [GALL89] and [LAET95] advocate this approach. In this work, the policing function is placed before the multiplexer. This means that cells which would be discarded by the policing function do not waste resources or delay conforming cells, while being queued at the multiplexer.

There are three different policing strategies, corresponding to the three Classes of traffic. Class 1 traffic is policed using a buffered leaky bucket. Cells arriving are placed in the buffer and served FIFO. If the leaky bucket runs out of credits, then the cells must wait in the buffer until the credits are refreshed. No Class 1 cells are tagged or discarded. If the queue length at the multiplexer exceeds the threshold $T_2$ then Class 1 cells are blocked at source, to prevent buffer overflow at the multiplexer.

Class 2 traffic is policed using a virtual leaky bucket. The counter is periodically decremented, according to the parameters declared during call initialisation. The cells are not delayed and pass through while the counter is below the threshold defined. If the source does not conform to the

declared parameters, then the cells are marked as low priority and allowed on to the network. At times of congestion these cells will be discarded first.

Class 3 traffic is also policed using a virtual leaky bucket, as this is a RT source and delay sensitive. Here the policing function has the added feature that violating cells are deleted and that once the deleting function has been activated, it continues to delete cells until a reset is received from the source. The reset indicates the start of the next video slice, and hence the start of the next re-synchronisation point for the video stream. It could be potentially more damaging to the final picture to lose a cell from the middle of a slice than the complete loss of the tail end of the slice. Also the clipping of the remaining cells of a damaged slice actually cause the cell losses to be focused into a smaller number of slices, rather than spread throughout a significantly larger number of slices. The results in Chapter 4 show that this is the case.

The method used to dimension the leaky bucket is determined by the class of traffic requesting a set-up, as discussed in Section 3.1.5.

## 3.5 Conclusion

A composite strategy has been proposed to police the access link to an ATM network and to manage the buffers at a multiplexer. A simulation model has been designed and built as described. The model has been used to compare the performance of a standard VLB and the proposed UPC and the results are presented in Chapter 4.

# Chapter 4 - Results

A series of simulations have been performed, using the model described in Chapter 3, which compare the performance of the various policing and multiplexer service strategies. RT and non-RT queues, with cyclic service, are implemented at the multiplexer, in all cases.

Questions to be answered :-

1. Does the positioning of the LB and the multiplexer have any effect on either the queue lengths or the policing function itself? The options are that cells can be passed to the LB first and then to the multiplexer or to the multiplexer first and then to the LB.

2. Can the proposed policing function :-

    i. improve QoS for RT cells by :-
        - preventing excessive loss of video cells
        - reduce the number of damaged video slices
        - minimise lost speech cells
        - preventing excessive delays to RT cells

    ii. maintain QoS for non-RT cells by :-
        - giving reasonable end-to-end delays for data cells
        - protect data cells from being discarded

## 4.1 Simulation Experiments

The simulation models have been run for 200 seconds with different proportions of RT and non-RT traffic, to examine the impact this has on the different strategies under investigation. Statistics have been collected every 20 seconds throughout the simulation. All error limits quoted to 10% confidence. The first 20 seconds are not included in the statistics as this was considered to be a 'warming up' period.

The first set of simulations were to ascertain if the positioning of the policing function has any impact on performance. The VLB model which deletes violating cells was used for this purpose.

Simulations 2, 3 and 4 are to compare the performance of the VLB and the S-LB with the policing function positioned before the multiplexer using different proportions of RT and non-RT traffic.

| | Function | Data | Video | Speech |
|---|---|---|---|---|
| Simulation 1 | positioning of VLB | 60 maximum | 3 at 10 Mb/s peak | 100 maximum |
| Simulation 2 | S-LB Vs VLB | 20 maximum | 3 at 10 Mb/s peak | 100 maximum |
| Simulation 3 | S-LB Vs VLB | 40 maximum | 3 at 10 Mb/s peak | 100 maximum |
| Simulation 4 | S-LB Vs VLB | 60 maximum | 3 at 10 Mb/s peak | 100 maximum |

**Table 4.0 Simulation Experiments Performed**

In each case the RT traffic parameters remain constant, while the maximum number of non-RT sources is varied, see Table 4.0. This causes an increase in the utilisation at each access link and hence on the network itself. The effect this has on the delays encountered by the various traffic types and on the multiplexer queue lengths can be observed.

### 4.1.1 Multiplexer Queue Management

Cyclic service is used at all multiplexers with priority service and low priority cell dropping for the RT queue. There are two thresholds associated with the RT queue ($RT_1$ and $RT_2$). When the queue length at the multiplexer exceeds threshold $RT_1$ the server switches over to priority service for the RT queue, at the start of the next service time. If the queue continues to grow and exceeds the second threshold $RT_2$ then all low priority cells in the RT queue are dropped as they reach the head of the queue. A previous simulation study [GAN95] has indicated that threshold values of 2 and 6 for $RT_1$ and $RT_2$ respectively, give a reasonable level of service to both the RT and non-RT queues.

The non-RT queue also has two thresholds ($T_1$ and $T_2$), which are only used in conjunction with the S-LB (Methods 4 - 7) and have the values 20 and 30 respectively. When the non-RT queue reaches the threshold $T_1$ then the LB slows the rate that data cells arrive at the multiplexer by increasing the service time to remove a cell from the buffer, When the queue reaches the second threshold $T_2$, then all data sources are told to stop sending cells and any cells arriving at the

multiplexer are dropped and must be re-sent by the source. Cells which are discarded by a multiplexer are counted for each source type separately at each user site.

### 4.1.2 Super Leaky Bucket

The Super Leaky Bucket (S-LB) uses tail-end-clipping for video traffic. Once the threshold value of the leaky bucket has been exceeded, the excess cells are discarded and not allowed onto the network. Cells continue to be discarded until the receipt of a reset command. This applies even if the LB counter falls below the threshold value. The first cell of the next video slice has its GFC field set to 1, which is the reset command and causes the tail-end-clipping function to stop discarding cells. This means that the tail end of the video slice is "clipped" to prevent corrupted slices from arriving at the decoder.

In this work the VLB algorithm [NIEST90] as described in Section 3.1.5 is used to police speech and data. For video connections the mean burst length and twice the mean burst length are used to dimension the S-LB.

Any violating speech cells are tagged as low priority cells, and passed to the multiplexer queue. The tagged speech cells may be allowed on to the network, but if the queue at the multiplexer grows too large, then they are discarded to protect video cells and high priority speech cells from excessive delay

Data cells are protected from being discarded using a buffered leaky bucket (B-LB) and may be delayed in a queue within the leaky bucket, until they are allowed to pass to the multiplexer. If the non-RT queue length at the multiplexer grows too large, then the service time to remove a cell from the B-LB queue is doubled to throttle back the flow of traffic. When the multiplexer queue is full any cells arriving are blocked from entering the queue and are considered lost.

### 4.1.3 Simulation Details

Details of the different policing function and the parameters used are given in Table 4.1.

| Method | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Policing | | | | | | | |
| data | VLB | VLB | VLB | B-LB | B-LB | B-LB | B-LB |
| speech | VLB | VLB | VLB | S-LB | S-LB | S-LB | S-LB |
| video | VLB | VLB | VLB | S-LB | S-LB | S-LB | S-LB |
| Thresholds | | | | | | | |
| $RT_1$ | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| $RT_2$ | 6 | 6 | 6 | 6 | 6 | 6 | 6 |
| $T_1$ | | | | 20 | 20 | 20 | 20 |
| $T_2$ | | | | 30 | 30 | 30 | 30 |
| LB Params | | | | | | | |
| Dec. Value | | | | | | | |
| data | 1 or 16 | 1 or 16 | 1 or 16 | 1 or 16 | 1 or 16 | 1 or 16 | 1 or 16 |
| speech | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| video | 16 | 1 x b | 1 x b | 1 x b | 1 x b | 2 x b | 2 x b |
| Video bit rate | effective | effective | effective | effective | peak | effective | peak |
| Action taken | | | | | | | |
| video | discard | tag | discard | discard | discard | discard | discard |
| speech | discard | tag | discard | tag | tag | tag | tag |
| data | discard | tag | discard | delay | delay | delay | delay |

b = video burst length
S-LB = super leaky bucket
B-LB = buffered leaky bucket

**Table 4.1 Summary of Policing Details**

Methods 1 to 3 are based on the VLB, while Methods 4 to 7 use the S-LB to police the UNI.

**Method 1**

Method 1 uses the algorithm proposed by [NIEST90], see Section 3.1.5, to dimension the VLB. The performance of this leaky bucket is used as a reference to compare all other strategies.

**Method 2**

This is also a VLB which uses Niestegge's algorithm for speech and data connections. However, video traffic uses a mean video burst length as the input to the algorithm, to dimension the threshold value and the decrement period of the leaky bucket. Violating cells are tagged as low priority and then allowed onto the network. The non-RT queue at the multiplexer has no threshold limit to restrict the queue length.

**Method 3**

This is essentially the same as Method 2, except violating cells are discarded and prevented from entering the network.

**Method 4**

Method 4 uses the mean burst length and the effective bit rate for video as the dimensioning parameters for the initialisation of the S-LB. Speech and data sources use Niestegge's algorithm to dimension the S-LB.

**Method 5**

Method 5 also uses the mean burst length, but the peak bit rate is used for video. Speech and data are as for Method 4.

**Method 6**

Method 6 uses twice the mean burst length and the effective video bit rate to dimension the S-LB. Speech and data as for Method 4.

**Method 7**

Here twice the mean burst size and the peak bit rate for video are used. Speech and data as for Method 4.

Methods 4 to 7 all implement tail-end-clipping for video traffic. All cells dropped or tagged by any of the LBs are counted by source type at each user site.

### 4.1.4 Input Parameters

The following input parameters were used for the simulations.

| Parameters for Main Module | Function | Value |
|---|---|---|
| runlen | length of simulation run. | 200 seconds |
| report | reporting interval | 20 seconds |
| randomseed | random seed (only one value per simulation run) | 2,3,5,6 or 7 |
| $RT_1$ | QLT for RT priority service | 2 |
| $RT_2$ | QLT for RT LP cell discarding | 6 |
| $T_1$ | QLT for non-RT throttle-back at leaky bucket | 20 |
| $T_2$ | QLT for blocking non-RT cells at source | 30 |
| **Parameters for each user site** | **Function** | **Value** |
| maxvideosource | maximum number of video sources allowed | 3 |
| peakrate | peak bit rate for video sources | 10 Mb/s |
| maxNodatafiles | maximum number of data sources allowed | 20, 40 or 60 |
| lam | inter-arrival time between data files | 0.001 |
| maxallowedcalls | maximum number of speech sources allowed | 100 |
| lam | inter-arrival time between phone calls (re-used) | 0.003 |

QLT = queue length threshold
LP = low priority

**Table 4.2 Input Parameters**

### 4.1.5 A Typical Simulation

A typical pattern of the utilisation found at a user site, for a single random seed (seed 5), can be seen in Figure 4.1 and includes the first 20 second interval, which is considered to be a "warming- up" period.

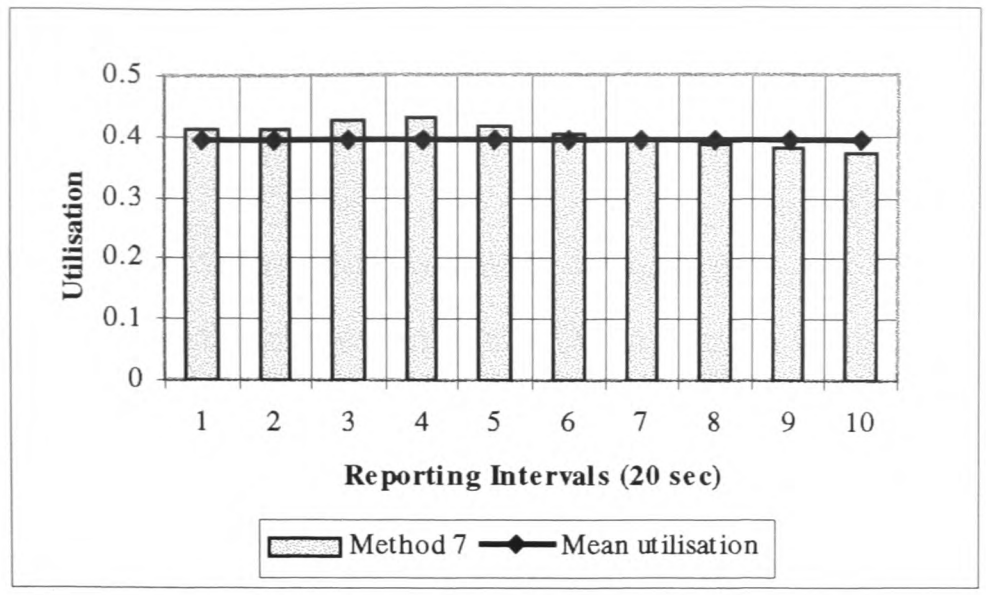



**Figure 4.1 Utilisation for S-LB Method 7**

The number of speech calls are maintained at the maximum allowed by having a very small inter-arrival time between calls. For each speech call in progress the proportion of time spent in talkspurt mode (T) is given by :-

$$T = \frac{\text{talkspurt}}{(\text{talkspurt} + \text{silence})} = 0.44 \qquad \text{----(4.1)}$$

The number cells generated per sec ($N_s$)

$$N_s = \frac{(\text{speech coding rate} * 0.44)}{\text{cell pay load}} \qquad \text{----(4.2)}$$

The expected number of speech cells generated in a 20 second period are 151,552 and the actual number generated during the simulation was 149,657.

The expected number of data cells has been estimated to compare with the number generated by the simulation Each data type has a range of bit rates and the midpoint has been used for the calculations within each category. Given that there are 20 data sources allowed and each has an equal probability of being selected there could be four of each type of data source active at any given moment. The number of data cells generated ($N_D$) in one second is :-

$$N_D = \frac{\text{mean bit rate}}{\text{cell pay load}} \qquad \text{-----(4.3)}$$

The number generated in 20 seconds becomes ($N_D$ * 20) = 326,059 and the number generated by the simulation is 326,828, which is a reasonable approximation.

There are also three VBR video sources active at each user site for the duration of the simulation. The video bit rate fluctuates throughout the simulation, see Figure 4.2.

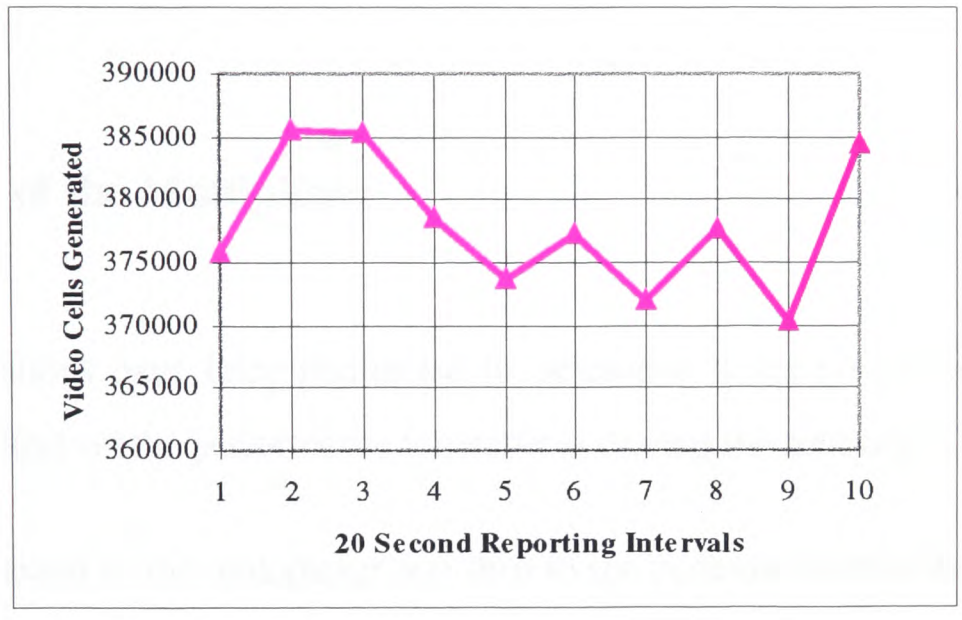



**Figure 4.2 Video Cells Generated**

The number of cells generated has been approximated to give an estimate of the expected access link utilisation (ut) for the given load.

$$u = \frac{\text{Total cells generated}}{\text{capacity of link}} \qquad \text{---(4.4)}$$

The estimate of the expected utilisation is 44% and the actual utilisation from the simulation was 40%.

The non-RT connections comprise a number of different sources with widely varied bit rates and time duration's. As one connection closes down another will start up fairly quickly, as the inter-arrival time for data connections is also very short, to maintain the maximum number active at

any given moment. The type of the next data source is chosen at random, within the allowed range. If the non-RT source that closed down had a low bit rate and the next one to start has a high bit rate, then there will be a large jump in the number of non-RT cells generated.

The VBR nature of the video sources and the wide variety of non-RT sources gives a dynamically fluctuating number of cells over an individual simulation run. This enables the simulation to model the diversity of traffic sources that will be found using an ATM network.

## 4.2 Positioning of the Multiplexer

A series of simulations have been performed to determine if the positioning of the policing function has any effect on the performance for traffic accessing the network. The options are:

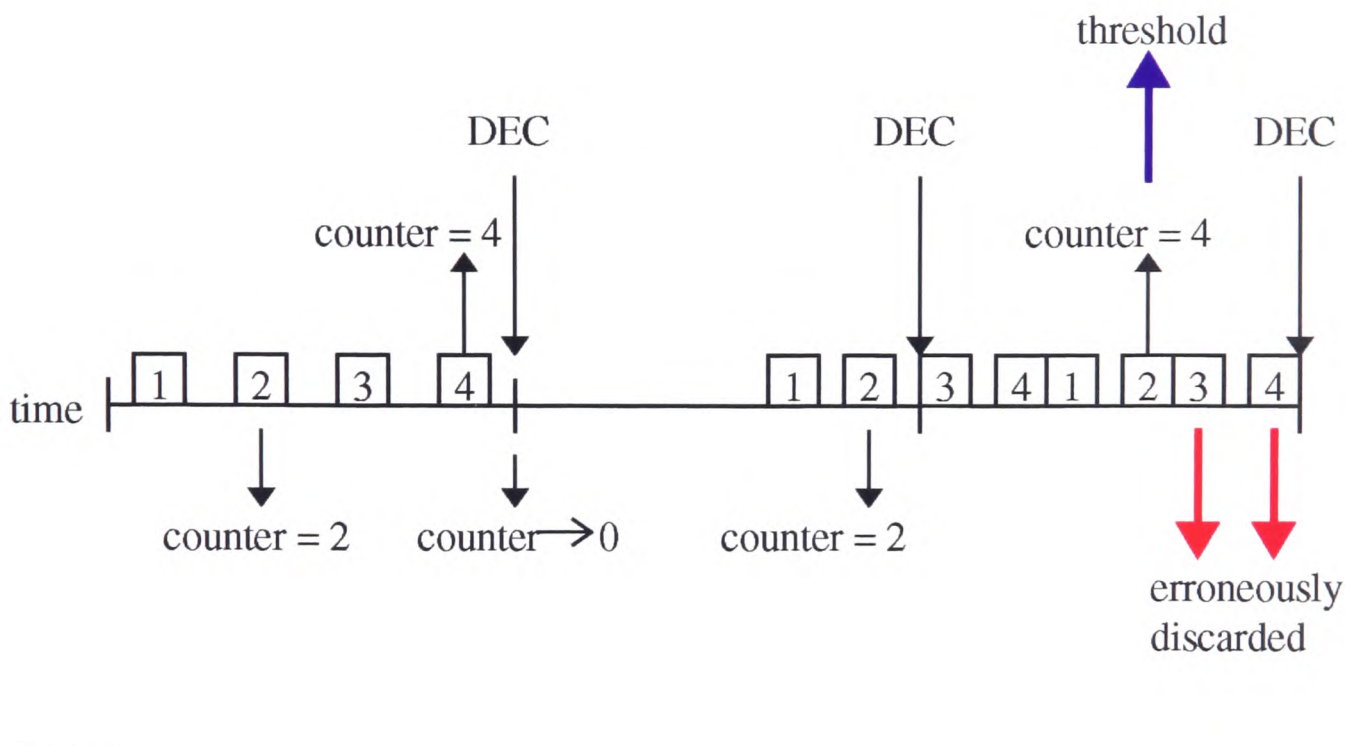
i. cells are passed to the multiplexer and then to the policing function before accessing the network (MuxToLB)

ii. cells are passed to the policing function first and are then queued at the multiplexer before going on to the network (LBtoMux)

The Method 3 has been modified to accept cells onto the multiplexer first and then pass them to the policing function. The only difference between the two models is the positioning of the multiplexer and the VLB, therefore any difference in performance can be attributed directly to this. The statistics for Method 2 are also included for comparison. The thresholds are as indicated in Table 4.2 for Method 3. There are three VBR video sources at 10 Mb/s peak rate, 100 speech sources and 60 data sources at each user site.

The number of RT and non-RT cells generated are identical for both methods, see Table A.1. Cells which are discarded by the multiplexer and the leaky bucket mechanism are counted explicitly. It was found that a number of non-RT cells were dropped by the leaky bucket, when the cells were queued at the multiplexer first and then passed to the VLB. Although the number of cells lost is small (765 cells) in relation to the number generated, this does violate the QoS requirements for Class 1 traffic. The cell loss is due to the CDV caused by the multiplexer queue.

Cells which start off evenly spaced become clumped together and then arrive too quickly one behind the other. The leaky bucket responds by erroneously discarding some cells. The counter in the VLB is decremented periodically until it reaches zero. Since it is not allowed to become negative, "credits" can not accumulate and so are lost. If cells belonging to one decrement period arrive in the following decrement period along with any cells belonging to that period, then the leaky bucket will overflow and cells will be lost, see Figure 4.3.




threshold = 4
DEC = decrement counter

**Figure 4.3 Discarding Cells at the Leaky Bucket**

It was found that the RT queue lengths were the same for both simulations, 0.25 and approximately 7.9 for the mean and maximum respectively.

The non-RT queue had a mean queue length of 1.13 [± 0.3], using Method 3 (LBtoMux). The MuxToLB method had a mean queue length 2.456, which is similar to the queue length found using Method 2, the tagging LB. The maximum queue length using Method 3 was 158, while using the MuxToLB method this increased to 288. This is again comparable with the queue lengths found using Method 2, see Table A.2. The longer queue lengths found may be the result of the server spending more time giving priority service to the RT queue, as cells which would

be discarded by the LB are allowed to pass through the multiplexer first. This gives a performance similar to that found using the tagging LB in Method 2.

RT access delays, video and speech end-to-end delays and the number of low priority speech cells that are dropped are identical regardless of the positioning of the multiplexer, see Tables A.3, A.6 and A.7. The positioning of the policing function also has no effect on the number of video cells discarded (10,895 cells), or on the number of damaged slices (285 damaged slice). The total number of slices generated was 7,558, which means that 3.77% of the slices generated were damaged in both cases, see Tables A.4 and A.5.

The end-to-end delays for data cells are increased using the MuxToLB method, compared to Method 3. The mean delay using MuxToLB is 74.5 μs, compared to 54 μs using Method 3. The maximum delay is 5.3 ms using MuxToLB and 2.9 ms using Method 3. The delays experienced when using the MuxToLB method are comparable with those found using Method 2, see Table A.7.

By placing the policing function before the multiplexer CDV can be reduced and network resources will not be wasted on cells which may be subsequently discarded by the policing function. Some cells which pass through the multiplexer before they are policed may be legitimately discarded by the policing function, particularly in the case of video traffic with the tail-end-clipping function. The video cells which are discarded by the tail-end-clipping function can also cause the RT queue at the multiplexer to increase in length if these cells are passed to the multiplexer first. This could have an impact on the low priority speech cells, which could be discarded unnecessarily by the multiplexer, if the RT queue was to exceed the threshold ($RT_2$).

Any cells which are delayed excessively in a multiplexer queue can cause the policing function to perceive them as violating cells and subsequently discard them. Alternatively, cells that have been queued can become "bunched up" and lose their smoothness, which causes them to arrive too rapidly in succession and also be discarded.

The results from the first set of simulations clearly indicate that there are advantages to placing the policing function before the multiplexer. The non-RT cells are at an increased risk of being discarded by the policing when they pass through the multiplexer first.

## 4.3 Results - 20 Data Sources

A series of simulation experiments have been performed with the LB positioned before the multiplexer, as proposed in Section 4.2, for each of the methods outlined in Table 4.1. Each was run with a maximum of 20 data sources and 100 speech sources allowed. The peak rate for video was 10 Mb/s and there were three video sources active throughout the simulation, at each user site. The performance of the different methods (1 to 7) were compared.

Statistics have been gathered at the multiplexer for both RT and non-RT cells, and for individual traffic sources. The number of cells discarded, by both the multiplexer and the policer, the queue lengths and the delays to receive service at the multiplexer are collected. The access delays, queue lengths and cell losses experienced by cells from user site 2 are comparable with those from user sites 1 and 4.

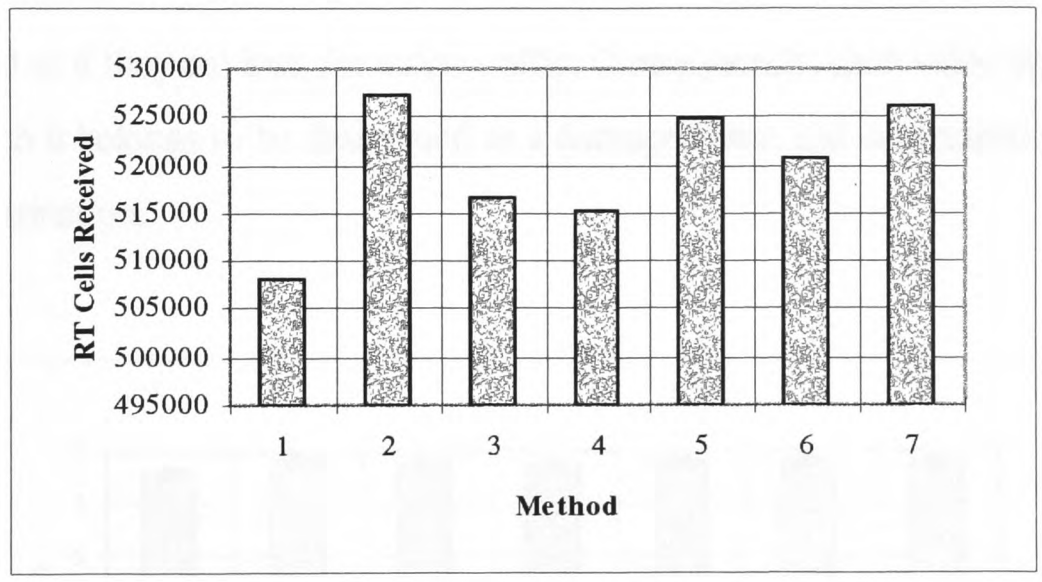



**Figure 4.4 RT Cells Received - 20 Data Sources**

The number of cells generated by and received from each user site are monitored for RT and non-RT traffic separately. The number RT cells generated was 527,233 for each 20 second reporting interval, see Table A.8. The number of non-RT cells generated during the same period was 326,828, which was also the same number as those received, as there were no non-RT cells lost. The number of RT cells received varied according to the method used, see Figure 4.4, with Method 2 having the largest number received, since no cells are deleted, only tagged.

The percentage of RT cells generated was 62% and 38% were non-RT cells, see Table A.10. This was initially implemented to ensure that the RT queues were large enough to trigger the various thresholds and to test the strategies used. The quality of service received by the video traffic, which is of particular interest, could then be observed.

The utilisation at the access link depends on the number of cells that pass through the multiplexer. Utilisation is fairly constant across all methods at 40% [± 1.06%], see Table A.9. The utilisation is marginally higher for Method 2, as no cells are discarded by this method and also for Method 7, which allows longer bursts of video cells onto the network.

Method 2 has the highest number of RT cells received. This is because no cells are discarded by the leaky bucket, so a higher proportion arrive at the destination. As there is little congestion in this network, the cells which have been tagged as low priority do not get dropped as they cross the network, during the simulation. However, there is always the possibility that they could be arbitrarily discarded as they cross the network, if there is a higher level of congestion present, so they are counted as if they are lost, for video traffic. Consequently, each video cell tagged causes the slice to which it belongs to be designated as a damaged slice and is assumed to be useless on arrival at the destination.

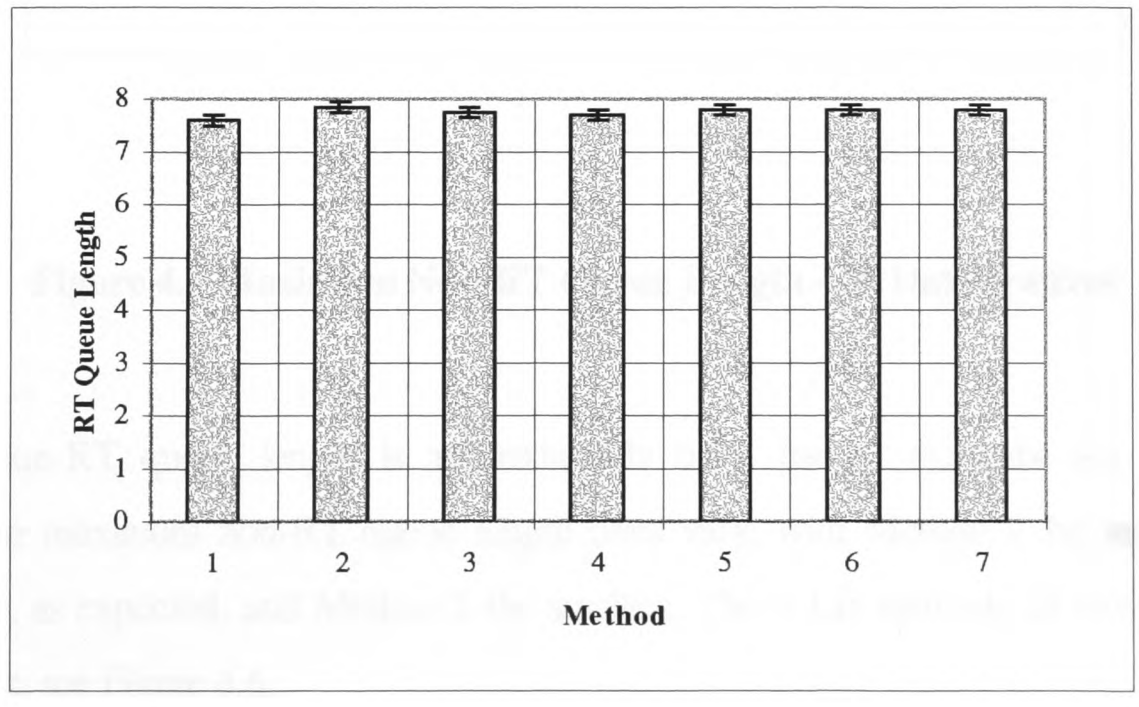



**Figure 4.5 Maximum RT Queue Length - 20 Data Sources**

The mean RT queue length fluctuates around 0.18 [± 0.003] for all methods. The maximum RT queue length is marginally larger for Method 2, while Method 1 has the smallest maximum queue length, see Figure 4.5. These differences between the different methods are slight.

The non-RT queue length is directly influenced by the size of the RT queue. If the RT queue length exceeds the first threshold, then the server begins to give priority service to the RT queue. This means that the non-RT queue receives no service at all, until the RT queue falls below the threshold again. The non-RT queue will continue to grow as cells join the queue at the multiplexer.

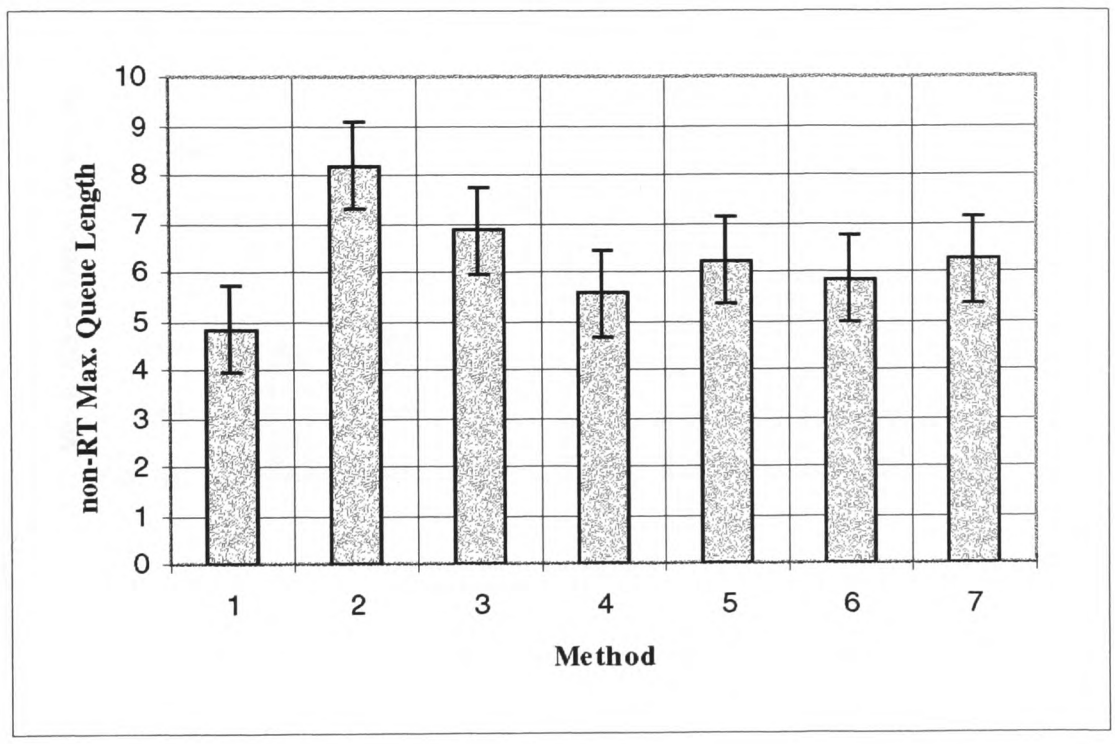



**Figure 4.6 Maximum Non-RT Queue Length - 20 Data Sources**

The mean non-RT queue length is approximately 0.07, for all methods, see Table A.12. However, the maximum non-RT queue length does vary, with Method 2 having the longest queue length, as expected, and Method 1 the smallest. The S-LB methods all have very similar queue lengths, see Figure 4.6.

The access delay is a measure of the time that a cell waits in the queue at the multiplexer until it is served and allowed onto the network. This is a variable delay, as discussed in Section 2.2.5.3,

and depends on the number of cells already waiting in the queue. It can be seen in Table A.13 that the RT access delays are fairly constant across all methods with the mean being approximately 7.4 μs and the maximum approximately 67 μs for RT traffic.

The mean non-RT access delays are increased for the super leaky bucket methods, as can be clearly seen in the graph in Figure 4.7. This is increased from approximately 4 μs, for the Methods 1 to 3, to approximately 13.5 μs for Methods 4 to 7, see Table A.14.

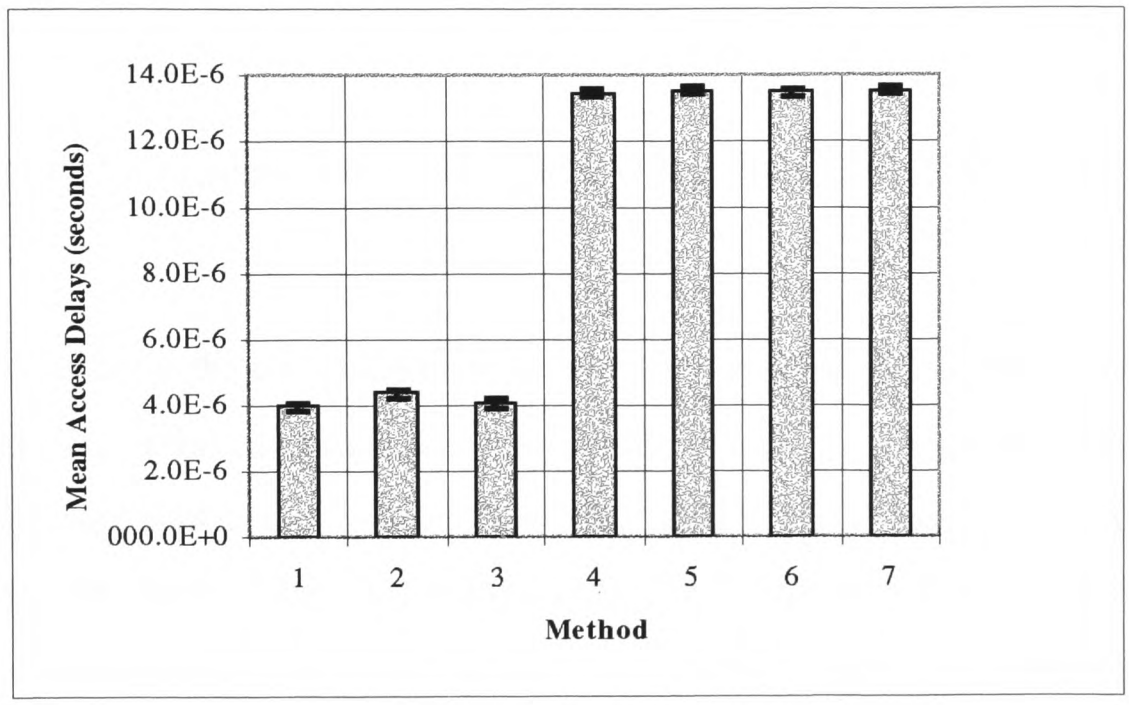



**Figure 4.7 Mean non-RT Access Delays (in sec) - 20 Data Sources**

The largest maximum delays experienced by non-RT cells accessing the multiplexer are found using Methods 5 and 7, with delays of approximately 286 μs. Method 2 has a maximum delay of 271 μs and for Method 6 the delay is 263 μs. The access delay for Method 4 is 236 μs while Method 1 has the smallest delay at 169 μs, see Table A.16. At this level of utilisation excessive delays are not imposed on the non-RT cells by any of the S-LB methods, see Figure 4.8.

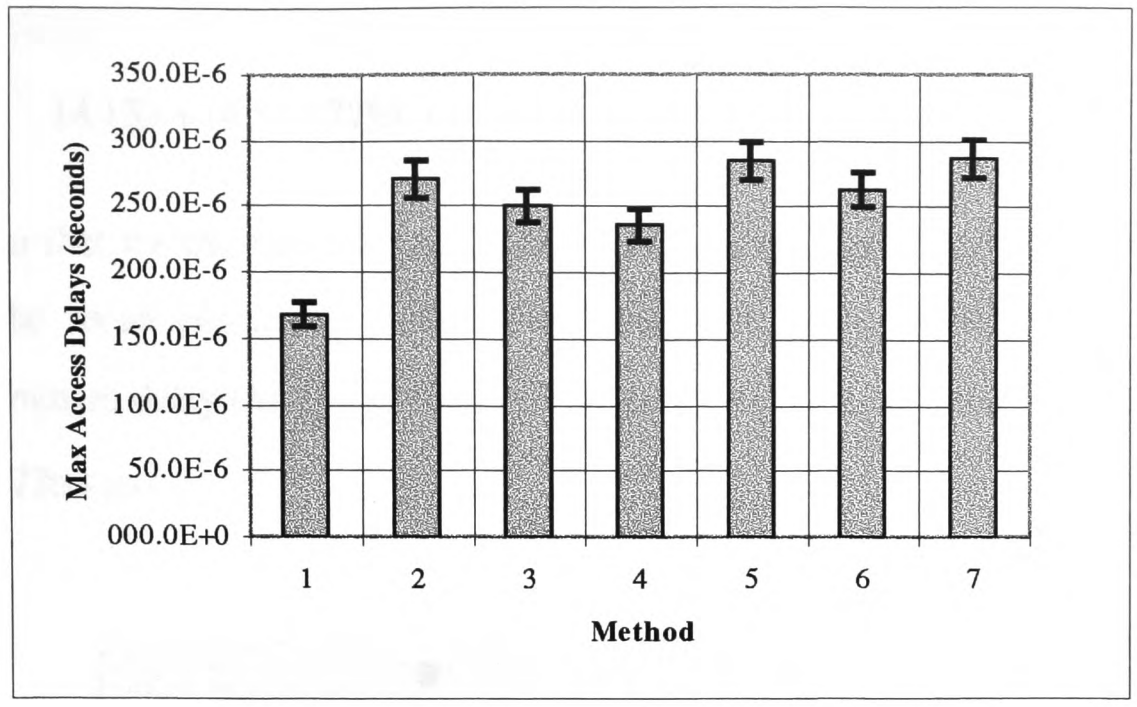



**Figure 4.8 Maximum non-RT Access Delays (in sec) - 20 Data Sources**

The end-to-end delay is the time it takes for a cell to travel from the originating user site to the destination site. The delay characteristics of an ATM network are explained in Section 2.2.5.3. The cells from user sites 1 and 4 which travel across all the switches in the network, to the destination user site 3 are used to measure the end-to-end delays across the network. Cells from user site 2 pass through only two switches and are all destined for site 3. The total delay for a cell does not include the packetisation delay or the propagation delay between switches. It does however, include a fixed switching delay as the cell is switched across the switching fabric, at each switch, and a service (transmission) delay at each server (multiplexer and switches). The service time at the source multiplexer is 9.422 μs for an access link operating at 45 Mb/s. The switching and service delays at each switch have been fixed at 2.726 μs each. For cells travelling through four switches this represents a total fixed delay of:-

$$9.422 + (4 * (2.7263 + 2.7263)) = 31.2324\ \mu s \qquad \text{----(4.5)}$$

However, ATM networks have a slotted structure, and a cell arriving at a multiplexer must wait for the start of the next available slot for service. It has been noted during this work, that for this type of network, a cell accessing an empty queue will, on average, wait the duration of half a service time. Assuming that the wait at the multiplexer is 1.5 * service duration = 14.133 μs, then the total minimum expected delay for a cell crossing the network becomes

$$14.133 + (4 * (2.7263 + 2.7263)) = 35.9434\ \mu s \qquad \text{----(4.6)}$$

It can be seen that for the non-RT cells, which have the smallest associated variable queuing delay, that the mean end-to-end delays experienced are only marginally greater than the calculated minimum delay (37.2 - 35.9434 = 1.2566 μs) which is less than the duration of one ATM slot (2.7263 μs).

| | Simulation delay | Calculated delay | Difference |
|---|---|---|---|
| 4 switches | 37.2 μs | 35.94 μs | 1.2566 μs |
| 2 switches | 25.8 μs | 25.038 μs | 0.7618 μs |

**Table 4.3 Comparison of End-to-End Delays**

The mean and maximum end-to-end delays for RT cells remain constant across all methods and at all sites, see Table A.15. This indicates that the delays for RT traffic are unaffected by the policing or access strategy at the multiplexer. Even more importantly, it does not have a detrimental effect on the delays experienced by speech and video traffic.

The peak bit rate for video is 10 Mb/s and the actual mean bit rate recorded during the simulations was approximately 3.4 Mb/s. The effective bit rate, which was used to during the allocation of bandwidth by the CAC, was 4.65 Mb/s, see Section 3.1.3.2. This means that bandwidth was not reserved at either the peak or the mean rate. This was also the bit rate used to dimension the S-LB methods 4 and 6. Using this over-dimensioning function provides a reasonable performance without wasting too much bandwidth. It also means that sufficient bandwidth was allocated for video traffic to ensure a reasonable QoS.

The number of video cells generated by each method is the same for each of the user sites. However, the numbers of cells discarded is significantly reduced for all of the S-LB methods. The largest number of discarded cells occurs with Method 1 with 5% of total generated lost. Since these cells are spread throughout many slices, it also results in the largest number of damaged slices, see Figure 4.9. The smallest number of discarded video cells occurs with Methods 5 and 7 which have 0.64% and 0.25% of the cells discarded respectively, see Table 4.4.

These cells are also compacted into the smallest number of damaged slices by these methods, 0.83% and 0.3% of total slices respectively. Methods 2 and 3 both have 3.67% damaged slices, respectively. Method 4 has 3.5% and Method 6 has 1.82% damaged slices out of a total of 7,539 slices generated.

| Method | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Cells generated | 377576 | 377576 | 377576 | 377576 | 377576 | 377576 | 377576 |
| Cells discarded | 19068 | 10485 | 10485 | 12110 | 2415 | 6429 | 960 |
| Percentage | 5.05% | 2.78% | 2.78% | 3.21% | 0.64% | 1.70% | 0.25% |
| | | | | | | | |
| Slices generated | 75389 | 75389 | 75389 | 75389 | 75389 | 75389 | 75389 |
| Slices damaged | 914.64 | 276.5 | 276.5 | 264.12 | 62.53 | 137.06 | 22.86 |
| Percentage | 12.13% | 3.67% | 3.67% | 3.50% | 0.83% | 1.82% | 0.30% |

**Table 4.4 Video Cells and Slices Damaged - 20 Data Sources**

Method 4 did discard a larger number of video cells than Methods 2 and 3, however, those cells are compacted into fewer damaged slices due to the tail-end-clipping function.

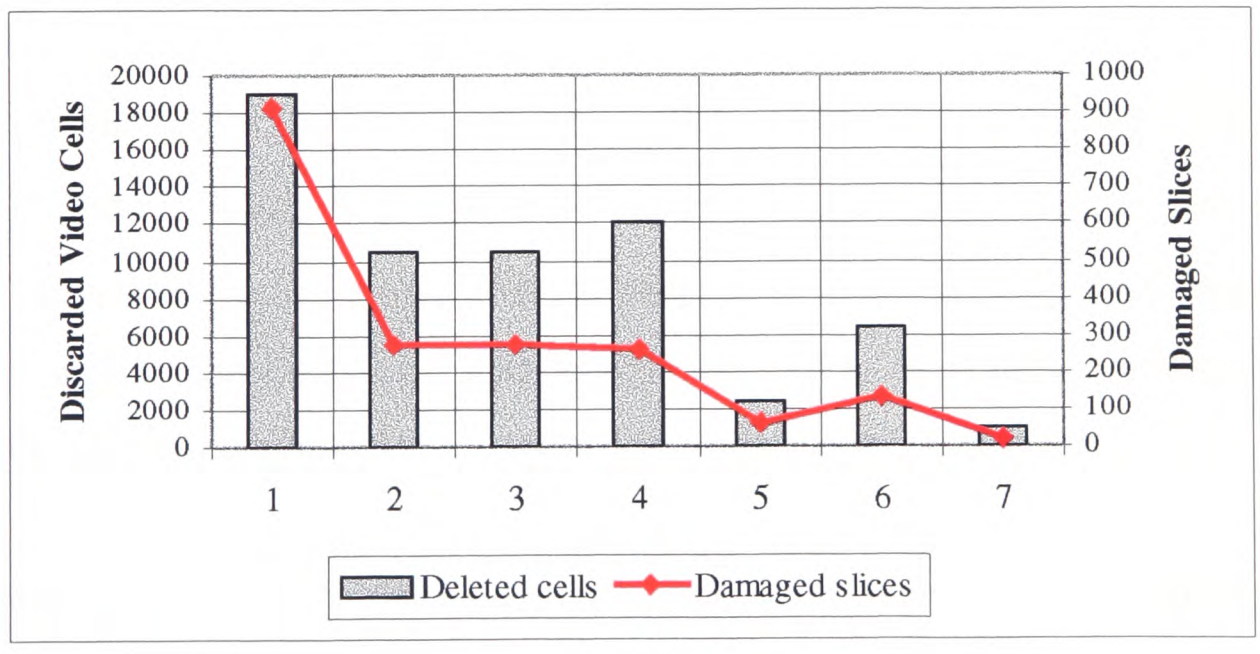



**Figure 4.9 Damaged Video Slices Vs Discarded Video Cells - 20 Data Sources**

In the case of Method 2 the number of cells generated is the same as the number of cells received, as the cells are not discarded by the leaky bucket, only tagged. If any tagged cells are

discarded as they cross the network, damaged slices could result. Tagged cells are therefore counted in the same way as discarded ones, and the slices that they belong to are considered damaged. It was noted that a small number of tagged video cells were dropped at the multiplexer by this method when the low priority dropping mechanism was triggered. As previously stated, a damaged slice is defined as any slice which contains a discarded or tagged video cell. The total number of slices generated is the same across all methods, while the number damaged varies greatly, see Table A.18 and Figure 4.9.

Similar numbers of speech cells are generated at each of the three user sites. Even though the multiplexer is allowed to drop low priority cells when the queue length grows beyond the threshold, 99.9% of all speech cells generated are received at the destination. The actual number of speech cells dropped is minimal, less than 0.01%, and fairly constant over all methods, see Table A.20

The speech cells lost are all discarded at the multiplexer so the length of the RT queue is a significant factor in the performance as seen by speech cells. The low priority dropping mechanism is activated when the RT queue length exceeds 6 cells in length. As can be seen in Figure 4.5, the maximum queue lengths do go above this threshold, causing some low priority speech cells to be discarded.

The mean and maximum end-to-end delays for speech are 44.6 μs [± 0.16 μs] and 98 μs [± 0.6 μs] for all methods. Video cells experience mean end-to-end delays of 39 μs and maximum delays of 95 μs. These delays remain fairly constant across all the methods, see Tables A.19 and A.21 The slightly longer delays for speech cells compared to video cells is attributable to the way that the speech cells are packetised and this is discussed more fully in Section 4.8

The numbers of non-RT cells generated are the same as the number received, for all methods. This fulfils the Class 1 traffic QoS requirement that no cells are lost. The queue lengths at the multiplexers, for data cells, are not excessive and within reasonable bounds at this level of utilisation, see Figure 4.6 and Table A.12. The largest maximum queue length (8.17) occurs using Method 2. This is because this method discards no cells and allows the largest number of cells to access the network.

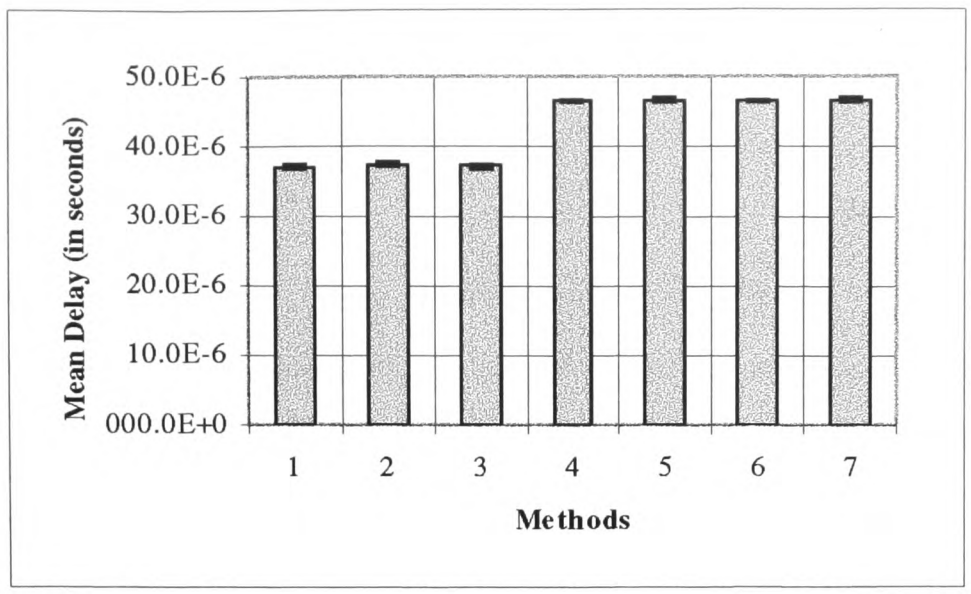



**Figure 4.10 Mean End-to-End Delays for Data Cells (in sec) - 20 Data Sources**

Methods 1, 2 and 3 all have comparable mean delays at approximately 37 μs [± 0.15 μs]. The mean end-to-end delays for data cells are increased by approximately 9 μs to 46.6 μs for all the super leaky bucket methods, compared to the VLB methods, see Figure 4.10. Priority service is given to the RT queue and this has the effect of increasing the delays in the non-RT queue, which has a significant impact on the overall end-to-end delays experienced.

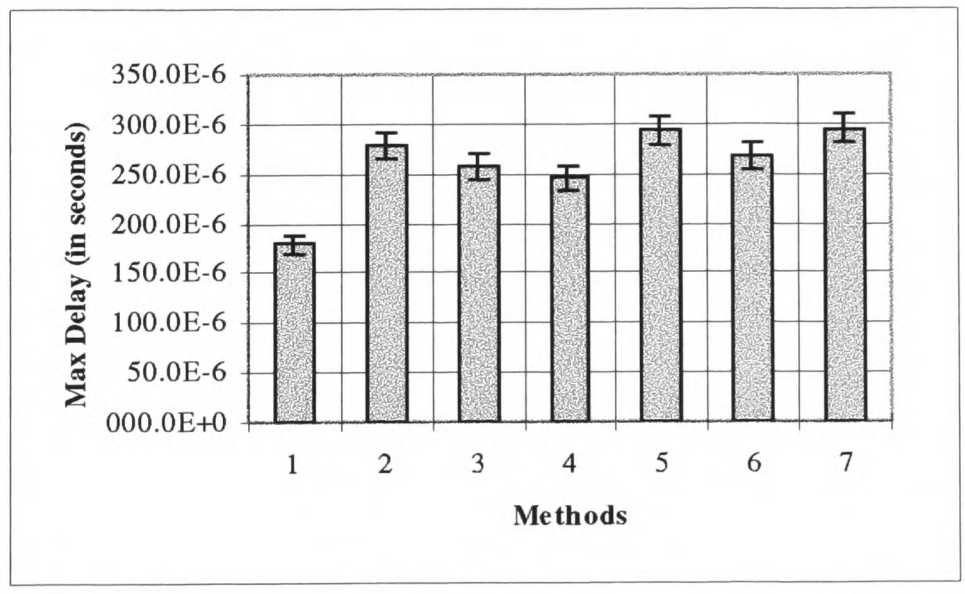



**Figure 4.11 Maximum End-to-End Delays for Data Cells (in sec) - 20 Data Sources**

Methods 5 and 7 both have the largest maximum end-to-end delays for data cells at approximately 294 μs. Method 6 has a maximum delay of 267 μs and Method 2 has a marginally

worse delay of 279 μs. Method 4 shows a maximum delay of 246 μs while Method 1 again has the smallest maximum delay of 179 μs for data cells.

Using the peak rate to dimension the leaky bucket reduces the number of cells lost and hence the number of damaged slices. Tail-end-clipping is also in operation at all the super leaky buckets. It can clearly be seen that even when a large number of cells are discarded, the impact of these can be minimised by using tail-end-clipping. When the peak rate and twice the mean burst length (Method 7) are used to dimension the leaky bucket, it causes the least number of cells to be discarded and also damages the least number of slices (0.3%). This implies an improved quality of service over the other methods examined in this work. The number of discarded video cells rises, along with the number of damaged slices (1.7% of cells discarded, 1.82% of slices damaged), when the effective bit rate is used to dimension the leaky bucket. However, this is still an improvement over the VLB methods (10,485 discarded cells, 277 damaged slices), which do not use the tail-end-clipping function.

It was clearly seen from the results that the super leaky buckets discarded the least number of cells, and that those losses were concentrated into the smallest number of video slices, with Method 7 giving the best performance for video. The worst performance, with the greatest number of discarded cells across the largest number of video slices was given by the reference VLB, Method 1, using the original parameters proposed by [NIEST90].

## 4.4 Results - 40 Data Sources

The third set of simulations have the same number of RT traffic sources as the previous simulations, i.e. 3 video and maximum of 100 speech calls at each user site. The number of non-RT sources was increased to 40 at each user site, which doubles the maximum number of non-RT sources allowed. The performance of the different methods used was compared to see if the increased number of non-RT sources would have any adverse effect on the RT traffic. The mix of traffic has now changed and the percentage of cells generated was 45% RT and 55% non-RT cells.

The number of RT cells generated was comparable with those found in Section 4.3, see Table A.8, while the number of non-RT cells has nearly doubled to 658,537 cells. However, none of

the non-RT cells were dropped at either the LB or the multiplexer, for all methods. There were however, a small number of data cells blocked at source by the multiplexer queue management by the S-LBs. The smallest number of blocked data cells was with Method 4 (130 data cells). Methods 5 and 6 blocked 200 cells and 185 cells respectively. Method 7 blocked the largest number of data cells, which was 229. These were cells blocked due to the throttle-back mechanism being activated when the multiplexer queue grows larger than threshold $T_2$.

The number of RT cells received is very similar to those in the 20 data sources simulation.

The utilisation at the access link has increased from 40 % to 55 % for all methods, [± 1.85 %], see Table A.9.

The mean RT queue length is 0.23 and the maximum queue length is 7.8, which is very slightly increased over the previous simulation results in Section 4.3.

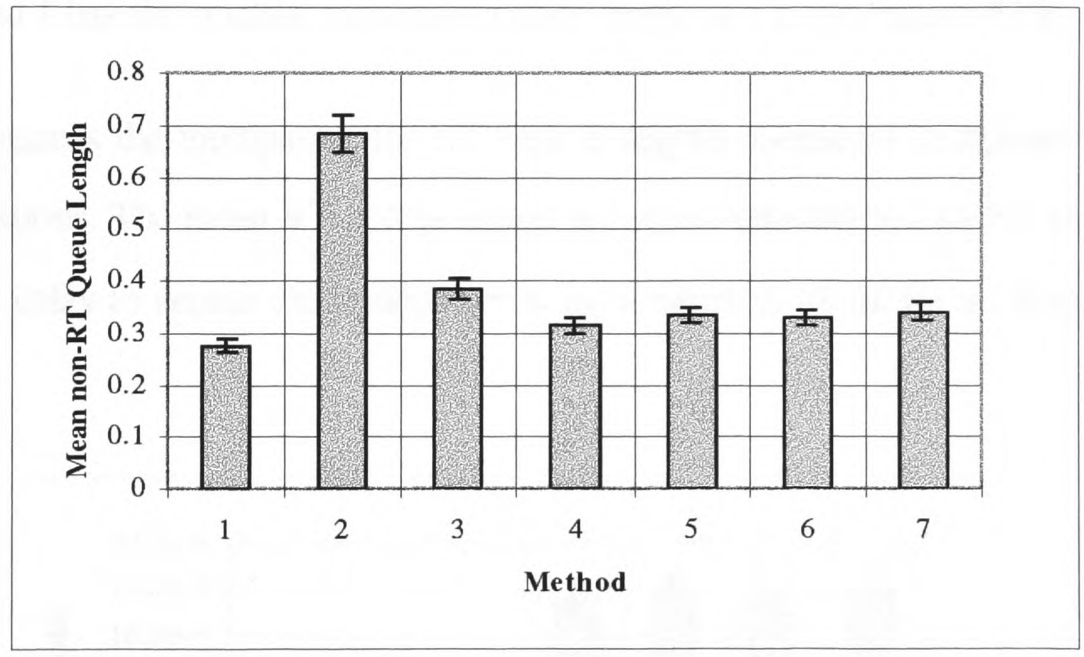



**Figure 4.12 Mean non-RT Queue Length - 40 Data Sources**

The mean non-RT queue length has increased to approximately 0.32 for Methods 4, 5 to 7. Method 3 has a queue length of 0.39. Figure 4.12 also shows that Method 2 has the largest mean queue length of 0.68 and Method 1 has the smallest mean queue length of 0.28 cells. The mean non-RT queue length remains small across all the methods.

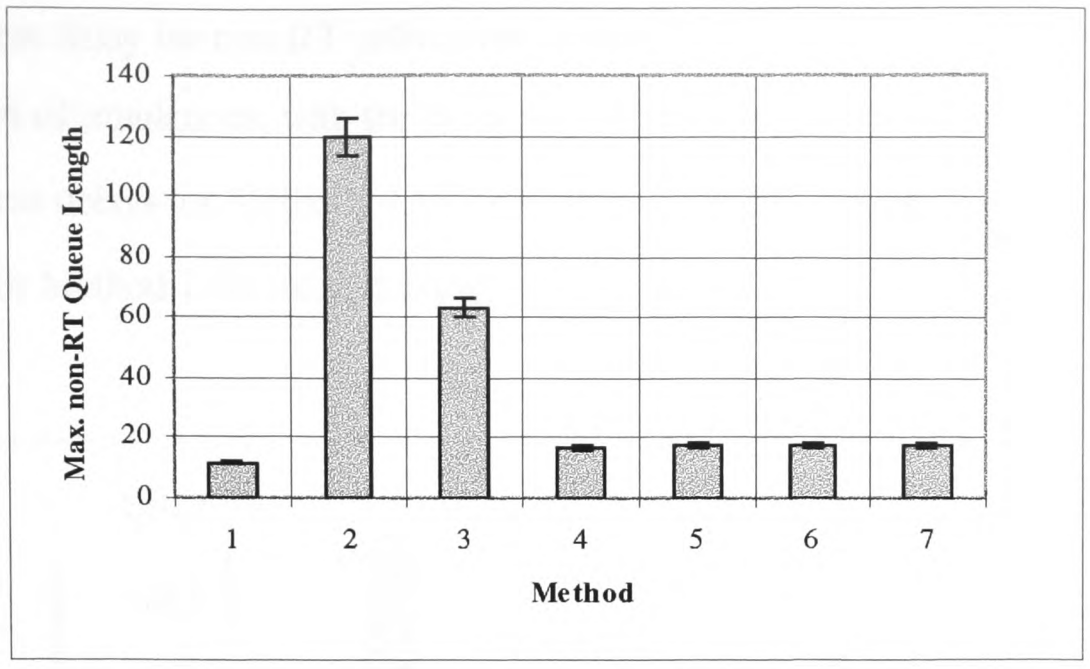



**Figure 4.13 Maximum non-RT Queue Length - 40 Data Sources**

The maximum non-RT queue length is approximately 17 for the S-LB methods (4 to 7). Method 2 has the largest maximum queue length of 120 compared to 63 for Method 3. The reference LB used in Method 1 has the smallest maximum queue length at 12, see Figure 4.13.

The delay to access the multiplexer for RT cells is slightly increased compared to the 20 data sources simulations. The mean RT access delays are approximately 9.2 μs for all Methods. The maximum RT delay to access the multiplexer is approximately 69 μs for all methods, see Table A.13.

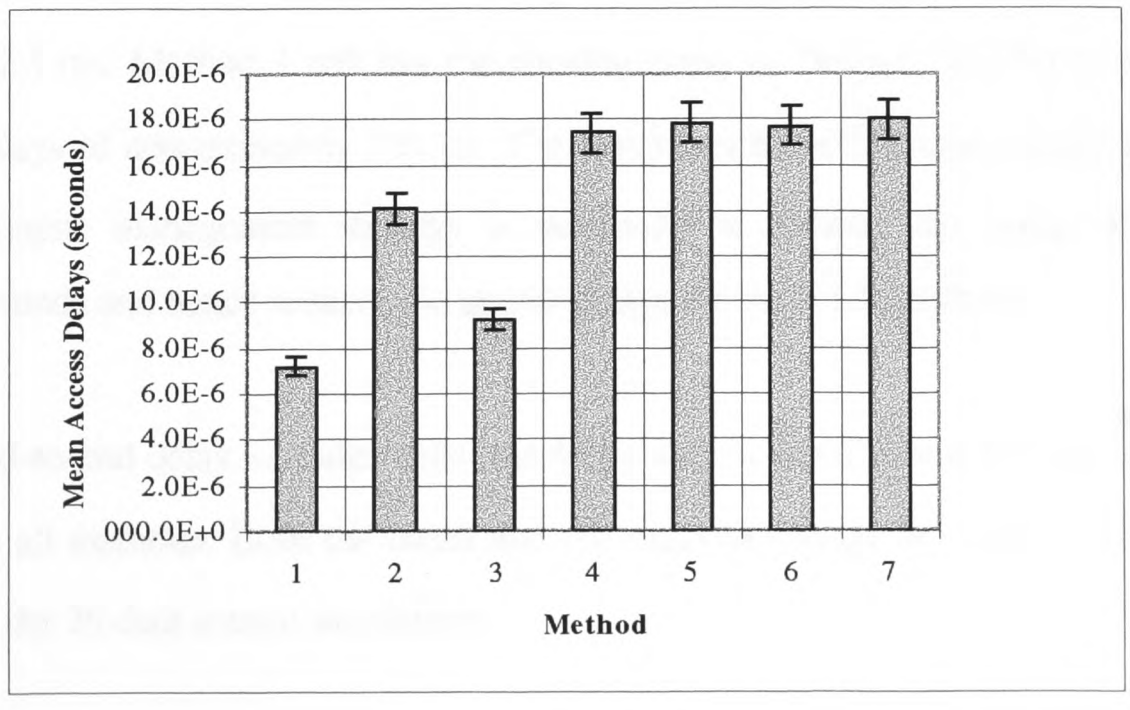



**Figure 4.14 Mean non-RT Access Delays (in seconds) - 40 Data Sources**

The mean access delay for non-RT cells at the multiplexer follow a similar trend as that seen in the previous set of simulations, with the exception of Method 2, which had increased to 14.1 μs. The mean access delays for Methods 4 to 7 are approximately 17.8 μs. For Method 3 the delay is 9.2 μs and for Method 1 the mean delay is 7.2 μs.

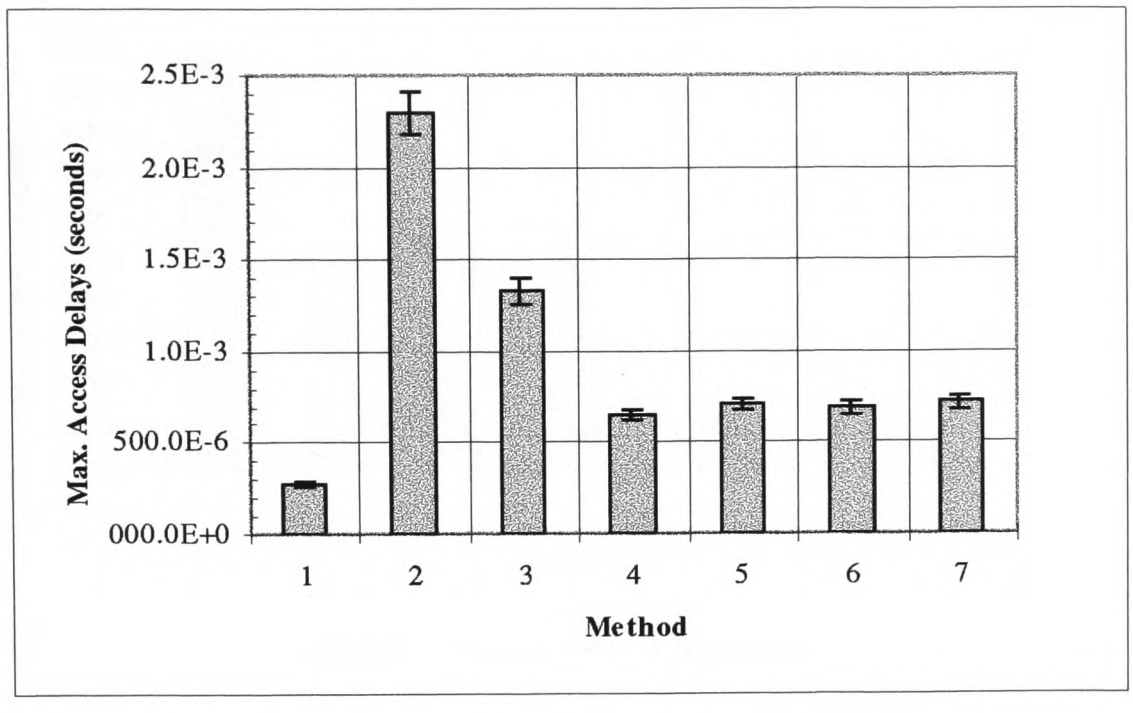



**Figure 4.15 Maximum non-RT Access Delays (in seconds) - 40 Data Sources**

Method 3 has a maximum non-RT access delay at the multiplexer of 1.33 ms and for Method 2 the delay is 2.3 ms. Method 1 still has the smallest delay at 269 μs. The S-LB methods have maximum delays of approximately 700 μs. The maximum non-RT access delays show that the multiplexer queue management strategy is successful in keeping the queue lengths within reasonable bounds and hence reduces the access delays for the S-LB methods.

The mean end-to-end delay for video cells was 41 μs and the maximum delay was approximately 97 μs across all methods. Both the mean and the maximum delay are slightly increased when compared to the 20 data source simulations.

The number of video cells generated and the number discarded was comparable to those found in the 20 data sources simulations. This also applied to the number of damaged slices which were

of similar magnitude. There has been no significant increase in the number of damaged slices, compare Figures 4.9 and 4.16. There were however a very small number of tagged video cells dropped at the multiplexer by Method 2. This method tags all violating cells and allows them on to the network. At high loading these cells may be discarded.

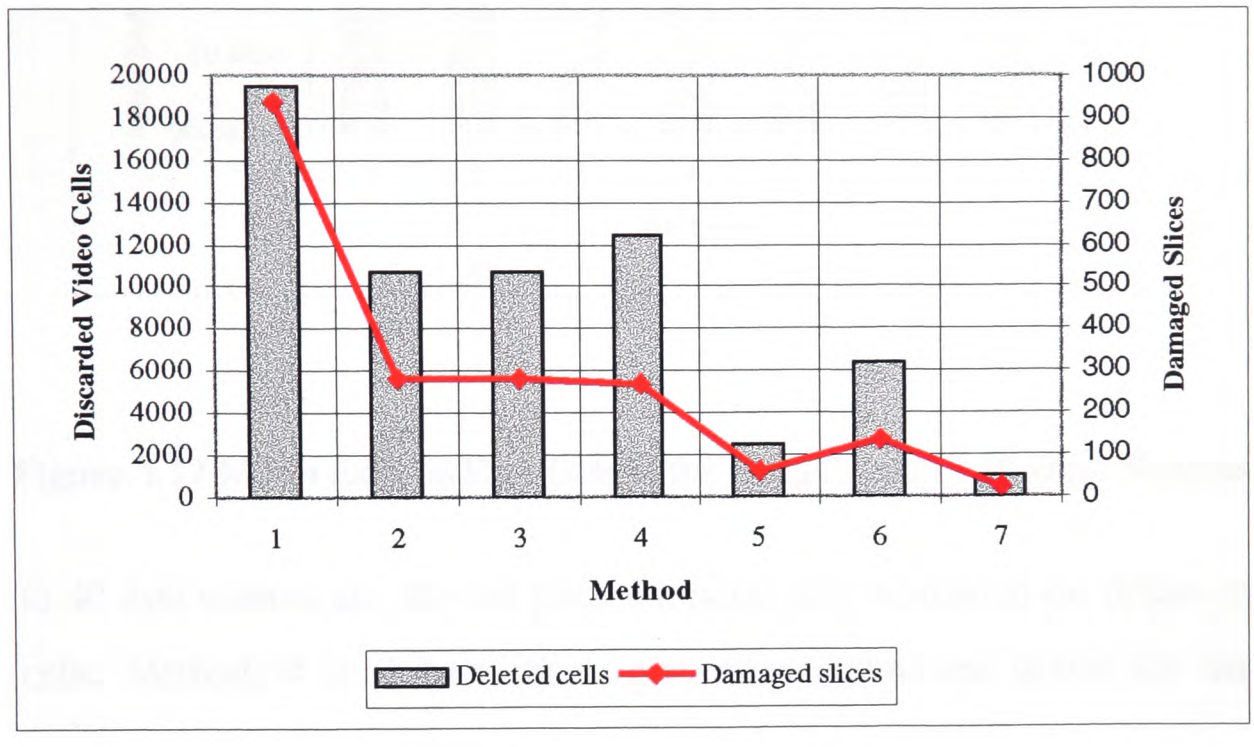



**Figure 4.16 Damaged Video Slices and Cells - 40 Data Sources**

The number of speech cells generated was similar to those in the previous simulation. The number of cells dropped at the multiplexer had increased very slightly. The end-to-end delays for speech cells were approximately 47 μs and 100 μs for the mean and maximum respectively. This is an overall increase of 2 μs over the previous simulation (20 data sources), which is negligible.

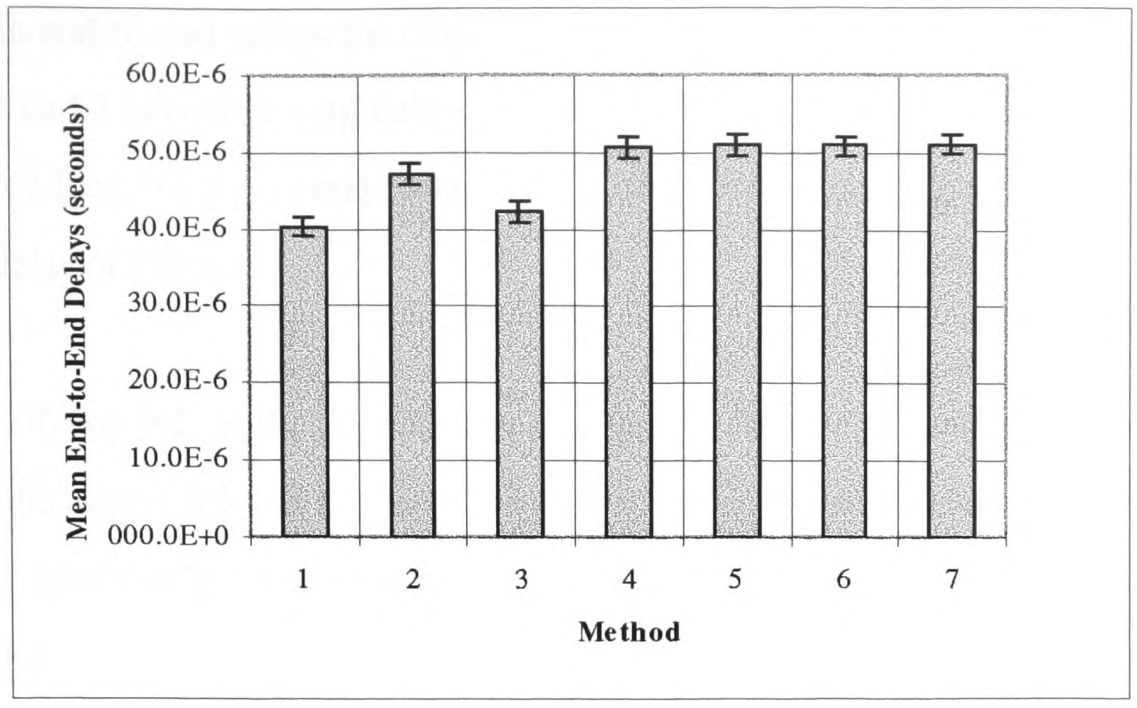



**Figure 4.17 Mean End-to-End Delays for Data (in sec) - 40 Data Sources**

When up to 40 data sources are allowed there is a noticeable increase in the delays experienced by these cells. Methods 4 to 7 have the largest mean end-to-end delays for data cells at approximately 51 μs, see Figure 4.17. This follows the same basic trend as the mean access delays, since the delay to access the multiplexer has a significant influence on the overall delays experienced. Method 1 has a mean delay of 40 μs, while Methods 2 and 3 have delays of 47 μs and 42 μs respectively.

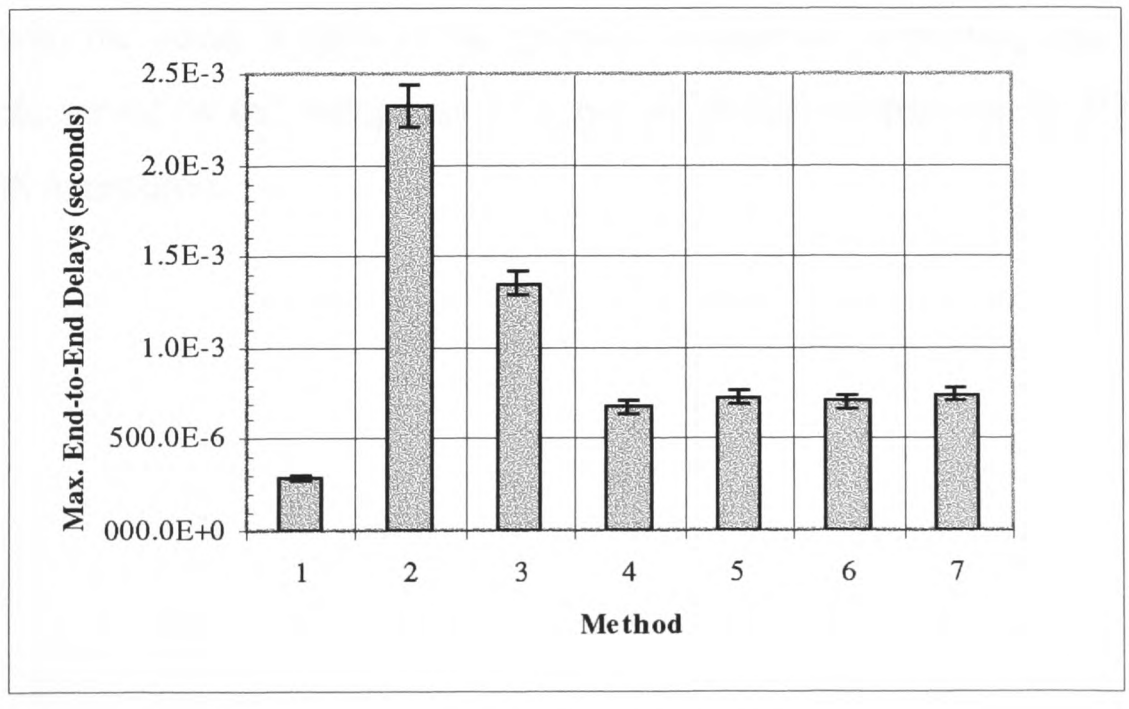



**Figure 4.18 Maximum End-to-End Delays for Data (in sec) - 40 Data Sources**

The maximum end-to-end delays for data cells clearly shows that the unmanaged queues found in Methods 2 and 3 introduce long delays at the multiplexer, 2.33 ms and 1.35 ms, respectively. Methods 4 to 7 have maximum end-to-end delays of approximately 0.7 ms. Method 1 again has the smallest delay of 290 μs.

The number of non-RT traffic sources has doubled in this set of simulations and hence the utilisation at the access link has increased. However, the delays experience by the RT cells have not increased significantly, indicating that the RT cells are relatively unaffected by the increased non-RT queues.

## 4.5 Results - 60 Data Sources

The final set of simulations were run with the same number of RT traffic sources and 60 data sources. The numbers of RT cells generated was as before. The number of non-RT cells had increased to 882,095. This meant that the RT component of the traffic mix was 37 % and the non-RT was 63 %. This caused the utilisation at the access link to increase to approximately 66% [± 2.8 %].

The mean RT queue length was 0.25 [± 0.006]. The maximum queue length was approximately 7.8 for methods 1, 3, 4, and 5 and was 7.9 using Method 2, 6 and 7, see Table A.11. This is comparable with the queue lengths in the previous simulations, indicating that the increased number of cells served by the multiplexer does not significantly impact on the RT cells, which have their QoS maintained.

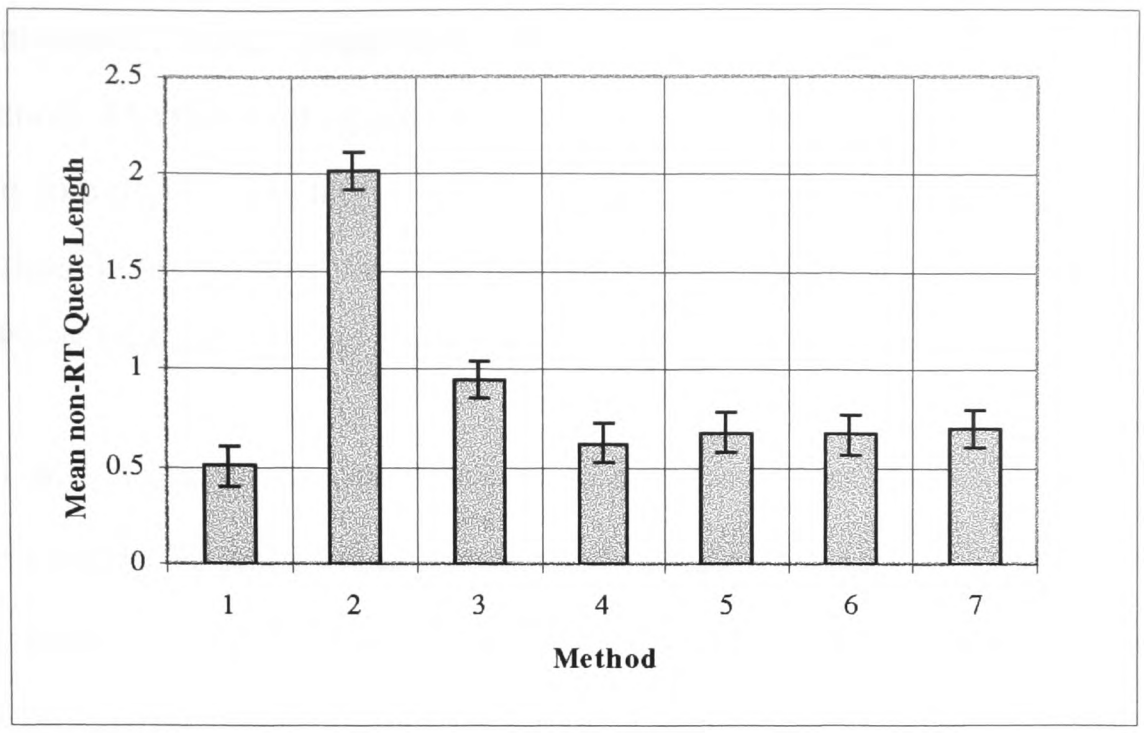



**Figure 4.19 Mean non-RT Queue Length - 60 Data Sources**

The mean non-RT queue length has increased to 2 for Method 2, however Method 1 still gives the lowest mean queue length at 0.5. Method 3 has the next longest mean queue length at 0.95. The S-LB methods show the same general trend as in the 40 data source simulations, with Method 4 having the smallest mean queue length at 0.62 and Method 7 having the longest at 0.69. This is approximately double that found in Section 4.4.

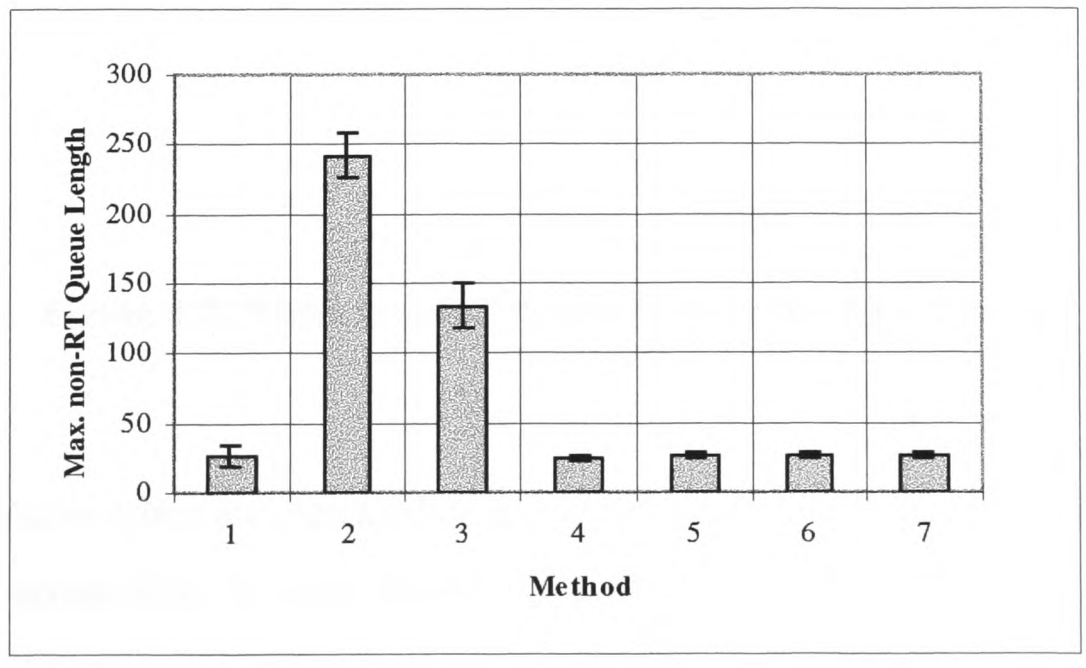



**Figure 4.20 Maximum non-RT Queue Length - 60 Data Sources**

The maximum non-RT queue length also follows the same pattern as that found in the 40 data source simulation. The maximum queue length for all the S-LB methods was 26, which was very similar to that found using Method 1, which was 27. Method 2 had the worst queue length at 241, and Method 3 was the next worst at 134. This indicates that the S-LB gives the same level of performance as Method 1 at this level of utilisation.

The mean RT access delays are approximately 10.2 μs [± 0.25 μs] and the maximum RT access delay is approximately 70 μs [± 0.6 μs] for all methods. This shows that keeping the non-RT and RT cells in separate queues does benefit the RT cells, which experience a minimal increase in their access delays, even though the utilisation has increased from 40% to 66%. The RT cells are protected from long delays at the multiplexer.

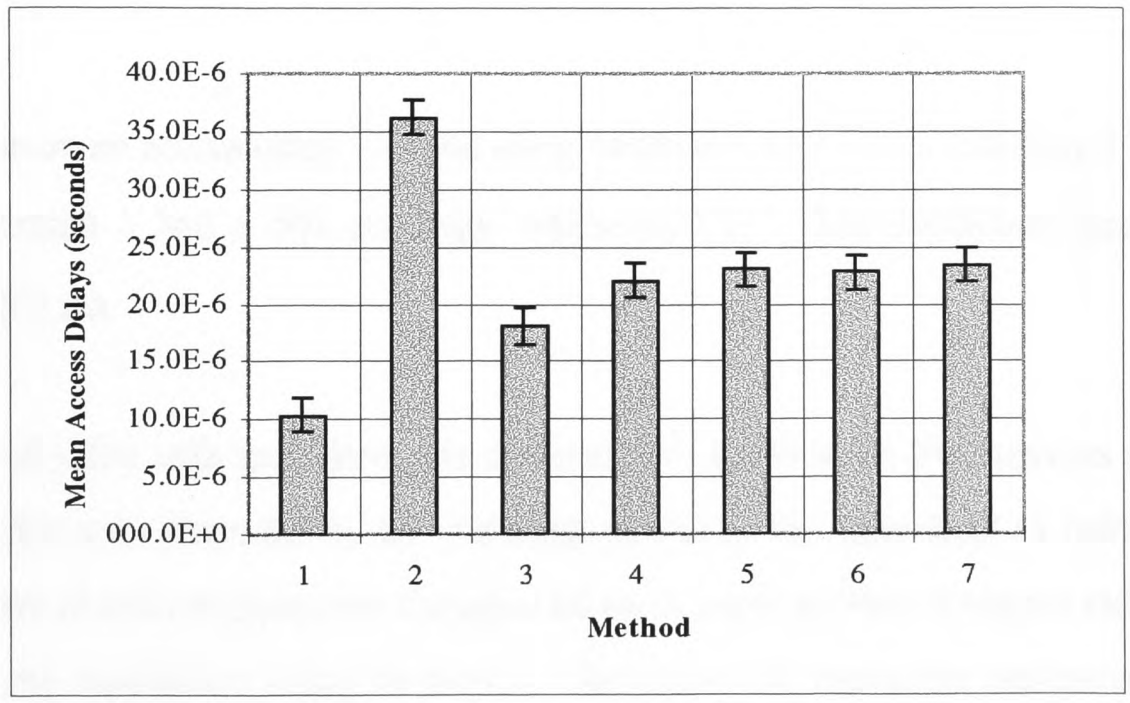



**Figure 4.21 Mean non-RT Access Delays - 60 Data Sources**

The non-RT access delays are increased as would be expected with longer queue lengths. The mean non-RT access delay is approximately 23 μs for the S-LB Methods and 10.4 μs using Method 1. The VLB methods 2 and 3 have mean access delays of 36 μs and 18 μs respectively.

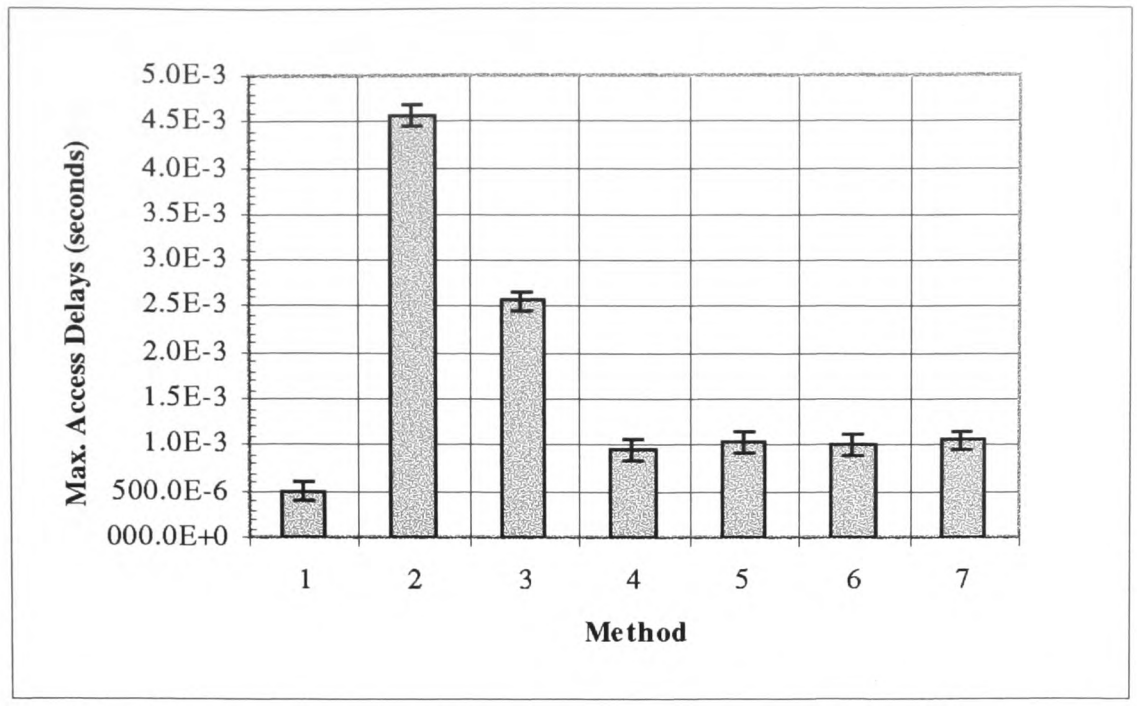



**Figure 4.22 Maximum non-RT Access Delays - 60 Data Sources**

The worst maximum access delay is found using Method 2, at 3.62 ms. Method 3 had a 1.8 ms delay and Method 1 had a 501 μs delay. Methods 4 to 7 had maximum access delays of approximately 1 ms.

The number of video cells generated was comparable with those in the previous sections. The number of video cells discarded by tail-end-clipping was at the same level as before, with very similar numbers of cells dropped and damaged slices. A small number of tagged video cells were discarded at the multiplexer using Method 2. The mean and maximum end-to-end delays for video cells were also very similar to those found in the 40 data source simulation results at approximately 41 μs and 97 μs respectively.

The number of speech cells dropped was also comparable to those found in the previous section, (4.4). The mean end-to-end speech delays were very slightly increased to 48.3 μs for all methods. The maximum end-to-end delays were comparable with those in the 40 data sources simulations, at 101 μs, [± 0.8 μs] for all methods.

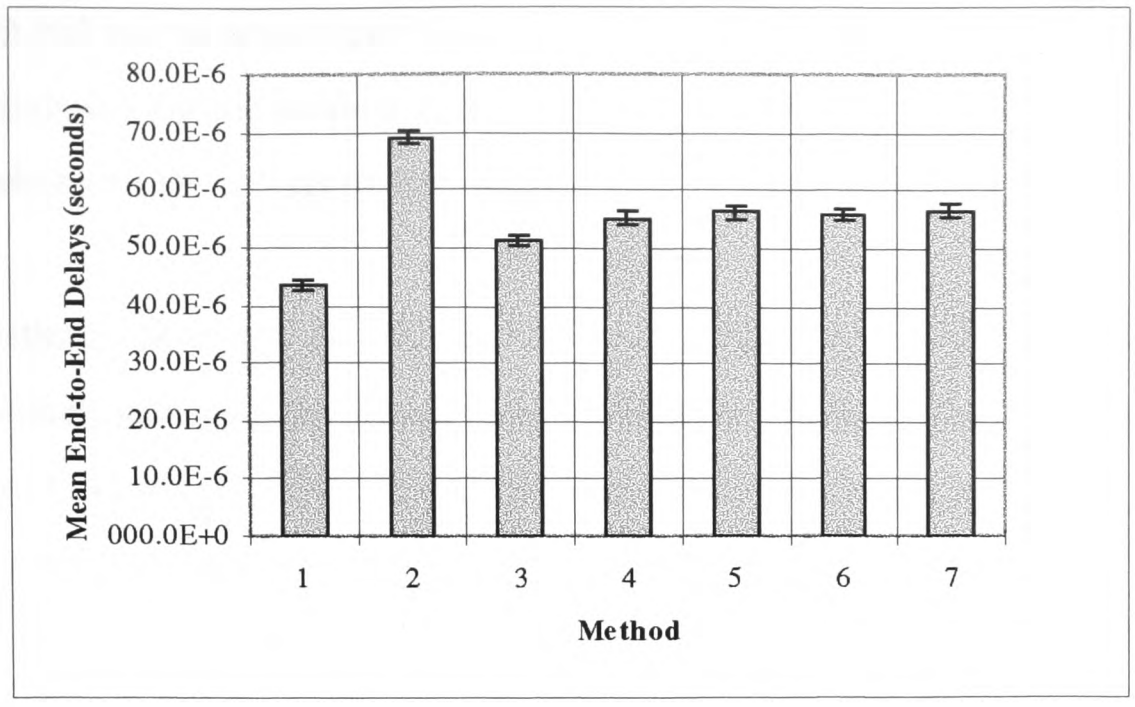



**Figure 4.23 Mean End-to-End Delay for Data Cells (in seconds) - 60 Data Sources**

At this level of utilisation the end-to-end delays for data cells are increased. The S-LB methods still perform well with a 56 μs delay for all these methods. Method 1 has the smallest mean end-to-end delay at 43.6 μs, while Method 2 has the largest, at 69 μs. Method 3 is slightly better than the S-LBs, with a mean end-to-end delay of 51 μs.

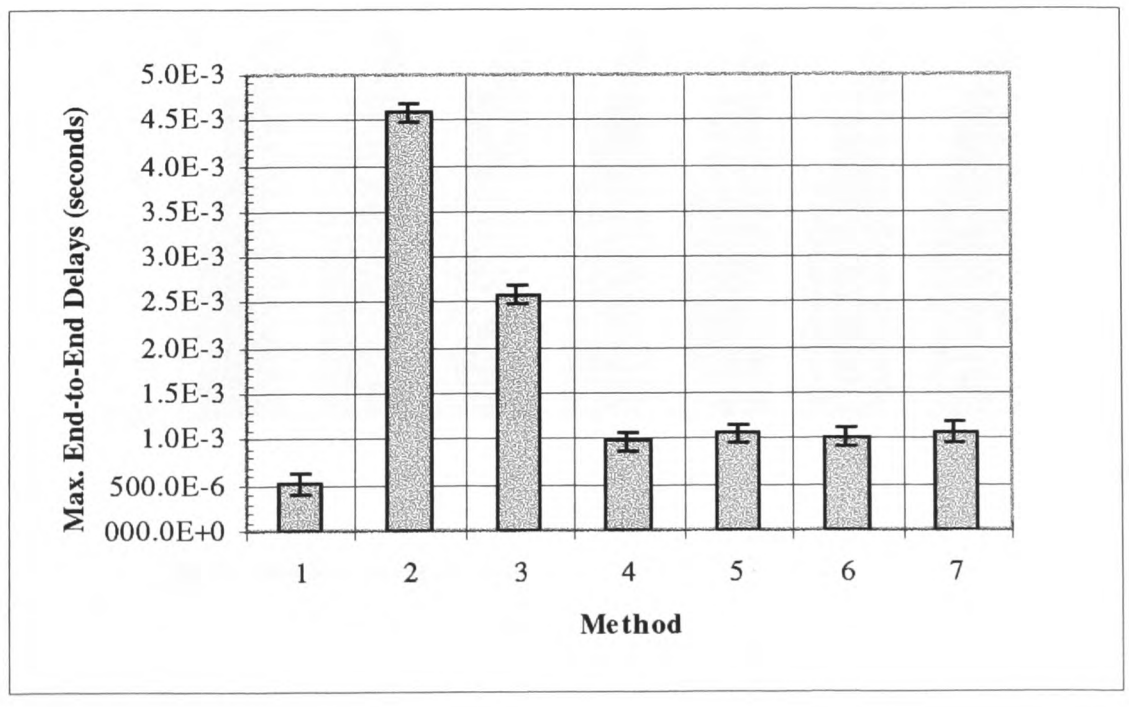



**Figure 4.24 Maximum End-to-End Delay for Data Cells (in seconds) - 60 Data Sources**

The maximum end-to-end delays again show that Method 2 has the largest delay for data cells at 4.6 ms and Method 1 has the smallest delay at 0.5 ms. The S-LB Methods 4 to 7 have increased to 1 ms and Method 3 has a delay of 2.6 ms.

The S-LB methods show that the overall delays experienced by data cells can be kept within bounds by carefully managing the queues at the multiplexer. However, the queue management at the multiplexer blocked 503, 704, 678 and 791 data cells at source for each of the Methods 4 to 7 respectively. By blocking these cells at the source and not allowing them into the queue, the data cells are protected from excessive delays caused by long queues at the multiplexer and also from lost cells due to buffer overflows.

## 4.6 Comparison of 20, 40 and 60 Data Sources

Comparing the 20, 40 and 60 data sources results it can be seen that utilisation has increased across the simulations, see Figure 4.25. The affect that this increase has on the various traffic types can be seen in the results in the previous section.

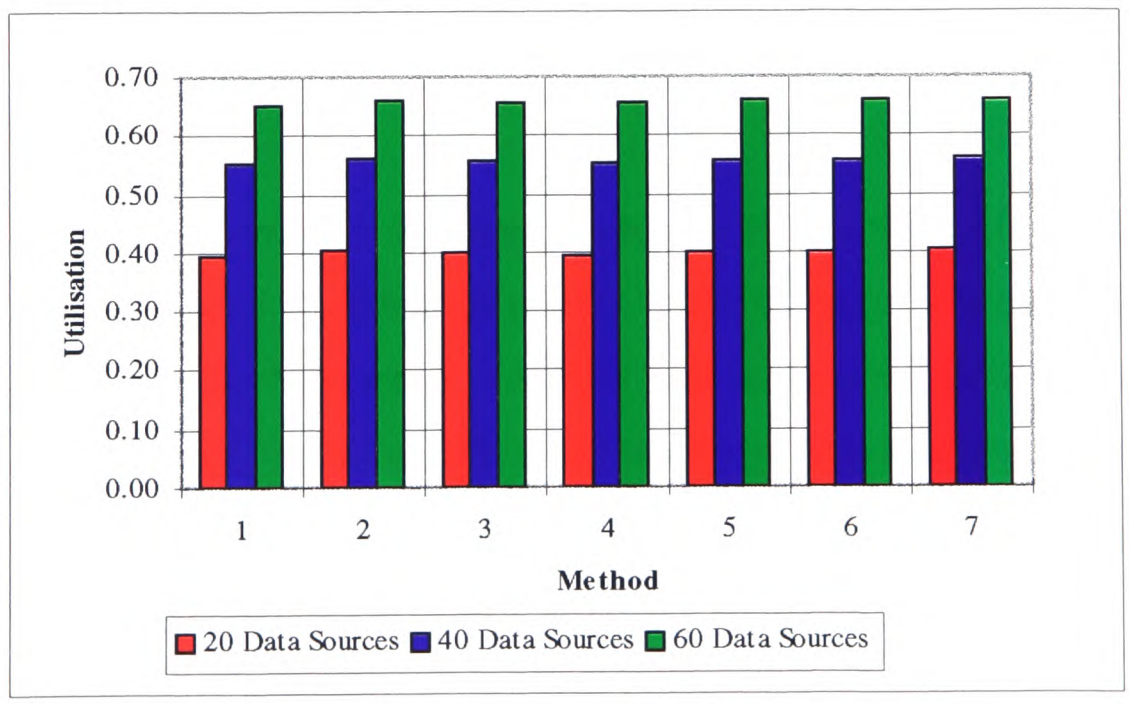



**Figure 4.25 Comparison of Utilisation**

The mean non-RT queue length at the multiplexer shows that Method 2 has the longest queue lengths for all simulations. This is because Method 2 tags all violating cells at the leaky bucket and allows them onto the network, which obviously impacts on the numbers of cells queuing at the multiplexer. The S-LBs are slightly worse than Method 1, but considerably smaller than those found with Methods 2 and 3. The effect that an uncontrolled multiplexer queue has and the performance of the S-LBs with queue management are compared in Figure 4.26.

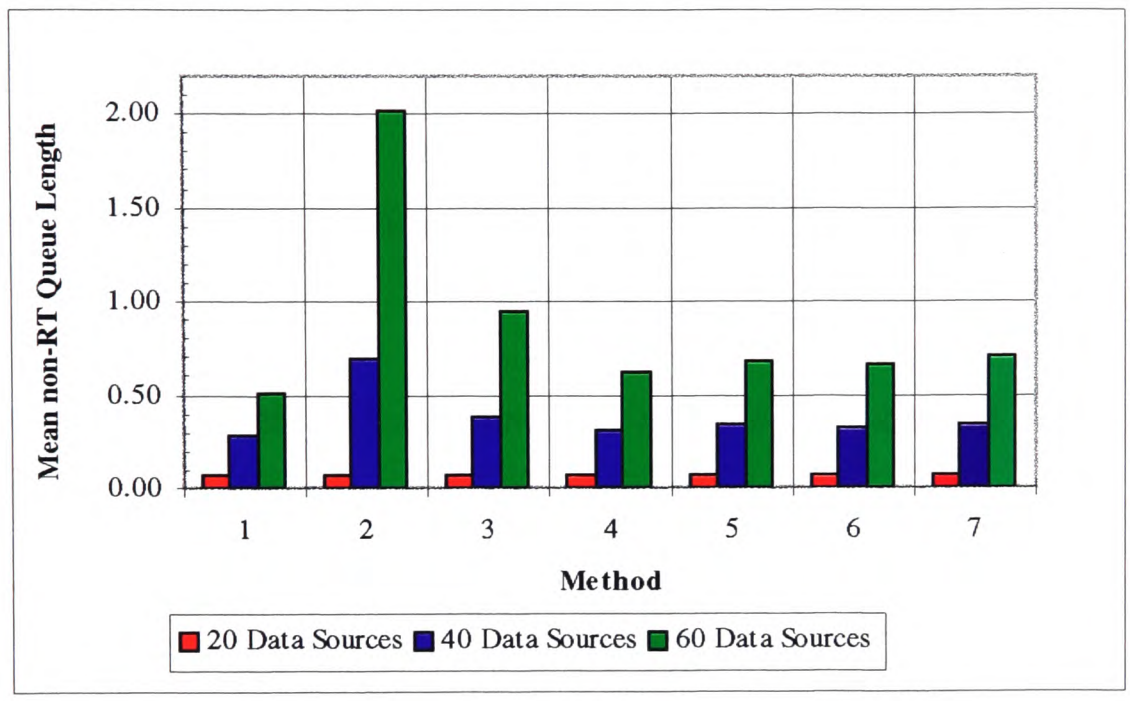



**Figure 4.26 Mean non-RT Queue Length**

The mean RT access delays have increased slightly as the numbers of non-RT cells have increased, as seen in Figure 4.27. When the non-RT queue is empty the server will give priority service to the RT queue. As the numbers of non-RT cells increase and more cells enter the non-RT queue at a multiplexer, the server will spend more time giving alternating service, which causes the slight increase in the delays experienced by the RT cells. It also indicates that the RT cells are protected from any effects of the successive increases in numbers of non-RT cells generated and queued at the multiplexer by the use of cyclic service at the multiplexers, as the increase in the delays are slight.

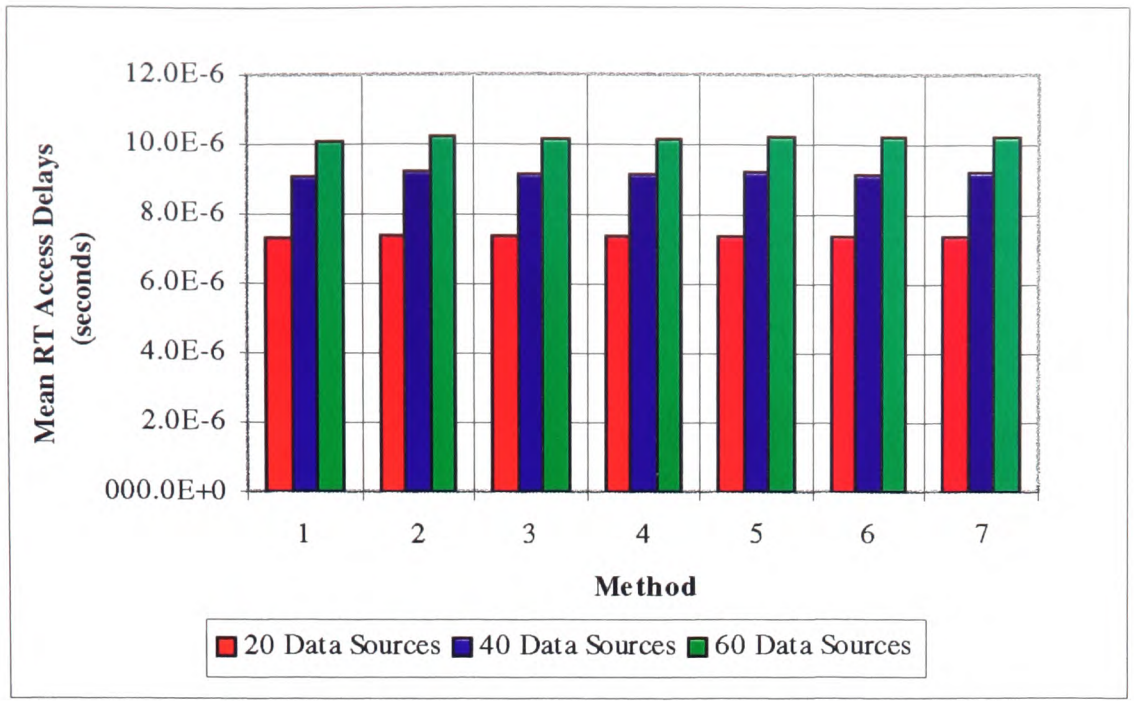



**Figure 4.27 Mean RT Access Delays (in seconds)**

The mean non-RT access delays clearly show the effect of the increased numbers of non-RT cells generated, see Figure 4.28.

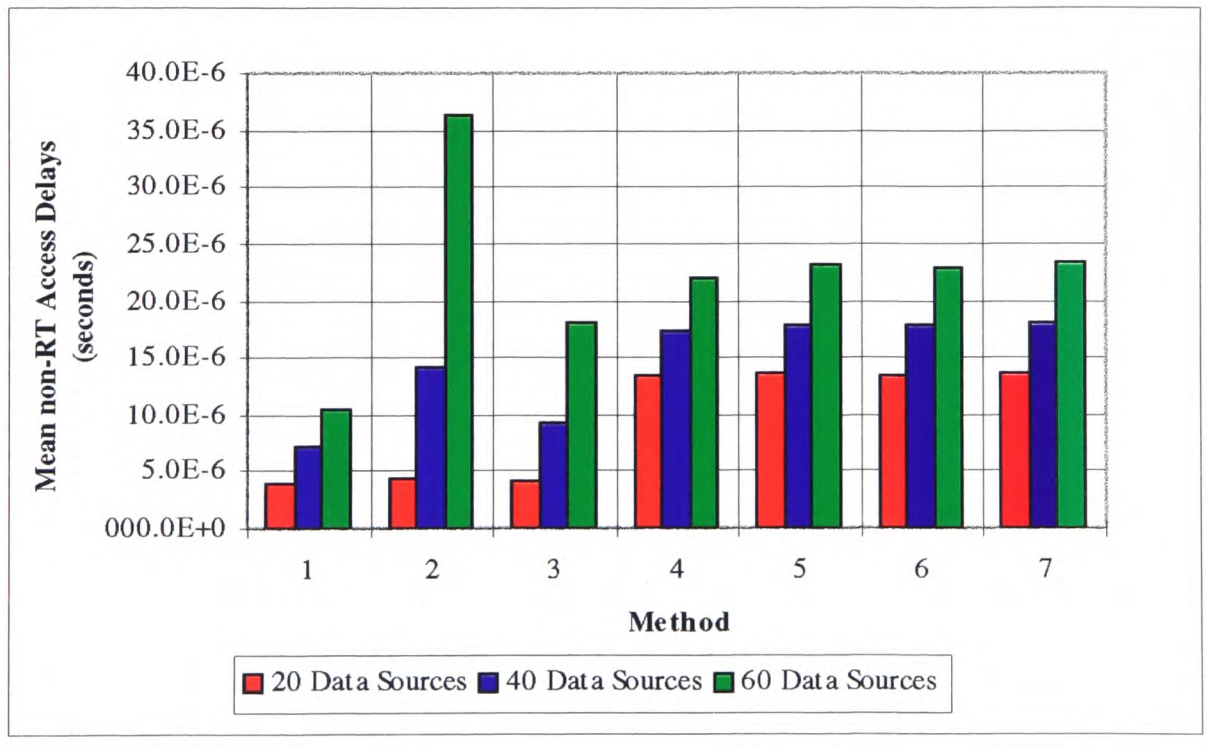



**Figure 4.28 Mean non-RT Access Delays (in seconds)**

The maximum delays encountered by non-RT cells are controlled by the S-LB methods, which restrict the maximum delays encountered across the three sets of simulations.

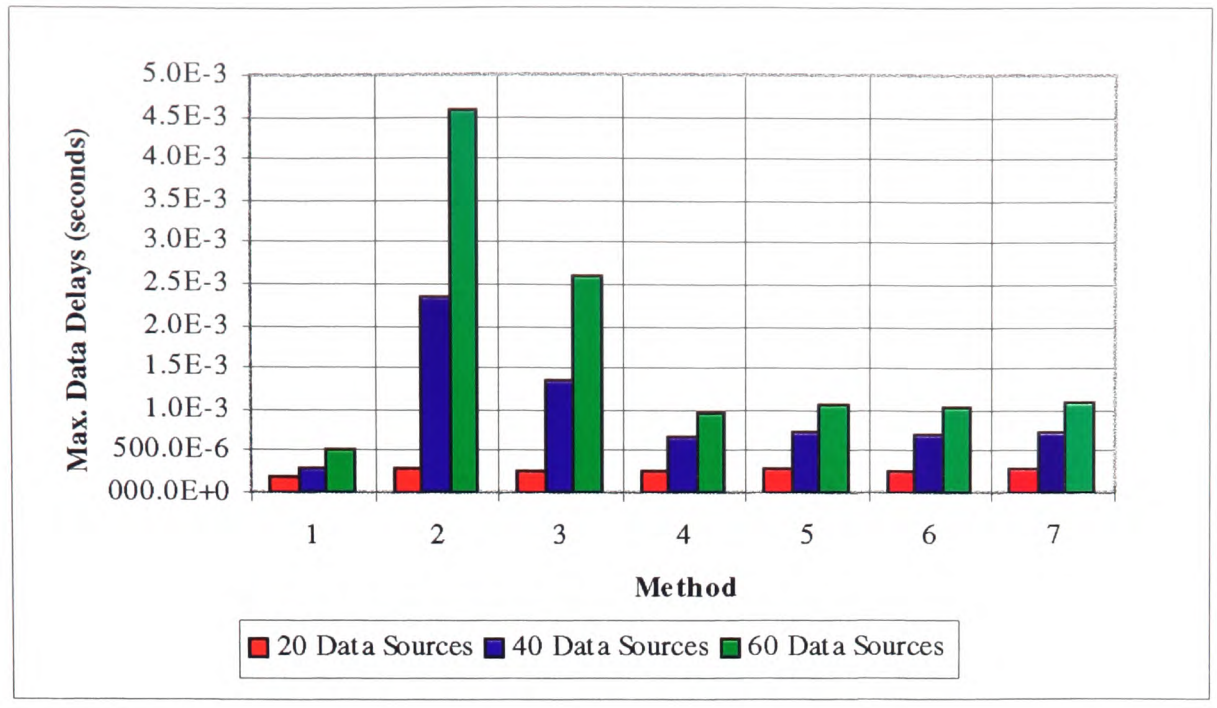



**Figure 4.29 Maximum non-RT Access Delays (in seconds)**

The number of video cells discarded by the policing function remained virtually the same during all the simulations, regardless of the number of non-RT cells present, with the S-LBs outperforming the VLBs. There were a small number of tagged video cells discarded by the multiplexer, at the higher utilisation's caused by 40 and 60 data sources, using Method 2. There is always the risk that tagged cells may see a higher loss probability at multiplexers and switches within the network.

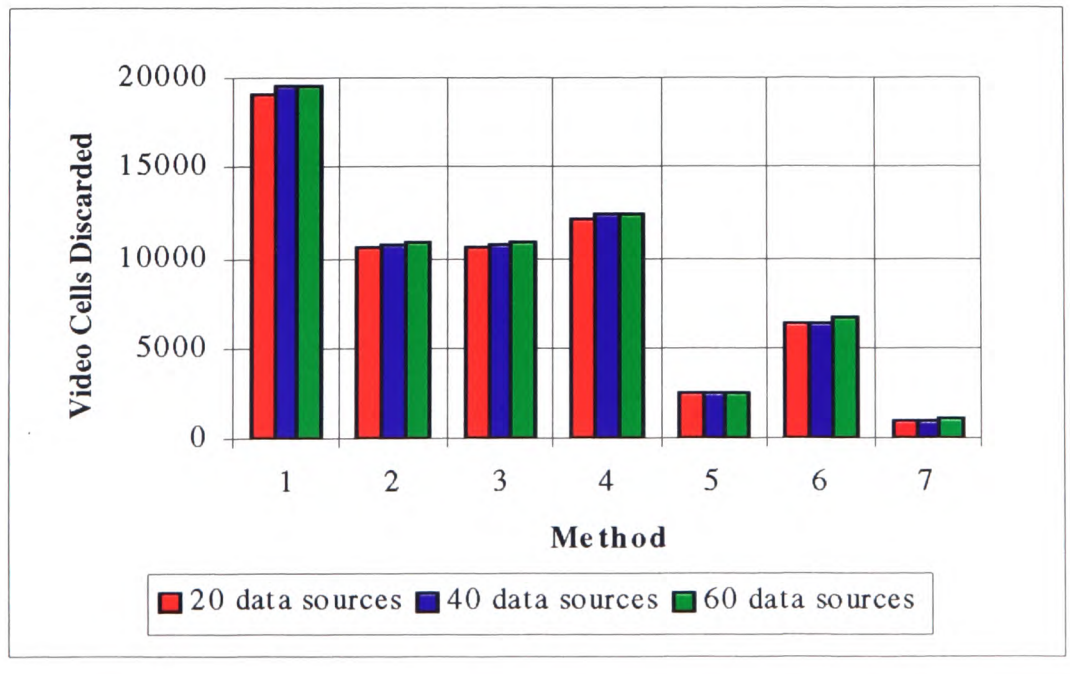



**Figure 4.30 Number of Video Cells Discarded**

The end-to-end delays experienced by the video cells are similar for each of the sets of simulations and are fairly constant across each of the methods within a set. The largest increase occurs between 20 data sources and 40 data sources.

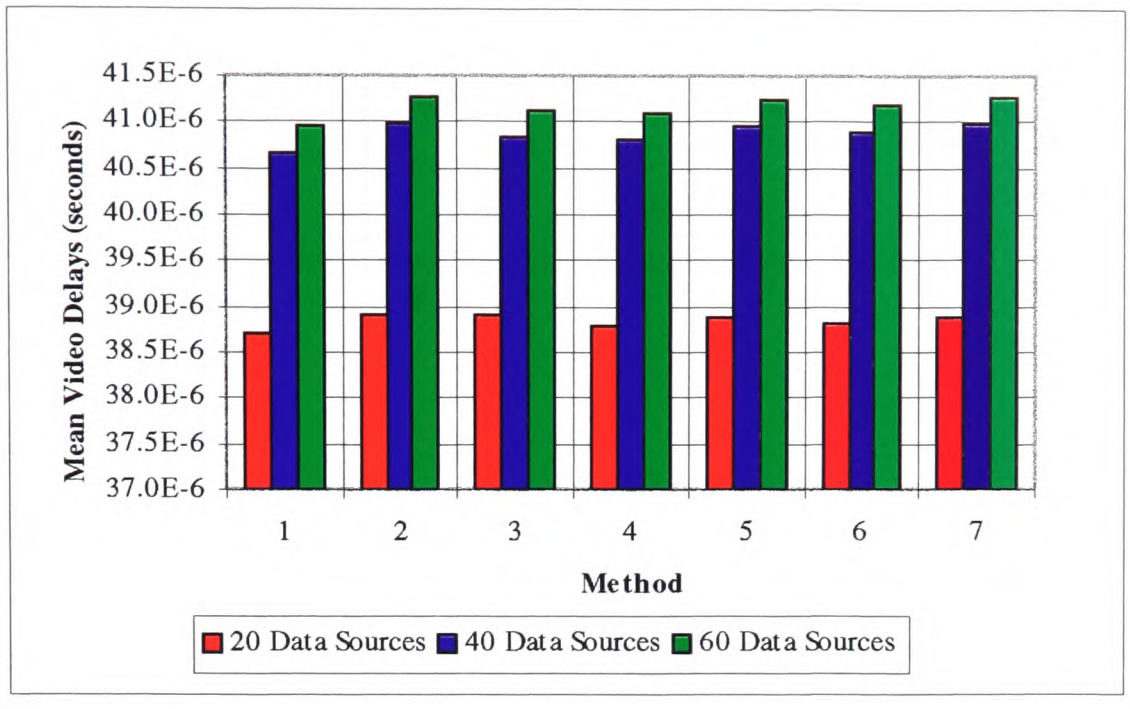



**Figure 4.31 Mean End-to-end Delay for Video Cells**

The mean end-to-end delay for data cells follows the same trend across all the methods, for each set of simulations. For the 20 data sources the differences between the delays found using the VLBs and the S-LBs was marked, with the S-LBs giving the longest end-to-end delays. As the number of data sources increases, that difference became less marked, and for the 60 data sources, the longest mean delay was given using Method 2.

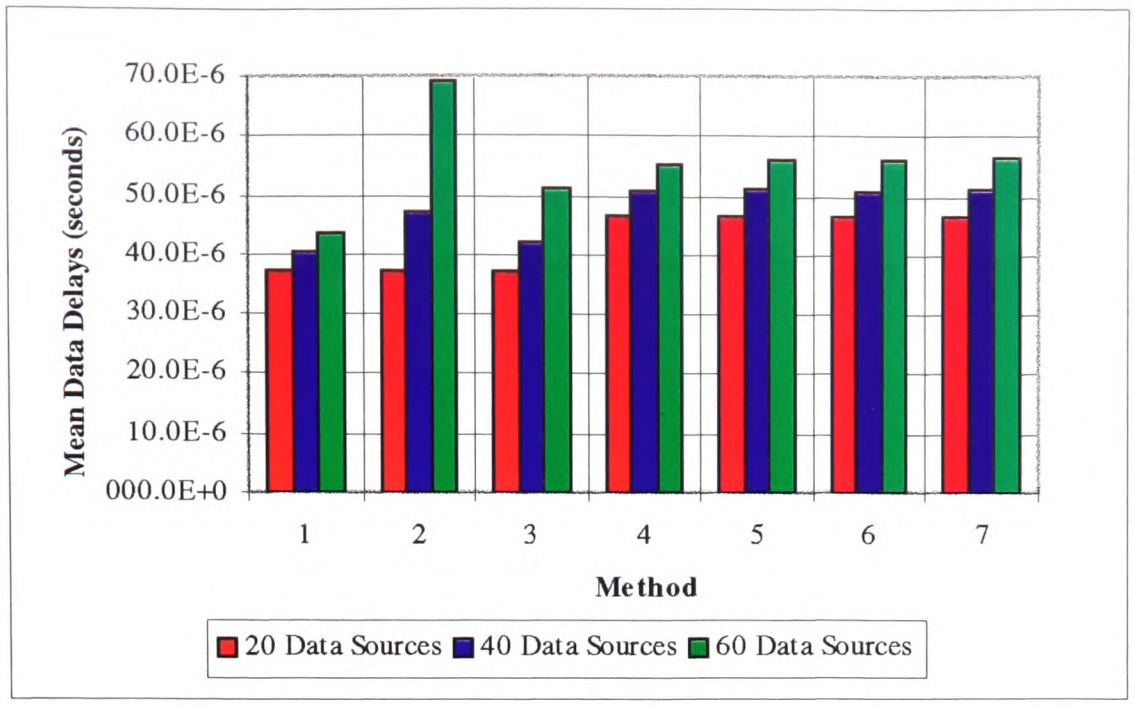



**Figure 4.32 Mean End-to-End Delay Data Cells**

The maximum end-to-end delay for data cells was increased significantly with 60 data sources using Method 2. With this method all cells are allowed onto the network (including violating cells which are tagged), so longer queue lengths at the multiplexer resulted, see Figure 4.33.

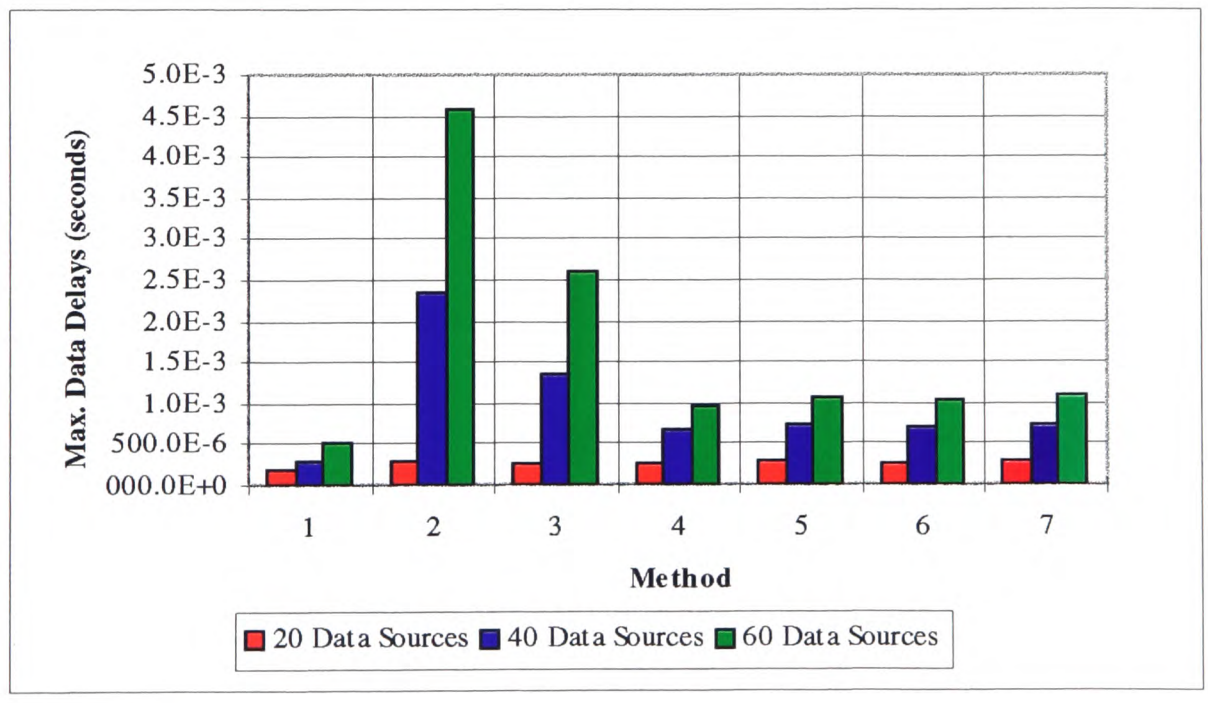



**Figure 4.33 Max. End-to-End Delay Data Cells**

The maximum delays encountered by data cells using the S-LBs are approximately 1 ms for the 60 data sources simulations compared to 4.6 ms using Method 2. The differences in the delays between the three sets of simulations are also less pronounced for the S-LBs, indicating that the management of the multiplexer queue is effective in limiting the delays experienced.

## 4.7 ATM Switches

The utilisation at the various switches of the ATM network remained fairly low, and consequently the queue lengths were correspondingly small. The main output ports of interest are on the last two switches in the network (Switch 5 - output port 2 and switch 2 - output port 2). These two switches have the highest utilisation at these output ports, as they are the focal point for the traffic from the other user sites destined for user site 3. A typical utilisation pattern is shown in Figure 4.34 for a maximum of 20 data sources.

### 4.7.1 Utilisation at Switches

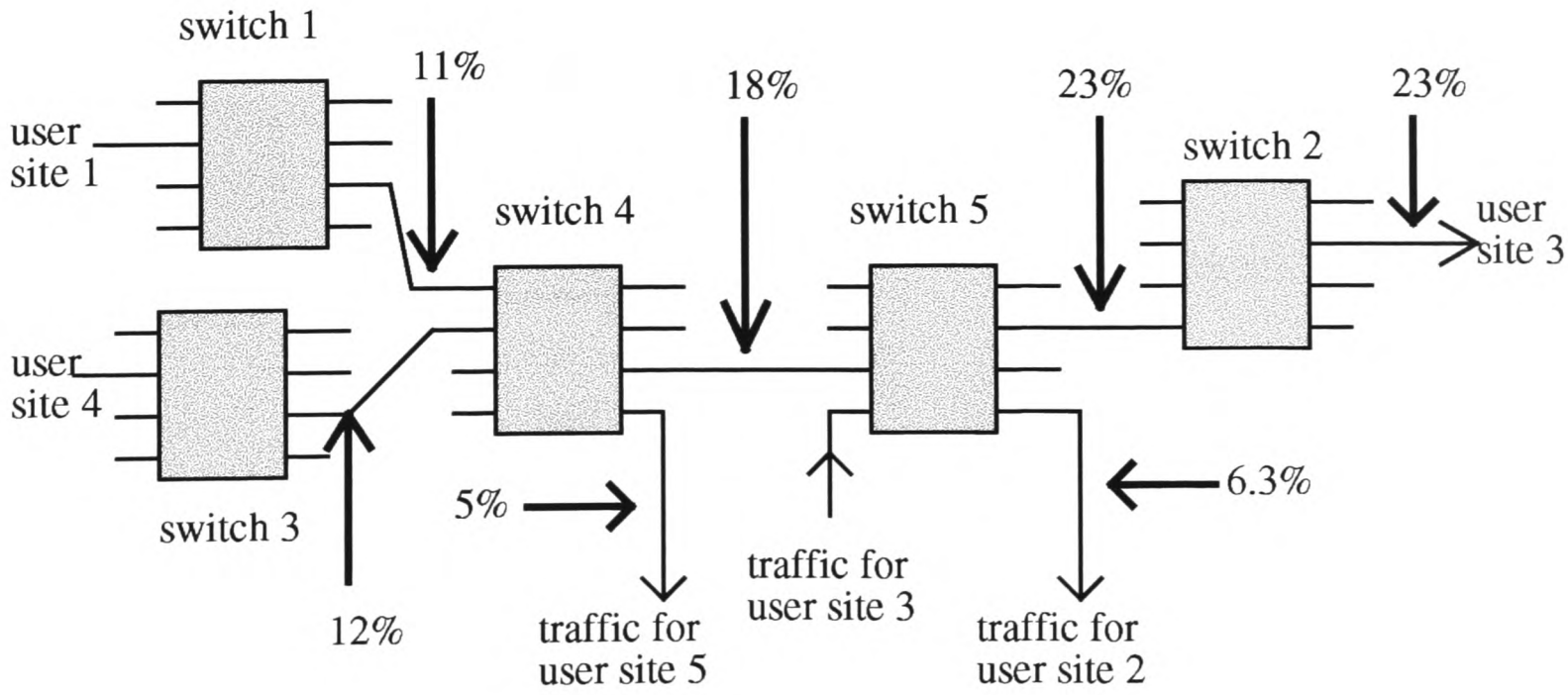



**Figure 4.34 Network Showing Utilisation at each Link using 20 Data Sources**

The utilisation at the output port of each switch is shown in Table A.23.

- Switch 1 - output port 3 - handles all traffic from user site 1.
  Using 20 data sources, three 10 Mb/s VBR video sources and 100 speech sources, the utilisation at this output port fluctuated around 11% for all methods, see Figure 4.34.

  Increasing the number of data sources to 40 causes the utilisation at this port to increase to approximately 14%, for all methods, fluctuating from 13.9% to 14.2% for Method 2 and Method 7, respectively. This is as expected, since Method 2 allows all cells onto the network, so the utilisation would be slightly higher than that found using the other methods.

  Allowing 60 data sources causes the utilisation to increase to approximately 17% and in the case of Method 2 to 17.5%.

- Switch 2 - output port 2 - all traffic destined for user site 3 arrives at this port.
  Traffic from all sites accesses user site 3 through output port 2, at switch 2. As would be expected, this is a heavily utilised port.

  Using 20 data sources the utilisation is 23%. With 40 and 60 data sources, the utilisation rises to 32% and 38% respectively.

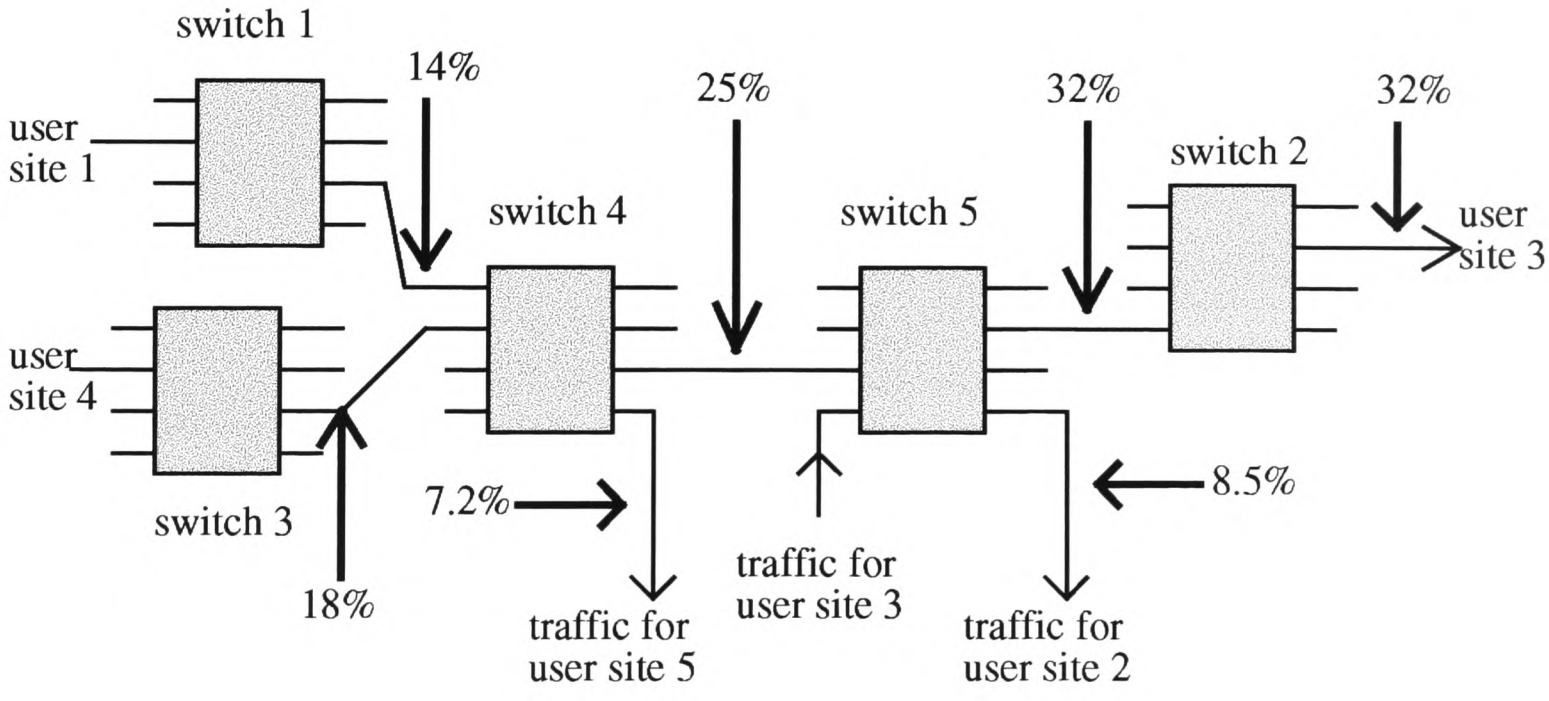



**Figure 4.35 Network Showing Utilisation at each Link for 40 Data Sources**

- Switch 3 - output port 3 - handles all traffic from user site 4.
  A similar trend is seen at switch 3 as that found in switch 1. Utilisation is slightly higher as the number of non-RT cells generated increases.

20 data sources and the same amount of RT traffic as before, give a utilisation of approximately 12%, while using 40 data sources the utilisation is approximately 18% and for 60 data sources the utilisation increases to approximately 19%.

- Switch 4 - output port 3 - handles through traffic from user site 1 and user site 4, destined for either user site 2 or user site 3.
  Utilisation at this output port is 18%, 25% and 29% for 20, 40 and 60 data sources respectively.

- Switch 4 - output port 4 - all traffic on this link is destined for user site 5.
  There is 5% utilisation on this link using 20 data sources and 7.2% and 9%, using 40 and 60 data sources respectively.

- Switch 5 - output port 2 - This output port handles all through traffic destined for user site 3.
  The traffic at this output port originates at user sites 1, 2 and 4, which means that the utilisation is high. For 20 data sources the utilisation is 23% which increases to 32% using 40 data sources and to 38% when 60 data sources are allowed.

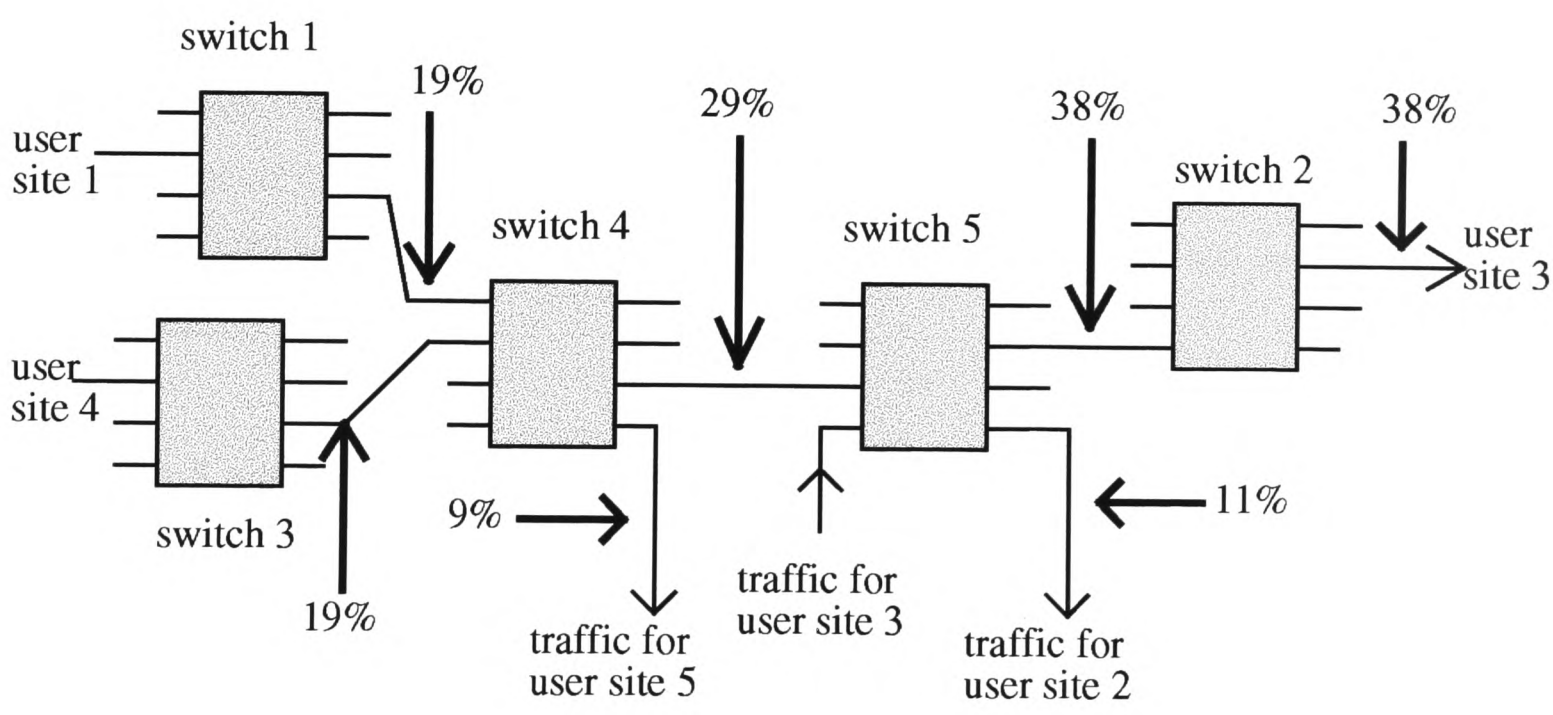



**Figure 4.36 Network Showing Utilisation at each Link for 60 Data Sources**

- Switch 5 - output port 4 - all cells destined for user site 2 must use this port.
  The traffic destined for user site 2 utilises 6.3% of the access link when 20 data sources are allowed. When 40 data sources are used the utilisation increases to 8.5% and to 10.6% when 60 data sources allowed.

### 4.7.2 Cells Served at Each Switch

The average number of cells served during a reporting interval (20 seconds) is shown in Table A.24. No cells are dropped at the switches, as the utilisation is not high enough to generate large queues and cause cells to be discarded.

### 4.7.3 Queue Lengths at Output Ports

The queue lengths at the output ports, during all the simulations, were very small, with the maximum queue length reaching 2 only at switch 5 - output port 2. At switch 5 - output port 2, two independent streams of cells merge. One stream is the combined cell stream from user sites 1 and 4, and the other is the traffic from user site 2. Once the cells have entered the network, they remain orderly. The service time at the access links is 9.422 μs (equivalent to 3.456 slots), which means that the cells are evenly spaced out as they enter the network. The service time at the output ports of the switches is 2.7 μs (1 ATM slot).

At switch 4 the two cell streams from user sites 1 and 4 merge. Even if two cells arrive simultaneously, only one cell will be queued, while the other cell is served. In any case the queue length will never exceed 1, as the next cell will not arrive for another 1.45 slots, by which time both cells will have been served. Two such simultaneously arriving cells will continue across the link to switch 5 back-to-back.

At switch 5 a new cell stream from user site 2 enters the switch destined for user site 3. At switch 5 - port 2, the output stream from switch 4 and the cells entering the network from user site 2 are merged. Here, the maximum queue length achieved is 2. This link is always relatively heavily loaded, compared to the other links.

The traffic arriving at switch 2 now contains cells from user sites 1, 4 and 2, all destined for user site 3. This traffic stream is ordered and will be switched to the output port and served at a steady rate, so that the queue length will never be greater than 1. This same trend is observed for all three sets of simulation experiments.

## 4.8 Discussion of Simulation Results

The first simulation experiment carried out was to determine the optimum position for the leaky bucket with respect to the multiplexer. The results indicate that placing the LB before the multiplexer is a valid choice. Cells which would be discarded by the policing mechanism are not placed in the multiplexer queues. The increased queue lengths found when the multiplexer is accessed first also impact on the access and end-to-end delays experienced. There is an increase in the CDV experienced by individual cells and an increased probability that data cells may be discarded by the policing mechanism. No data cells are lost during any of the simulations which have the policing function positioned before the multiplexer.

In the 20 data sources simulation the average utilisation is 40%. During the 60 data sources, the level of utilisation is increased to 66%, which obviously has an impact on the service seen by the RT queue, in terms of marginally increased delays in crossing the network.

The mean RT queue length is less than 0.185 at each user site in the 20 data sources simulation, less than 0.23 using 40 data sources and less than 0.257 with 60 data sources. The priority service threshold $RT_1$ is set to 2 and the low priority dropping threshold, $RT_2$ is 6, for all the simulations. Since the mean non-RT queue length for 20 data sources is very small, at less than 0.075 cells, then the RT queue will receive priority service for much of the time, simply because the non-RT queue is empty. In the 40 data sources simulation, the non-RT queue has increased to 0.33 for the S-LB methods and 0.28 for Method 1, 0.38 for Method 3 and 0.68 for Method 3. With 60 data sources the non-RT queue length again increased. In both these cases, the server at each user site will spend more time giving alternating service to both queues, and this is reflected in the slight increase RT queue lengths that are found using both 40 and 60 data sources.

Although the utilisation increases with increased load, the delays encountered by the RT traffic remain fairly constant. By managing the access link and giving priority service to the RT queue at the multiplexer, the delays for RT traffic can be kept to a minimum. Allowing some low priority cells to be dropped to prevent the RT queue from growing too long also contributes to keeping individual RT cell delays to a minimum.

Increasing the number of data sources and hence the overall number of data cells generated has increased the utilisation at each of the user sites. This has had very little impact on the delays encountered by RT cells accessing the network. The mean and maximum RT access delays have increased only marginally when comparing the 20 data sources with the 40 and 60 data sources.

The RT end-to-end delays are also relatively unaffected. The end-to-end delays have remained fairly constant for both classes of RT traffic, across all methods. The very slight increase in these delays as seen during the 40 and 60 data source simulations is caused by the overall increased traffic levels. This is a clear indication that the RT cells are protected from the increased queue lengths and increased delays which are encountered by the non-RT cells.

In all the simulations, speech cells experience slightly longer end-to-end delays than video cells. This is because when speech cells are generated, two cells are filled and sent out together. If the first cell is serviced at the multiplexer immediately on arrival, then the second cell must wait at least one service time duration before receiving service. This causes the overall end-to-end delay statistics for speech cells to be higher than those for video cells.

Management of the multiplexer queues allows some controlled dropping of low priority speech cells to enable the RT queue to remain small and hence to keep access delays small. The number of speech cells dropped at the multiplexer is very slightly increased using 40 and 60 data sources, above the levels found with 20 data sources. Only 0.01% of the speech cells are dropped at all the user sites and for all methods, with the exception of user site 2 with 60 data sources, where 0.02% of the speech cells are dropped. This is well within the bounds acceptable for packetised speech.

The number of video cells lost is consistent across all methods, with Method 1 discarding the greatest number and Method 7 discarding the least number of video cells. The number of damaged slices also follows this same trend. The use of the tail-end clipping function within the super leaky bucket function, can be clearly seen to reduce the number of damaged slices, by confining all the damaged cells into a smaller number of slices. Since the presence of one damaged video cell causes the rest of the slice to be discarded, it might be expected that the total number of cells discarded would increase. This has not been the case, and the quality of service for the video has been improved as a lower loss probability has been achieved. All video cells

allowed onto the network by the S-LB methods are high priority cells and are not subject to early discard at times of congestion.

The number of non-RT cells generated has increased from around 300,000 for 20 data sources, to 900,000 with 60 data sources, while the number of RT cells has remained constant at approximately 530,000 cells. With an average 900,000 cells per reporting interval, there will be very little time when the server will not be serving alternately. The mean RT queue length remains small, while the maximum RT queue length is still less than 8.

As the number of data sources increases the delays for data traffic also increases. The longest delay for data cells is 4.5 ms and is found using 60 data sources and the Method 2. Using the S-LBs the longest delay is approximately 1 ms, indicating that the multiplexer queue management policy can keep delays to a minimum.

As the number of data sources is increased, during the simulations, the utilisation at each switch also increases. The highest utilisation occurs during the 60 data sources simulation, at switch 2, output port 2 and at switch 5, output port 2, each with approximately 38% utilisation, see Table A.22. However, the queue lengths at the output ports of these two switches are not significant. This is because, apart from some conflict at switch 5, when cells join from user site 2, the traffic arriving at switch 2 is orderly and evenly spaced, causing no contention within the switch.

Separate queues at the multiplexer for RT and non-RT cells benefit both types of traffic. The RT cells are protected from being excessively delayed behind long queues of non-RT cells. Controlled discarding of low priority speech cells is allowed at the multiplexer to ensure that the RT delays are kept within bounds and to a minimum

Using twice the mean burst size and the peak bit rate to dimension the super leaky bucket gave the best performance for video cells. The next best performance was given by using the mean burst size and the peak bit rate. Using the effective bit rate and twice the mean burst size, also gave a reasonable performance.

It can be seen from the results presented that the RT traffic can be protected, without penalising the non-RT traffic too much.

## 4.9 Conclusion

Previous work [GAN95] has shown that dual queues with cyclic service and queue length thresholds can significantly improve the delays experienced by RT cells and this is the approach which has been used in this work. Priority service for the RT queue when the first RT threshold is exceeded keeps delays at the multiplexer to a minimum. Allowing some controlled discarding of low priority cells (mainly speech) also helps to prevent long queues at the multiplexer for RT cells.

Positioning the multiplexer before the policing function causes increased CDV with subsequent increased cell loss. Also any cells which are destined to be discarded by the policing function are queued at the multiplexer along with conforming cells. This causes an additional unnecessary delay to those cells.

The QoS for RT cells is maintained. Using Methods 5, 6 and 7 the number of video cells discarded is reduced, with Methods 5 and 7 having the least number lost. The use of tail-end-clipping within the S-LB ensures that any discarded video cells are compacted into the smallest number of video slices. This also significantly reduces the number of damaged video slices when compared to the standard VLB. The number of discarded speech cells is not significantly increased. The delays for RT cells remain constant as the utilisation at the access link increases.

The QoS is also maintained for non-RT cells. Data cells are protected from being discarded and the end-to-end delays are kept within reasonable bounds.

# Chapter 5 - Conclusion

## 5.1 Summary

A composite strategy has been proposed to police the access link to an ATM network and to manage the buffers at a multiplexer. The policing function is based on the leaky bucket mechanism. A simulation model has been implemented to compare the performance of the proposed strategy with a standard VLB.

The objective has been to maintain the QoS for each of the three classes of traffic under investigation, represented by data, speech and video traffic. A review of the area has been carried out and is discussed in Chapter 2. A series of simulation experiments have been performed using the model outlined in Chapter 3, to answer the questions posed in Section 1.1 and the results are presented in Chapter 4.

Three widely diverse types of traffic are included in this work, each have different QoS requirements. A single policing mechanism is not able to maintain the QoS for all three types and so a policing mechanism is required which treats each in accordance with its QoS requirements. Data traffic needs loss free delivery and delays are less important, so a buffered leaky bucket (B-LB) is proposed which delays cells and protects them from being discarded. Speech traffic has strict delay requirements but can tolerate the loss of some low priority cells. For speech cells a standard virtual leaky bucket is used, which tags violating cells as low priority. Video has strict delay requirements and the loss of a single cell within the video stream may invalidate the remaining cells which will be discarded on arrival as unusable. Video traffic is policed by a modified virtual leaky bucket which clips the tail end of any damaged video slice and thereby reduces the danger of desynchronising the video decoder and introducing errors into the displayed picture. The start of the next video slice causes a tail-end-clipping function to reset and stop discarding cells. A video slice is the smallest resynchronisation point within the coded video stream.

At the multiplexer RT and non-RT traffic is segregated into separate queues. Cells from each queue are serviced alternately, unless the RT queue grows beyond a threshold ($RT_1$). The server will then switch over to priority service for the RT queue, until the queue length falls below the

threshold. If the RT queue exceeds a second threshold ($RT_2$), then all low priority cells are dropped as they reach the head of the queue.

The non-RT queue also has two thresholds associated with it, which are used to throttle-back the flow of non-RT cells to prevent buffer overflow. Data cells which can not be serviced wait in a buffer until service is available. If the non-RT queue at the multiplexer grows too long, then the service rate is slowed to reduce the flow of cells. If the queue continues to grow then the data sources will be blocked at source from sending more cells.

The simulation model was modified to produce a reference model based on the virtual leaky bucket proposed by [NIEST90]. This was modified to allow the VLB to be dimensioned using the mean burst length for video. There are two versions of the VLB, one which tags violating cells and the other discards them.

Four versions of the super leaky bucket have been included in the simulation. Two of the super leaky buckets were dimensioned using a single video burst length and the second pair have the leaky bucket dimensioned using twice the peak bit rate. For each pair, one was dimensioned using the effective bit rate (see Section 3.1.3.2) and the other used the peak bit rate in each category. Using twice the mean burst size for video allows longer bursts of video cells to pass through the leaky bucket without being discarded..

Four sets of simulation experiments have been performed and in each case the number of RT sources remained the same. The first two sets of simulations used identical numbers of non-RT traffic and the third and fourth sets had the proportion of non-RT traffic increased successively. The performance of the proposed composite UPC strategy was tested under these different loads and the QoS for the various traffic sources was observed.

It has been shown that the delays for RT cells are unaffected by the position of the policing function. However, non-RT cells do experience a slightly greater risk of being discarded at the leaky bucket due to increased CDV caused by delays in the multiplexer queue. From the results it can be seen that the proposal to position the leaky bucket before the multiplexer is a valid one.

Initially, it was found that the non-RT cells using the super leaky bucket, do experience a slightly increased delay at the multiplexer, compared to the standard VLBs. This occurs at low utilisation and when RT traffic is the larger component of the traffic stream but the delay is not excessive and within reasonable bounds. However, as the number of data sources increased these differences at the multiplexer became less. At high utilisation it was found that the super leaky buckets actually outperformed the VLBs, with the exception of the reference model, which gave the smallest access delays of all. With data cells the main focus is to prevent cell losses and slightly increased delays are less significant.

From the results presented in Chapter 4, it can be clearly seen that the QoS requirements for each category of traffic source have been maintained. The S-LB discards fewer video cells and those that are dropped are concentrated into a smaller number of video slices than the standard VLB.

By preventing RT cells from being queued behind large numbers of non-RT cells the delays experienced by individual cells are contained. The performance for the RT cells remained constant even when there was a large increase in the utilisation at the UNI. Since the multiplexer queue can be a bottleneck, this can have a significant influence on the end-to-end delays experienced and this is an important consideration when transporting RT traffic.

In each case it has been shown that the super leaky bucket outperforms the standard virtual leaky bucket for each of the three classes of traffic.

## 5.2 Limitations/Difficulties

For any performance analysis of an ATM network it is necessary to simulate at the cell level to obtain any useful statistics. This means that the granularity of the simulations was very fine (2.7 μs). The simulations were run for 200 seconds and each one took between 3 and 7 days to complete. Since there is very limited access to Sparcs with sufficient power to complete these runs, this often meant using the slower and less powerful machines. Problems with the network also meant that runs would be prematurely terminated, requiring the work to be repeated. There were also problems with colleagues terminating jobs which had been put onto powerful machines

to run over the weekend, which would otherwise have sat idle. Ideally, the simulations should have been performed with a larger number of traffic sources and over a longer time period. However, this would have made the runs even more susceptible to the whims of the network.

Initially, when this project was begun it was necessary to read widely on a large range of topics to obtain the necessary back-ground knowledge for the work. There are a great many facets to this work, which included the various types of traffic and the individual characteristics and requirements of each when being transported across an ATM network. The evolving ATM network standards were also interesting to observe. My research skills have improved considerably over the course of this work. The MODSIM simulation language had to be mastered. Being an object oriented language helped in the construction of the simulation model, as this approach maps well onto the model designed. User sites, traffic sources, multiplexers and ATM switches are all easily modelled and implemented as objects.

This work is part of an ongoing research project. As more ATM networks become operational, the traffic mixes found on these networks would be of interest to compare with the model presented here. Further work will also include analysis of the loading of the switches within the network. Queue length restrictions could then be added to switch buffers and the performance for the various traffic types could then be observed.

# Appendix I – Tables of Results

# Appendix I – Tables of Results

Positioning of Multiplexer - using Method 3 and 60 Data Sources

| All Cells | MuxToLB | VLBtoMux Method 3 | VLBtoMux Method 2 |
|---|---|---|---|
| RT Gen. | 521260 | 521260 | 521260 |
| RT Rec. | 510345 | 510347 | 521239 |
| Non-RT Gen. | 937482 | 937482 | 937482 |
| Non-RT Rec. | 936625 | 937482 | 937482 |
| Discarded Data cells | 765 | 0 | 0 |
| | | | |
| | MuxToLB | Method 3 | Method 2 |
| Utilisation | 0.686 | 0.682 | 0.687 |

**Table A.1 Positioning of the Muliplexer - Cells Generated, Received and Utilisation**

| RT Q Len | MuxToLB | VLBtoMux Method 3 | VLBtoMux Method 2 |
|---|---|---|---|
| mean | 0.265 | 0.256 | 0.265 |
| max. | 7.99 | 7.9 | 7.97 |
| No still in queue | 0.389 | 0.361 | 0.389 |
| | | | |
| Non-RT Q Len | MuxToLB | Method 3 | Method 2 |
| mean | 2.45 | 1.126 | 2.455 |
| max. | 288.15 | 158.4 | 288.15 |
| No still in queue | 0. 847 | 0.639 | 0.847 |

**Table A.2 RT and non-RT Queue Lengths at Multiplexer**

| RT Access | MuxToLB | VLBtoMux Method 3 | VLBtoMux Method 2 |
|---|---|---|---|
| mean | 10.5 | 10.4 | 10.5 |
| max. | 69.2 | 68.6 | 69.1 |
| | | | |
| Non-RT Access | MuxToLB | Method 3 | Method 2 |
| mean | 43.3 | 20.8 | 43.3 |
| max. | 5,324 | 2,955 | 5,319 |
| | | | |

**Table A.3 Access Delays at Multiplexer (in $\mu$s)**

| Video Cells | MuxToLB | VLBtoMux Method 3 | VLBtoMux Method 2 |
|---|---|---|---|
| Gen. | 378528 | 378528 | 378528 |
| Rec. | 367634 | 367634 | 378528 |
| Discarded | 10895 | 10895 | 10895 |
| Percentage | 2.88% | 2.88% | 2.88% |
| | | | |

**Table A.4 Video Cells Generated and Received**

| Video Slices | MuxToLB | VLBtoMux Method 3 | VLBtoMux Method 2 |
|---|---|---|---|
| $N^{O.}$ of slices | 7558 | 7558 | 7558 |
| Damaged slices | 285 | 285 | 285 |
| % damaged | 3.77% | 3.77% | 3.77% |
| | | | |

**Table A.5 Video Slices Damaged**

| Speech Cells | MuxToLB | VLBtoMux Method 3 | VLBtoMux Method 2 |
|---|---|---|---|
| Gen. | 142732 | 142732 | 142732 |
| Received | 142711 | 142713 | 142711 |
| Number lost | 19.3 | 18.6 | 20.7 |
| % loss | 0.01% | 0.01% | 0.01% |
| | | | |

**Table A.6 Speech Cells Lost**

| Video Delays | MuxToLB | VLBtoMux Method 3 | VLBtoMux Method 2 |
|---|---|---|---|
| mean | 41.2 | 41.1 | 41.2 |
| max. | 96.0 | 96.3 | 96.5 |
| | | | |
| Speech Delays | MuxToLB | Method 3 | Method 2 |
| mean | 48.7 | 48.7 | 48.8 |
| max. | 100 | 100 | 100 |
| | | | |
| Data Delays | MuxToLB | Method 3 | Method 2 |
| mean | 74.5 | 54.0 | 76.1 |
| max. | 5,324 | 2,955 | 5,342 |
| | | | |

**Table A.7 End-to-End Delays Video Cells (in $\mu$s)**

VLB Vs Super Leaky Bucket

| All Cells | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| RT Gen. | 527233 | 527233 | 527233 | 527233 | 527233 | 527233 | 527233 |
| RT Rec. | 508152 | 527217 | 516733 | 515110 | 524803 | 520789 | 526258 |
| Non-RT Gen. | 326828 | 326828 | 326828 | 326828 | 326828 | 326828 | 326828 |
| Non-RT Rec. | 326828 | 326828 | 326828 | 326828 | 326828 | 326828 | 326828 |
| Blocked Data Cells | | | | 1.156 | 1.633 | 1.533 | 2.022 |
| 40 Data Sources | | | | | | | |
| RT Gen. | 528613 | 528613 | 528613 | 528613 | 528613 | 528613 | 528613 |
| RT Rec. | 509113 | 528594 | 517862 | 516203 | 526095 | 522188 | 527625 |
| Non-RT Gen. | 658537 | 658537 | 658537 | 658537 | 658537 | 658537 | 658537 |
| Non-RT Rec. | 658537 | 658537 | 658537 | 658407 | 658337 | 658352 | 658308 |
| Blocked Data Cells | | | | 130.12 | 199.72 | 184.97 | 228.93 |
| 60 Data Sources | | | | | | | |
| RT Gen. | 522622 | 522622 | 522622 | 522622 | 522622 | 522622 | 522622 |
| RT Rec. | 503085 | 522602 | 511787 | 510148 | 520122 | 515869 | 521574 |
| Non-RT Gen. | 882095 | 882095 | 882095 | 882095 | 882095 | 882095 | 882095 |
| Non-RT Rec. | 882095 | 882095 | 882095 | 881591 | 881390 | 881416 | 881303 |
| Blocked Data Cells | | | | 503.37 | 704.06 | 678.12 | 790.83 |

**Table A.8 Total Cells Generated and Received**

| Utilisation | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | 0.393 | 0.402 | 0.397 | 0.397 | 0.401 | 0.399 | 0.402 |
| 40 Data Sources | 0.549 | 0.558 | 0.554 | 0.553 | 0.558 | 0.556 | 0.559 |
| 60 Data Sources | 0.652 | 0.662 | 0.656 | 0.656 | 0.660 | 0.658 | 0.661 |

**Table A.9 Utilisation**

| Percentage | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| Percentage RT | 61.9 | 61.9 | 61.9 | 61.9 | 61.9 | 61.9 | 61.9 |
| non-RT | 38.1 | 38.1 | 38.1 | 38.1 | 38.1 | 38.1 | 38.1 |
| 40 Data Sources | | | | | | | |
| Percentage RT | 45.2 | 45.2 | 45.2 | 45.2 | 45.2 | 45.2 | 45.2 |
| non-RT | 54.8 | 54.8 | 54.8 | 54.8 | 54.8 | 54.8 | 54.8 |
| 60 Data Sources | | | | | | | |
| Percentage RT | 37.2 | 37.2 | 37.2 | 37.2 | 37.2 | 37.2 | 37.2 |
| non-RT | 62.8 | 62.8 | 62.8 | 62.8 | 62.8 | 62.8 | 62.8 |

**Table A.10 Percentage RT and Non-RT Cells Generated**

| RT Q length | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 data sources | | | | | | | |
| mean | 0.176 | 0.185 | 0.180 | 0.179 | 0.184 | 0.182 | 0.185 |
| max. | 7.62 | 7.84 | 7.73 | 7.69 | 7.80 | 7.78 | 7.79 |
| No still in RT Q | 0.2 | 0.211 | 0.1889 | 0.2 | 0.2 | 0.233 | 0.2 |
| 40 data sources | | | | | | | |
| mean | 0.221 | 0.234 | 0.226 | 0.226 | 0.232 | 0.229 | 0.233 |
| max. | 7.78 | 7.83 | 7.79 | 7.82 | 7.90 | 7.82 | 7.89 |
| No still in RT Q | 0.1889 | 0.2 | 0.2 | 0.189 | 0.189 | 0.2 | 0.2222 |
| 60 data sources | | | | | | | |
| mean | 0.243 | 0.258 | 0.250 | 0.249 | 0.257 | 0.254 | 0.258 |
| max. | 7.8 | 7.92 | 7.83 | 7.77 | 7.88 | 7.91 | 7.91 |
| No still in RT Q | 0.3667 | 0.3667 | 0.344 | 0.3667 | 0.3778 | 0.3667 | 0.3889 |

**Table A.11 RT Q Len**

| non-RT Q Length | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| mean | 0.071 | 0.0752 | 0.0734 | 0.0725 | 0.074 | 0.0731 | 0.0741 |
| Max | 4.82 | 8.178 | 6.86 | 5.54 | 6.23 | 5.856 | 6.256 |
| No still in non-RT Q | 0.044 | 0.044 | 0.044 | 0.056 | 0.056 | 0.056 | 0.056 |
| | | | | | | | |
| 40 Data Sources | | | | | | | |
| mean | 0.276 | 0.684 | 0.386 | 0.313 | 0.334 | 0.3285 | 0.3408 |
| Max | 11.83 | 119.7 | 63.2 | 17.07 | 17.93 | 17.56 | 17.98 |
| No still in non-RT Q | 0.36 | 0.33 | 0.33 | 0.31 | 0.3 | 0.3 | 0.311 |
| | | | | | | | |
| 60 Data Sources | | | | | | | |
| mean | 0.502 | 2.018 | 0.949 | 0.624 | 0.677 | 0.667 | 0.6997 |
| Max | 26.94 | 241.29 | 133.77 | 25.51 | 26.03 | 25.82 | 26.144 |
| No still in non-RT Q | 0.522 | 0.767 | 0.578 | 0.578 | 0.589 | 0.578 | 0.578 |

**Table A. 12 Non-RT Q Len**

| RT Access Delay | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| Mean | 7.33 | 7.42 | 7.37 | 7.366 | 7.406 | 7.388 | 7.41 |
| Max | 67.6 | 67.4 | 67.5 | 67.46 | 67.68 | 67.48 | 67.5 |
| 40 Data Sources | | | | | | | |
| Mean | 9.07 | 9.22 | 9.14 | 9.14 | 9.21 | 9.18 | 9.22 |
| Max | 69.5 | 69.2 | 69.1 | 69.61 | 69.45 | 69.49 | 69.6 |
| 60 Data Sources | | | | | | | |
| Mean | 10 | 10.3 | 10.2 | 10.15 | 10.24 | 10.21 | 10.3 |
| Max | 69.3 | 69.0 | 68.7 | 69.41 | 69.75 | 69.40 | 69.4 |

**Table A.13 RT Access Delays (in $\mu$s)**

| non-RT Access Delays | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| Mean | 3.98 | 4.38 | 4.08 | 13.46 | 13.54 | 13.5 | 13.5 |
| Max | 169 | 271 | 251 | 236.4 | 285.26 | 263.11 | 288 |
| 40 Data Sources | | | | | | | |
| Mean | 7.21 | 14.1 | 9.27 | 17.41 | 17.85 | 17.73 | 18.0 |
| Max | 269 | 2,304 | 1,328 | 653.66 | 708.34 | 686.65 | 722 |
| 60 Data Sources | | | | | | | |
| Mean | 10.4 | 36.2 | 18.1 | 22.01 | 23.02 | 22.81 | 23.4 |
| Max | 501 | 4,559 | 2,562 | 957.27 | 1,040.05 | 1,003.9 | 1,059 |

**Table A.14 Non-RT Access Delays (in $\mu$s)**

| RT Delays | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| Mean | 40.4 | 40.5 | 40.4 | 40.41 | 40.46 | 40.44 | 40.5 |
| Max | 99.2 | 98.8 | 98.9 | 98.655 | 98.708 | 98.78 | 98.8 |
| 40 Data Sources | | | | | | | |
| Mean | 42.4 | 42.6 | 42.5 | 42.51 | 42.599 | 42.565 | 42.6 |
| Max | 101 | 101 | 100 | 100.73 | 100.74 | 100.68 | 101 |
| 60 Data Sources | | | | | | | |
| Mean | 43.0 | 43.2 | 43.1 | 43.08 | 43.16 | 43.13 | 43.2 |
| Max | 101 | 101 | 101 | 101.03 | 101.38 | 101 | 101 |

**Table A.15 RT End-to-End Delays (in $\mu$s)**

| non-RT Delays | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| Mean | 37.0 | 37.4 | 37.1 | 46.51 | 46.6 | 46.55 | 46.6 |
| Max | 179 | 279 | 257 | 245.71 | 293.73 | 267.21 | 295 |
| 40 Data Sources | | | | | | | |
| Mean | 40.4 | 47.1 | 42.3 | 50.55 | 51.0 | 50.87 | 51.1 |
| Max | 290 | 2,327 | 1,349 | 672.6 | 726.66 | 704.96 | 741 |
| 60 Data Sources | | | | | | | |
| Mean | 43.6 | 69.1 | 51.2 | 55.24 | 56.24 | 56.038 | 56.6 |
| Max | 525 | 4,582 | 2,585 | 969.63 | 1,059.5 | 1,022.62 | 1,076 |

**Table A.16 Non-RT End-to-End Delays (in $\mu$s)**

| Video Cells | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| No. Generated. | 377576 | 377576 | 377576 | 377576 | 377576 | 377576 | 377576 |
| No. Received | 358509 | 377576 | 367091 | 365467 | 375162 | 371147 | 376616 |
| Discarded | 19068 | 10485 | 10485 | 12109 | 2415 | 6429 | 960 |
| Percentage | 5.05% | 2.78% | 2.78% | 3.21% | 0.64% | 1.7% | 0.25% |
| 40 Data Sources | | | | | | | |
| No. Generated. | 378851 | 378851 | 378851 | 378851 | 378851 | 378851 | 378851 |
| No. Received | 359366 | 378851 | 368117 | 366458 | 376351 | 372444 | 377882 |
| Discarded | 19485 | 10735 | 10735 | 12394 | 2500 | 6407 | 969 |
| Percentage | 5.14% | 2.83% | 2.83% | 3.27% | 0.66% | 1.69% | 0.26% |
| 60 Data Sources | | | | | | | |
| No. Generated. | 378501 | 378501 | 378501 | 378501 | 378501 | 378501 | 378501 |
| No. Received | 358979 | 378501 | 367683 | 366044 | 376020 | 371766 | 377472 |
| Discarded | 19522 | 10818 | 10818 | 12457 | 2482 | 6735 | 1030 |
| Percentage | 5.16% | 2.86% | 2.86% | 3.29% | 0.66% | 1.78% | 0.27% |

**Table A.17 Video Cells**

| Video Slices | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| $N^{O.}$ of slices | 7539 | 7539 | 7539 | 7539 | 7539 | 7539 | 7539 |
| Damaged slices | 914.64 | 276.5 | 276.5 | 264.12 | 62.53 | 137.06 | 22.86 |
| % damaged | 12.13% | 3.67% | 3.67% | 3.50% | 0.83% | 1.82% | 0.30% |
| 40 Data Sources | | | | | | | |
| $N^{O.}$ of slices | 7579 | 7579 | 7579 | 7579 | 7579 | 7579 | 7579 |
| Damaged slices | 935.13 | 282.86 | 282.86 | 268.77 | 64.76 | 136.1 | 22.96 |
| % damaged | 12.34% | 3.73% | 3.73% | 3.55% | 0.85% | 1.80% | 0.30% |
| 60 Data Sources | | | | | | | |
| $N^{O.}$ of slices | 7557 | 7557 | 7557 | 7557 | 7557 | 7557 | 7557 |
| Damaged slices | 933.8 | 282.4 | 282.4 | 268.8 | 63.9 | 143.13 | 24.76 |
| % damaged | 12.36% | 3.74% | 3.74% | 3.56% | 0.85% | 1.89% | 0.33% |

**Table A.18 Video Slices**

| Video delays | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| mean | 38.7 | 38.9 | 38.9 | 38.8 | 38.9 | 38.8 | 38.9 |
| max. | 95.0 | 95.1 | 95.1 | 95.1 | 95.2 | 95.0 | 95.0 |
| 40 Data Sources | | | | | | | |
| mean | 40.7 | 41.0 | 40.8 | 40.8 | 41.5 | 40.9 | 41.0 |
| max. | 96.5 | 96.7 | 96.6 | 96.2 | 96.2 | 96.2 | 96.3 |
| 60 Data Sources | | | | | | | |
| mean | 41.0 | 41.3 | 41.1 | 41.1 | 41.2 | 41.2 | 41.3 |
| max. | 97.0 | 97.2 | 97.1 | 96.9 | 96.7 | 97.0 | 97.0 |

**Table A.19 Video Delays (in $\mu$s)**

| Speech Cells | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| generated | 149657 | 149657 | 149657 | 149657 | 149657 | 149657 | 149657 |
| received | 149643 | 149641 | 149642 | 149643 | 149642 | 149642 | 149641 |
| number lost | 13.29 | 15.26 | 14.41 | 13.522 | 15.067 | 14.811 | 15.278 |
| % loss | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% |
| 40 Data Sources | | | | | | | |
| generated | 149762 | 149762 | 149762 | 149762 | 149762 | 149762 | 149762 |
| received | 149747 | 149744 | 149745 | 149745 | 149744 | 149744 | 149744 |
| number lost | 15.2 | 18.233 | 16.667 | 16.411 | 18.033 | 17.5 | 18.41 |
| % loss | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% |
| 60 data sources | | | | | | | |
| generated | 144121 | 144121 | 144121 | 144121 | 144121 | 144121 | 144121 |
| received | 144106 | 144102 | 144104 | 144104 | 144102 | 144103 | 144102. |
| number lost | 15.07 | 19.11 | 17.12 | 17.12 | 18.72 | 18.06 | 18.84 |
| % loss | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% | 0.01% |

**Table A.20 Speech Cells**

| Speech Delays | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| mean | 44.5 | 44.6 | 44.6 | 44.6 | 44.6 | 44.6 | 44.6 |
| max. | 98.3 | 98.1 | 98.1 | 97.7 | 97.7 | 97.8 | 97.9 |
| 40 Data Sources | | | | | | | |
| mean | 46.6 | 47.0 | 46.9 | 46.9 | 47.0 | 46.9 | 47.0 |
| max. | 100 | 100 | 99.9 | 100 | 100 | 100 | 100 |
| 60 Data Sources | | | | | | | |
| mean | 48.3 | 48.4 | 48.3 | 48.3 | 48.4 | 48.4 | 48.4 |
| max. | 101 | 100 | 100 | 101 | 101 | 101 | 101 |

**Table A.21 Speech Delays (in $\mu$s)**

| Utilisation | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| Switch 1 - port 3 | 10.8% | 11.1% | 10.9% | 10.9% | 11.1% | 11% | 11.1% |
| Switch 2 - port 2 | 22.5% | 23.1% | 22.8% | 22.7% | 23.1% | 22.9% | 23.1% |
| Switch 3 - port 3 | 11.9% | 12.2% | 12% | 12% | 12.2% | 12.1% | 12.2% |
| Switch 4 - port 3 | 17.7% | 18.1% | 17.9% | 17.9% | 18.1% | 17.9% | 18.1% |
| Switch 4 - port 4 | 5% | 5.15% | 5.1% | 5.1% | 5.14% | 5.12% | 5.15% |
| Switch 5 - port 2 | 22.5% | 23.1% | 22.8% | 22.8% | 23.1% | 22.9% | 23.1% |
| Switch 5 - port 4 | 6.2% | 6.4% | 6.3% | 6.3% | 6.35% | 6.33% | 6.36% |
| 40 Data Sources | | | | | | | |
| Switch 1 - port 3 | 13.9% | 14.2% | 14.0% | 14.0% | 14.1% | 14.1% | 14.2% |
| Switch 2 - port 2 | 31.5% | 32.1% | 31.8% | 31.7% | 32.0% | 31.9% | 32.0% |
| Switch 3 - port 3 | 17.9% | 18.1% | 18.0% | 18.0% | 18.1% | 18.1% | 18.2% |
| Switch 4 - port 3 | 24.7% | 25.2% | 24.9% | 24.9% | 25.1% | 25.0% | 25.1% |
| Switch 4 - port 4 | 7.1% | 7.2% | 7.1% | 7.1% | 7.2% | 7.2% | 7.2% |
| Switch 5 - port 2 | 31.5% | 32.1% | 31.8% | 31.7% | 32.0% | 31.9% | 32.0% |
| Switch 5 - port 4 | 8.5% | 8.6% | 8.5% | 8.5% | 8.6% | 8.5% | 8.6% |
| 60 Data Sources | | | | | | | |
| Switch 1 - port 3 | 18.9% | 19.2% | 19.0% | 19.0% | 19.1% | 19.1% | 19.1% |
| Switch 2 - port 2 | 38.0% | 38.6% | 38.3% | 38.2% | 38.5% | 38.4% | 38.5% |
| Switch 3 - port 3 | 18.9% | 19.1% | 19.0% | 19.0% | 19.1% | 19.0% | 19.1% |
| Switch 4 - port 3 | 28.8% | 29.2% | 29.0% | 28.9% | 29.2% | 29.1% | 29.2% |
| Switch 4 - port 4 | 9.0% | 9.1% | 9.0% | 9.0% | 9.1% | 9.0% | 9.1% |
| Switch 5 - port 2 | 38.0% | 38.6% | 38.3% | 38.2% | 38.5% | 38.4% | 38.5% |
| Switch 5 - port 4 | 10.5% | 10.7% | 10.6% | 10.6% | 10.6% | 10.6% | 10.7% |

**Table A.22 Utilisation for the Output Ports at each Switch**

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 20 Data Sources | | | | | | | |
| Switch 1 - port 3 | 795156 | 814253 | 803785 | 802170 | 811851 | 807846 | 813322 |
| Switch 2 - port 2 | 1654188 | 1696323 | 1672956 | 1669356 | 1690971 | 1681794 | 1694102 |
| Switch 3 - port 3 | 874803 | 893836 | 883337 | 881702 | 891408 | 889383 | 892845 |
| Switch 4 - port 3 | 1299120 | 1329738 | 1312860 | 1310222 | 1325848 | 1319299 | 1328182 |
| Switch 4 - port 4 | 370839 | 378351 | 374262 | 373651 | 377411 | 375930 | 377885 |
| Switch 5 - port 2 | 1654188 | 1696323 | 1672956 | 1669356 | 1690971 | 1681794 | 1694102 |
| Switch 5 - port 4 | 458525 | 467254 | 462471 | 461709 | 466155 | 464285 | 466843 |
| 40 Data Sources | | | | | | | |
| Switch 1 - port 3 | 1020649 | 1040001 | 1029457 | 1027762 | 1037501 | 1033741 | 1039028 |
| Switch 2 - port 2 | 2314010 | 2355655 | 2332765 | 2329017 | 2350057 | 2341771 | 2353367 |
| Switch 3 - port 3 | 1314650 | 1334261 | 1323340 | 1321457 | 1331363 | 1327340 | 1332839 |
| Switch 4 - port 3 | 1814749 | 1845996 | 1828716 | 1825861 | 1841665 | 1835380 | 1844066 |
| Switch 4 - port 4 | 520551 | 528266 | 524082 | 523359 | 527199 | 515701 | 527800 |
| Switch 5 - port 2 | 2314010 | 2355655 | 2332765 | 2329017 | 2350057 | 2341771 | 2353367 |
| Switch 5 - port 4 | 619937 | 628912 | 633981 | 623162 | 627657 | 625850 | 628288 |
| 60 Data Sources | | | | | | | |
| Switch 1 - port 3 | 1385487 | 1404876 | 1394131 | 1392180 | 1401947 | 1397686 | 1403265 |
| Switch 2 - port 2 | 2787666 | 2830191 | 2806505 | 2801762 | 2823123 | 2813844 | 2826098 |
| Switch 3 - port 3 | 1384872 | 1404518 | 1393631 | 1391298 | 1401077 | 1396885 | 1402490 |
| Switch 4 - port 3 | 2112845 | 2144217 | 2126737 | 2123330 | 2139132 | 2132378 | 2141378 |
| Switch 4 - port 4 | 657514 | 665177 | 661025 | 660148 | 663892 | 662193 | 664376 |
| Switch 5 - port 2 | 2787666 | 2830191 | 2806505 | 2801764 | 2823123 | 2813844 | 2826098 |
| Switch 5 - port 4 | 773322 | 782426 | 777260 | 776185 | 780813 | 778821 | 781484 |

**Table A.23 Average Number of Cells Served in a Reporting Interval**

# Appendix II – Simulation Code

```
MAIN MODULE atmslb2;    {new expanded network version}

FROM SimMod  IMPORT SimTime, StartSimulation;
FROM UtilMod IMPORT ClockTimeSecs;
FROM RandMod IMPORT RandomObj, FetchSeed;
FROM atmnw IMPORT ATMnetwork;
FROM usersite IMPORT usersitemanager;

VAR {global variables}
        runlen              : REAL;     {length of simulation run}
        wait                : REAL;     {waiting time between reports}
        numberofswitches    : INTEGER;  {number of switches in the network}
        Numberofsites       : INTEGER;  {number of user sites}
        randomseed          : INTEGER;
        {random number entered from batch file and passed as a parameter to modules}

{MAIN PROGRAM}
BEGIN
        OUTPUT("Super LB - DUAL QUEUE - with thresholds");
        OUTPUT("Enter runlength and report interval in seconds");
        OUTPUT("2 x burst params - calc mean");

        INPUT(runlen);
        INPUT(wait);

        OUTPUT("wait between stats ", wait, " run length = ", runlen, " in seconds");
        {OUTPUT("enter random seed 1 - 10");}
        INPUT(randomseed);          {random seed entered from batch file}

        numberofswitches := 5;
        Numberofsites := 5;
        NEW(ATMnetwork);                        {generate a new ATM network object}
        NEW(usersitemanager);                   {generate a user site manager object}
        TELL ATMnetwork TO Monitor(runlen, wait);       {set monitor objects running to}
        TELL usersitemanager TO Monitor(runlen, wait);  {tell all objects when to report}

        {initalise ATM network and user site manager objects}
        ASK ATMnetwork TO Initalise(numberofswitches, Numberofsites);
        ASK usersitemanager TO Initalise(Numberofsites, randomseed);

        StartSimulation;                {start the simulation}

        OUTPUT();
        OUTPUT("simulation end - made it back to main module ");
END MODULE. {main module ATM}
```

```
DEFINITION MODULE types;
{Definition module for global type declarations of the ATM cell}

TYPE
  sourceType = (video, speech, data, data1, data2, data3, data4, data5, data6);
                                          {source identifier for cells}

   cell = RECORD;                                          {an ATM cell type declaration}
        msgstartTime, cellstartTime  : REAL;       {time stamps}
        source                       : sourceType; {cell origin}
        VCIlabel                     : CHAR;       {address label}
        typeofcell                   : cellType;   {identifiers for the cell}
        origin                       : INTEGER;    {site number of sender}
        CLP                          : INTEGER;    { priority =  (0, 1)}
        GFCfield                     : INTEGER;    {set to 1 or 0 – to reset TEC}
        VCid                         : INTEGER;    {identifier for VC}
        prev, next                   : cell;       {pointers to previous and next records}
        cellcount                    : INTEGER;    {cell number}
        dest                         : INTEGER;    {destination user site}

        END RECORD;

VAR

END MODULE.
```

```
DEFINITION MODULE atmnw;     {Definition module for the ATM network}

FROM SimMod  IMPORT SimTime, StartSimulation;
FROM UtilMod IMPORT ClockTimeSecs;
FROM RandMod IMPORT RandomObj, FetchSeed;
FROM StatMod IMPORT StatObj, RStatObj, IStatObj, ITimedStatObj, TSINTEGER,
        SINTEGER, RTimedStatObj, TSREAL, SREAL;
FROM usersite IMPORT usersitemanager;
FROM types IMPORT ALL cellType, ALL cell ;    {import cell definition}

CONST
        capacity = 155.52E6;                  {capacity of network links}
        slot = 424.0 / capacity;              {duration of a slot – 2.7 micro secs}
        recipofslot = 1.0 / slot;             {used to speed up calculations}
        cellsize = 424.0;                     {total cell size}
        cellcapacity = capacity / cellsize;   {capacity of link in cells}
        cellpayload = 384.0;                  {payload of cell}
        servicetime = 1.0;                    {is a constant - 1 slot}
        switchingtime = 1.0;                  {is a constant for switches - 1 slot}
        transmissionTime = 1.0;               {time to transmit 1 cell}

TYPE
   dynamictabledefn = RECORD      {table entry for dynamic routing table}
        IPport, OPport          : INTEGER;
                                  {input and output ports to use for routing purposes}
        oldVCI                  : CHAR;       {previous VCI label for cell header}
        newVCI                  : CHAR;       {next VCI label for the cell headers}
        lastswitch              : BOOLEAN;  {identifys the last switch for this call}
   END RECORD;

   networktabledefn = RECORD;      {routing layout for network}
        switch                  : INTEGER;   {number of the switch}
        OPport                  : INTEGER;   {the output port to use}
        Attachedswitch          : INTEGER;   {the number of the next switch for this route}
        IPport                  : INTEGER;   {input port to use}
        usersite                : CHAR;      {identifier for the user site}
        sitenumber              : INTEGER;   {the site number}
   END RECORD;

   flagType = (idle, busy);     {type for the server at each OP port}


{OBJECTS}

{ATM network declarations}
```

```
  ATMswitchObj = OBJECT;
        switchidentity     : INTEGER;
        deletedcellcount, servedcellcount : ARRAY INTEGER OF SINTEGER;
                        {counters for dropped an served cells}
        outputQtail        : ARRAY INTEGER OF cell;              {switch queues pointers}
        outputQhead        : ARRAY INTEGER OF cell;
        qlen               : ARRAY INTEGER OF INTEGER;           {queue length counter}
        resetqlen          : ARRAY INTEGER OF TSINTEGER;         {statistical queue counter}
        flag               : ARRAY INTEGER OF flagType;          {server serving flag}

        ASK METHOD Initalise(IN switchidentifier : INTEGER);
        TELL METHOD Inputcontroller(IN ATMcell : cell; IN IPport : INTEGER);
        ASK METHOD  AddtoOPqueue(IN ATMcell : cell; IN OPport : INTEGER);
        TELL METHOD ServeOPportQ(IN queue : INTEGER); {output controller}
  END OBJECT;

  ATMnetworkObj = OBJECT;
        ASK METHOD Initalise(IN numofswitches : INTEGER;
                                                IN numberofsites : INTEGER);
        ASK METHOD AccessNW(IN ATMcell : cell; IN switchNo : INTEGER;
                                        IN portNo : INTEGER);
        TELL METHOD Monitor(IN runlen : REAL; IN wait : REAL);
        ASK METHOD NWStats;
  END OBJECT;
{**********end of ATM network declarations**********}

{end of type declarations}

VAR {global variables}
        {declaring objects and arrays}
        ATMnetwork      : ATMnetworkObj;
        ATMswitch       : ARRAY INTEGER OF ATMswitchObj;
        networkroutingtable : ARRAY INTEGER OF networktabledefn;
        dynamicroutingtable : ARRAY INTEGER OF dynamictabledefn;

        {utilization for each queue in each switch}
        ut                  : ARRAY INTEGER OF ARRAY INTEGER OF TSINTEGER;

        dynamicmaxentry     : INTEGER;  {max entry for dynamic routing table}
        networkmaxentry     : INTEGER;  {max entry for N/W routing table}
        endrun              : BOOLEAN;  {end of run flag - used to stop run}
        numberofswitches    : INTEGER;  {No of switches this run}
        numberofusersites   :  INTEGER; {No of usersites this run}
        numberofports       : INTEGER;  {No ports per switch}

END MODULE. {Definition module for ATM network}
```

```
IMPLEMENTATION MODULE atmnw;    {expanded network (5 switches) with new routing tables etc}

FROM SimMod  IMPORT SimTime, StartSimulation;
FROM UtilMod IMPORT ClockTimeSecs;
FROM RandMod IMPORT RandomObj, FetchSeed;
FROM StatMod IMPORT StatObj, RStatObj, IStatObj, ITimedStatObj, TSINTEGER, SINTEGER,
                                                          RTimedStatObj, TSREAL, SREAL;
FROM usersite IMPORT usersitemanager;
FROM types IMPORT  ALL sourceType, ALL cellType, ALL cell;

{IMPLEMENTATION}
{----Procedure Declarations-------}
{********Code for Objects**************}

OBJECT ATMswitchObj;

  ASK METHOD Initalise(IN switchidentifier : INTEGER);

  VAR  index : INTEGER;

  BEGIN
        switchidentity := switchidentifier;
        NEW(outputQhead, 1 .. numberofports);      {1 to 4 for a 4x4 switch}
        NEW(outputQtail, 1 .. numberofports);      {queue stuff}
        NEW(qlen, 1 .. numberofports);             {queue length counter}
        NEW(resetqlen, 1 .. numberofports);        {queue length stats}

        NEW(flag, 1 .. numberofports);             {create server flags for the array of output ports}
         FOR index := 1 TO numberofports
               flag[index] := idle;                {initalise flag to idle}
         END FOR;
        NEW(deletedcellcount, 1 .. numberofports); {counts mis-routed cells}
        NEW(servedcellcount, 1 .. numberofports);  {FOR TESTING only}

END METHOD; {Initalise}
 {--------------------}

TELL METHOD Inputcontroller(IN ATMcell : cell; IN IPport : INTEGER);
{Input controller looks up VCI labels and adds header that directs}
{cell to the correct OP port within the switch}

VAR  OPport   : INTEGER;                   {pass cell to this output port}
       endsearch : BOOLEAN;                {stops search of routing tables}
       index     : INTEGER;                {indexes for searching the routing tables}
       index1    : INTEGER;

BEGIN
       endsearch := FALSE;                 {initlaise the variables for searching the routing tables}
       index     := 1;

       {find next VCI label and the OP port}
       WHILE (endsearch = FALSE) AND (index <= dynamicmaxentry);

         {match IP port for this connection}
          IF (dynamicroutingtable[index].IPport = IPport) AND
```

```
                                        (dynamicroutingtable[index].oldVCI = ATMcell.VCIlabel)
                                {then match old VCI with VCI label in cell and get the new VCI label}
            endsearch := TRUE;
            ATMcell.VCIlabel := dynamicroutingtable[index].newVCI;
            OPport := dynamicroutingtable[index].OPport;
        ELSE
            INC(index); {increment index if port does not match}
        END IF;

    END WHILE; {end search should now be TRUE}

    IF (endrun = TRUE) TERMINATE; END IF; {and do not add cell to queue}

    IF (endsearch = FALSE)  {then}
            DISPOSE(ATMcell);
            INC(deletedcellcount[IPport]);
            OUTPUT("disposing of address error cell at switch ", switchidentity);
    ELSE
            WAIT DURATION switchingtime; END WAIT; {switching through to Q}
            AddtoOPqueue(ATMcell, OPport);          {passes cell to the OPport queue}
    END IF;

END METHOD;  {Inputcontroller}
{--------------------}

ASK METHOD AddtoOPqueue(IN ATMcell : cell; IN OPport : INTEGER);
    {passes cells to correct output port and adds it to the queue}

BEGIN
    INC(qlen[OPport]);              {increment queue counter}
    INC(resetqlen[OPport]);         {update stats}
    IF (qlen[OPport] = 1)                       {then 1st item on queue & call server}
      outputQtail[OPport] := ATMcell;           {head and tail both}
      outputQhead[OPport] := ATMcell;           {point to the first cell}
      ATMcell.prev      := NILREC;
      ATMcell.next      := NILREC;
      IF (flag[OPport] = idle) {then}
            flag[OPport] := busy;       {set flag to busy}
            ServeOPportQ(OPport);       {call server for this OP port}
      END IF;                           {check for empty queue, but server still serving}
    ELSE    {add cell to end of queue}
      ATMcell.next := NILREC;                   {last cell}
      outputQtail[OPport].next := ATMcell;
      ATMcell.prev := outputQtail[OPport];
      outputQtail[OPport] := ATMcell;           {tail points to new cell}
    END IF;

END METHOD;     {add to output port}
{--------------------}

TELL METHOD ServeOPportQ(IN queue : INTEGER);  {output controller}

VAR servedATMcell       : cell;         {the cell removed from the output queue}
    sitenumber          : INTEGER;      {site number to send it to}
    endsearch           : BOOLEAN;      {flag for searching rotuing tables}
```

```
label                   : CHAR;      {VCI label from tables}
IPport                  : INTEGER;   {input port used by this cell}
destination             : INTEGER;   {destination user site}
index, index2, index3   : INTEGER;   {indexes for arrays}

BEGIN

    INC(ut[queue, switchidentity]);     {update utilization Stats}
    WHILE (qlen[queue] > 0)             {server serves while there are cells in the queue}

    IF (endrun = TRUE) EXIT; END IF;            {terminate simulation}

     servedATMcell := outputQhead[queue];               {remove cell from the queue}
     outputQhead[queue] := outputQhead[queue].next;     {reset pointer}
     DEC(qlen[queue]);                                  {decrement Q length}
     DEC(resetqlen[queue]);                             {decrement statistical Q length}
     INC(servedcellcount[queue]);                       {temp var to count cells served}

     WAIT DURATION transmissionTime; END WAIT;

     {need to look up the destination associated with this op port}
     index := 1;
     endsearch := FALSE;
     label := servedATMcell.VCIlabel;

     WHILE (endsearch = FALSE) AND (index <= dynamicmaxentry);
      {look up VCI label in dynamicrouting table - and find out if this is the last switch}

       IF (label = dynamicroutingtable[index].newVCI)
            AND (dynamicroutingtable[index].lastswitch = TRUE);
            {is label same as new VCI in table then check if last switch then this cell is for a user site}
       index2 := 1; {initalise index for networkrouting table}
       {find usersite number and pass cell to that usersite}

       WHILE (endsearch = FALSE) AND (index2 <= networkmaxentry);
            {look up routing table for usersite number}

         IF (networkroutingtable[index2].usersite = label);
            endsearch  := TRUE;
            sitenumber := networkroutingtable[index2].sitenumber;
         ELSE
            INC(index2);  {look at next item in NW routing table}
         END IF;

       END WHILE;

       ASK usersitemanager TO Accessusersite(servedATMcell, sitenumber);

ELSIF  (label = dynamicroutingtable[index].newVCI) AND
                    (dynamicroutingtable[index].lastswitch = FALSE);
            {not last switch, so pass to next switch}

       index3 := 1;                         {initalise array indexes}
       index2 := 1;                         {find next switch and the IP port to use}
       WHILE (endsearch = FALSE) AND (index3 <= dynamicmaxentry);
```

```
                {find old VCI label and get IP port}
                IF (label = dynamicroutingtable[index3].oldVCI) {then}

                    IPport := dynamicroutingtable[index3].IPport;

                    {find destination switch to pass cell to}
                    WHILE (endsearch = FALSE) AND (index2 <= networkmaxentry);

                      {match OPort and IPport in networkrouting table}
                      IF (queue = networkroutingtable[index2].OPport) AND
                                  (IPport = networkroutingtable[index2].IPport) AND
                                  (switchidentity = networkroutingtable[index2].switch)
                    {then}
                            destination :=
                            networkroutingtable[index2].attachedswitch;
                            endsearch := TRUE;
                      ELSE
                            INC(index2);  {look at next item in NWrouting table}
                      END IF;

                    END WHILE;

                    TELL ATMswitch[destination] TO Inputcontroller(servedATMcell, IPport);

                  ELSE {no match for VCI - look at next dynamicrouting entry}
                      INC(index3);
                END IF; {to find IP port and destination switch}

              END WHILE;

ELSE          {look at next item in dynamic routing table}

        INC(index);

            END IF;

            END WHILE; {matches cell label to dynamicrouting table new VCIlabel}

        END WHILE;      {loops around if there is another cell to serve}

            DEC(ut[queue, switchidentity]);                 {utilization of switch}
            flag[queue] := idle;                            {release flag for this OP port}
            IF (endrun = TRUE) TERMINATE; END IF;           {stop server, simulation end}

    END METHOD;  {ServeOPportQ}
    {----------------------}

   END OBJECT; {ATMswitchObj}
{------------end ATM switch Object--------------------}
OBJECT ATMnetworkObj;

    ASK METHOD Initalise(IN numofswitches : INTEGER; IN numberofsites : INTEGER);

    VAR  switch          : INTEGER;
         aswitch         : ATMswitchObj;
         NWrecord        : networktabledefn;
```

```
    dynamicrec        : dynamictabledefn;

BEGIN
    numberofswitches := numofswitches;
    numberofports     := 4;
    numberofusersites := numberofsites;

    OUTPUT("No switches ", numberofswitches, " number of ports ", numberofports);

    {enter information on network topology}
    networkmaxentry := 11;
    NEW(networkroutingtable, 1 .. networkmaxentry);

    FOR switch := 1 TO networkmaxentry        {initalising network routing table}
          NEW(NWrecord);
          networkroutingtable[switch] := NWrecord;
    END FOR;

    networkroutingtable[1].switch := 1;                {S1 to S4}
    networkroutingtable[1].attachedswitch := 4;        {OP3 > IP1}
    networkroutingtable[1].OPport := 3;                {switch to switch}
    networkroutingtable[1].IPport := 1;
    networkroutingtable[1].usersite := ' ';            {no attached user site}
    networkroutingtable[1].sitenumber := 0;

    networkroutingtable[2].switch := 4;                {S4 to U5}
    networkroutingtable[2].attachedswitch := 0;        {no attached switch}
    networkroutingtable[2].OPport := 4;                {destination U5}
    networkroutingtable[2].IPport := 0;                {pass to usersite 5}
    networkroutingtable[2].usersite := 'c';
    networkroutingtable[2].sitenumber := 5;

    networkroutingtable[3].switch := 5;                {S5 to S2}
    networkroutingtable[3].attachedswitch := 2;        {OP2 > IP4}
    networkroutingtable[3].OPport := 2;                {switch to switch}
    networkroutingtable[3].IPport := 4;
    networkroutingtable[3].usersite := ' ';            {no user site for route}
    networkroutingtable[3].sitenumber := 0;

    networkroutingtable[4].switch := 2;                {S2 to U3}
    networkroutingtable[4].attachedswitch := 0;
    networkroutingtable[4].OPport := 2;
    networkroutingtable[4].IPport := 0;                {pass to usersite 3}
    networkroutingtable[4].usersite := 'h';
    networkroutingtable[4].sitenumber := 3;

    networkroutingtable[5].switch := 4;                {S4 to S5}
    networkroutingtable[5].attachedswitch := 5;
    networkroutingtable[5].OPport := 3;                {OP3 > IP3}
    networkroutingtable[5].IPport := 3;
    networkroutingtable[5].usersite := ' ';            {no user site for route}
    networkroutingtable[5].sitenumber := 0;

    networkroutingtable[6].switch := 5;                {S5 to U2}
    networkroutingtable[6].attachedswitch := 0;
    networkroutingtable[6].OPport := 4;
```

```
networkroutingtable[6].IPport := 0;                    {pass to usersite 2}
networkroutingtable[6].usersite := 'e';
networkroutingtable[6].sitenumber := 2;

networkroutingtable[7].switch := 3;                    {S3 to S4}
networkroutingtable[7].attachedswitch := 4;
networkroutingtable[7].OPport := 3;                    {OP3 to IP2}
networkroutingtable[7].IPport := 2;
networkroutingtable[7].usersite := ' ';                {no attached user site}
networkroutingtable[7].sitenumber := 0;

networkroutingtable[8].switch := 4;                    {S4 to U5}
networkroutingtable[8].attachedswitch := 0;
networkroutingtable[8].OPport := 4;
networkroutingtable[8].IPport := 0;                    {pass to usersite 5}
networkroutingtable[8].usersite := 'C';
networkroutingtable[8].sitenumber := 5;

networkroutingtable[9].switch := 5;                    {S5 to U2}
networkroutingtable[9].attachedswitch := 0;
networkroutingtable[9].OPport := 4;
networkroutingtable[9].IPport := 0;                    {pass to usersite 2}
networkroutingtable[9].usersite := 'E';
networkroutingtable[9].sitenumber := 2;

networkroutingtable[10].switch := 2;                   {S2 to U3}
networkroutingtable[10].attachedswitch := 0;
networkroutingtable[10].OPport := 2;
networkroutingtable[10].IPport := 0;                   {pass to usersite 3}
networkroutingtable[10].usersite := 'G';
networkroutingtable[10].sitenumber := 3;

networkroutingtable[11].switch := 2;                   {S2 to U3}
networkroutingtable[11].attachedswitch := 0;
networkroutingtable[11].OPport := 2;
networkroutingtable[11].IPport := 0;                   {pass to usersite 3}
networkroutingtable[11].usersite := 'g';
networkroutingtable[11].sitenumber := 3;


NEW(ATMswitch, 1 .. numberofswitches);                 {create ATM switches}
FOR switch := 1 TO numberofswitches
        NEW(aswitch);
        ATMswitch[switch] := aswitch;
        ASK ATMswitch[switch] TO Initalise(switch);
END FOR;

{=============================================}

dynamicmaxentry := 20;   {entries in the dynamicroutingtable}
NEW(dynamicroutingtable, 1..dynamicmaxentry);     {creating the dynamic routing table}
FOR switch := 1 TO dynamicmaxentry
        NEW(dynamicrec);
        ynamicroutingtable[switch] := dynamicrec;
END FOR;
```

```
dynamicroutingtable[1].IPport      := 4;          {At switch 5}
dynamicroutingtable[1].OPport      := 2;          {U2 > U3}  {*0*}
dynamicroutingtable[1].oldVCI      := 'z';        {S5 > S2}
dynamicroutingtable[1].newVCI      := 'f';
dynamicroutingtable[1].lastswitch := FALSE;

dynamicroutingtable[2].IPport      := 4;          {At switch 2}
dynamicroutingtable[2].OPport      := 2;          {U2 > U3}
dynamicroutingtable[2].oldVCI      := 'f';        {S2 > U3}
dynamicroutingtable[2].newVCI      := 'h';
dynamicroutingtable[2].lastswitch := TRUE;
{-------------------------------------------------------}

dynamicroutingtable[3].IPport      := 2;          {At switch 1}
dynamicroutingtable[3].OPport      := 3;          {U1 > U5}
dynamicroutingtable[3].oldVCI      := 'a';        {S1 > S4}
dynamicroutingtable[3].newVCI      := 'b';
dynamicroutingtable[3].lastswitch := FALSE;

dynamicroutingtable[4].IPport      := 1;          {At switch 4}
dynamicroutingtable[4].OPport      := 4;          {U1 > U5}
dynamicroutingtable[4].oldVCI      := 'b';        {S4 > U5}
dynamicroutingtable[4].newVCI      := 'c';
dynamicroutingtable[4].lastswitch := TRUE;
{-------------------------------------------------------}

dynamicroutingtable[5].IPport      := 2;          {At switch 1}
dynamicroutingtable[5].OPport      := 3;          {U1 >  U2}
dynamicroutingtable[5].oldVCI      := 'x';        {S1 > S4}
dynamicroutingtable[5].newVCI      := 'y';
dynamicroutingtable[5].lastswitch := FALSE;

dynamicroutingtable[6].IPport      := 1;          {At switch 4}
dynamicroutingtable[6].OPport      := 3;          {U1 to U2}
dynamicroutingtable[6].oldVCI      := 'y';        {S4 to S5}
dynamicroutingtable[6].newVCI      := 'd';
dynamicroutingtable[6].lastswitch := FALSE;

dynamicroutingtable[7].IPport      := 3;          {At switch 5}
dynamicroutingtable[7].OPport      := 4;          {U1 to U2}
dynamicroutingtable[7].oldVCI      := 'd';        {S5 to U2}
dynamicroutingtable[7].newVCI      := 'e';
dynamicroutingtable[7].lastswitch := TRUE;
{-------------------------------------------------------}

dynamicroutingtable[8].IPport      := 2;          {At switch 1}
dynamicroutingtable[8].OPport      := 3;          {U1 to U3}
dynamicroutingtable[8].oldVCI      := 'A';        {U1 to S1}
dynamicroutingtable[8].newVCI      := 'B';
dynamicroutingtable[8].lastswitch := FALSE;

dynamicroutingtable[9].IPport      := 1;          {At switch 4}
dynamicroutingtable[9].OPport      := 3;          {U1 to U3}
dynamicroutingtable[9].oldVCI      := 'B';        {S4 to S5}
dynamicroutingtable[9].newVCI      := 'D';
dynamicroutingtable[9].lastswitch := FALSE;
```

```
dynamicroutingtable[10].IPport      := 3;          {At switch 5}
dynamicroutingtable[10].OPport      := 2;          {U1 to U3}
dynamicroutingtable[10].oldVCI      := 'D';        {S5 to S2}
dynamicroutingtable[10].newVCI      := 'F';
dynamicroutingtable[10].lastswitch := FALSE;

dynamicroutingtable[11].IPport      := 4;          {At switch 2}
dynamicroutingtable[11].OPport      := 2;          {U1 to U3}
dynamicroutingtable[11].oldVCI      := 'F';        {S2 to U3}
dynamicroutingtable[11].newVCI      := 'G';
dynamicroutingtable[11].lastswitch := TRUE;
{-----------------------------------------------------}

dynamicroutingtable[12].IPport      := 2;          {At switch 3}
dynamicroutingtable[12].OPport      := 3;          {U4 to U5}
dynamicroutingtable[12].oldVCI      := 'm';        {S3 to S4}
dynamicroutingtable[12].newVCI      := 'n';
dynamicroutingtable[12].lastswitch := FALSE;

dynamicroutingtable[13].IPport      := 2;          {At switch 4}
dynamicroutingtable[13].OPport      := 4;          {U4 to U5}
dynamicroutingtable[13].oldVCI      := 'n';        {S4 to U5}
dynamicroutingtable[13].newVCI      := 'C';
dynamicroutingtable[13].lastswitch := TRUE;
{-----------------------------------------------------}

dynamicroutingtable[14].IPport      := 2;          {At switch 3}
dynamicroutingtable[14].OPport      := 3;          {U4 to U2}
dynamicroutingtable[14].oldVCI      := 'M';        {S3 to S4}
dynamicroutingtable[14].newVCI      := 'N';
dynamicroutingtable[14].lastswitch := FALSE;

dynamicroutingtable[15].IPport      := 2;          {At switch 4}
dynamicroutingtable[15].OPport      := 3;          {U4 to U2}
dynamicroutingtable[15].oldVCI      := 'N';        {S4 to S5}
dynamicroutingtable[15].newVCI      := 'O';
dynamicroutingtable[15].lastswitch := FALSE;

dynamicroutingtable[16].IPport      := 3;          {At switch 5}
dynamicroutingtable[16].OPport      := 4;          {U4 to U2}
dynamicroutingtable[16].oldVCI      := 'O';        {S5 to U2}
dynamicroutingtable[16].newVCI      := 'E';
dynamicroutingtable[16].lastswitch := TRUE;
{-----------------------------------}

dynamicroutingtable[17].IPport      := 2;          {At switch 3}
dynamicroutingtable[17].OPport      := 3;          {U4 to U3}
dynamicroutingtable[17].oldVCI      := 'p';        {S3 to S4}
dynamicroutingtable[17].newVCI      := 'q';
dynamicroutingtable[17].lastswitch := FALSE;

dynamicroutingtable[18].IPport      := 2;          {At switch 4}
dynamicroutingtable[18].OPport      := 3;          {U4 to U3}
dynamicroutingtable[18].oldVCI      := 'q';        {S4 to S5}
dynamicroutingtable[18].newVCI      := 'o';
```

```
        dynamicroutingtable[18].lastswitch := FALSE;

        dynamicroutingtable[19].IPport      := 3;          {At switch 5}
        dynamicroutingtable[19].OPport      := 2;          {U4 to U3}
        dynamicroutingtable[19].oldVCI      := 'o';        {S5 to S2}
        dynamicroutingtable[19].newVCI      := 'r';
        dynamicroutingtable[19].lastswitch := FALSE;

        dynamicroutingtable[20].IPport      := 4;          {At switch 2}
        dynamicroutingtable[20].OPport      := 2;          {U4 to U3}
        dynamicroutingtable[20].oldVCI      := 'r';        {S2 to U3}
        dynamicroutingtable[20].newVCI      := 'g';
        dynamicroutingtable[20].lastswitch := TRUE;
        {-----------------------------------}

        NEW(ut, 1 .. numberofports, 1 .. numberofswitches); {initalise the variable to monitor utilisation}

END METHOD; {Initalise}
 {-------------------}

 ASK METHOD AccessNW(IN ATMcell : cell; IN switchNo : INTEGER; IN portNo : INTEGER);
 {a usersite can not access the ATM switch directly - needs to pass cell to ATM network Object first}

 BEGIN
        TELL ATMswitch[switchNo] TO Inputcontroller(ATMcell, portNo);
 END METHOD; {Access network method}
 {-------------------}

 TELL METHOD Monitor(IN runlen : REAL; IN interval : REAL);
 {run length and interval are in seconds - convert to slots}
 {synchronise reporting intervals}

 BEGIN
        OUTPUT("ATM n/w monitor starting ");
        runlen := runlen * recipofslot;        {convert times into slots}
        interval := interval * recipofslot;
        endrun := FALSE;
        WHILE (SimTime() < runlen)
           WAIT DURATION interval; END WAIT;
           OUTPUT("asking switches to OP Stats - Time = ", SimTime(), " slots, ", (SimTime() * slot),
                                                                    " seconds ")
           OUTPUT();
           ASK SELF NWStats;
        END WHILE;

        endrun := TRUE;            {sets end of run flag when simulation is completed}
        TERMINATE;
 END METHOD; {end Monitor method}

 {----------------------}

 ASK METHOD NWStats;        {Method to output statistics to file at every reporting interval}
 VAR   switch : INTEGER;
        port : INTEGER;
        realtime : REAL;
 BEGIN
```

```
        {stats}
  realtime := SimTime();              {convert slotted simulation time into real time}
  OUTPUT("SimTime = ", realtime, " in slots and in seconds = ", (realtime * slot));
  OUTPUT();

   FOR switch := 1 TO numberofswitches;  {loop through all switches and ports to get Stats}
     OUTPUT();
     FOR port := 1 TO numberofports;

        IF ((switch = 1) AND (port = 3)) OR
                ((switch = 2) AND (port = 2)) OR
                ((switch = 3) AND (port = 3)) OR
                ((switch = 4) AND (port = 3)) OR
                ((switch = 4) AND (port = 4)) OR
                ((switch = 5) AND (port = 2)) OR
                ((switch = 5) AND (port = 4))  {then}

OUTPUT("---SWITCH  ", switch, " port ", port);
OUTPUT("cells still in Q =     ", ATMswitch[switch].qlen[port] );
OUTPUT("mean Q len =          ",
                ASK(GETMONITOR(ATMswitch[switch].resetqlen[port], ITimedStatObj)) Mean());
OUTPUT("max Q len =           ",
                ASK(GETMONITOR(ATMswitch[switch].resetqlen[port], ITimedStatObj)) Maximum );
{*********}
IF (ATMswitch[switch].deletedcellcount[port] > 0) {then}
   OUTPUT("Error Cells deleted  ",
        (ASK(GETMONITOR(ATMswitch[switch].deletedcellcount[port], IStatObj)) Count));
END IF; {only output if cell was deleted}
OUTPUT("cells thru port ", port, "       ",
        ASK(GETMONITOR(ATMswitch[switch].servedcellcount[port], IStatObj)) Count);
OUTPUT();
OUTPUT("Mean utilization      ", (ASK(GETMONITOR(ut[port, switch], ITimedStatObj)) Mean()));
OUTPUT("Max utilization       ", (ASK(GETMONITOR(ut[port, switch], ITimedStatObj)) Maximum));
OUTPUT();
END IF;

   END FOR;     {end of loops for outputing stats}
 END FOR;
OUTPUT();

{RESETTING STATS}
FOR switch := 1 TO numberofswitches;
   FOR port := 1 TO numberofports;
        ASK(GETMONITOR(ut[port, switch], ITimedStatObj)) TO Reset;
        ASK(GETMONITOR(ATMswitch[switch].resetqlen[port], ITimedStatObj)) TO Reset;
        ASK(GETMONITOR(ATMswitch[switch].deletedcellcount[port], IStatObj))     TO Reset;
        ASK(GETMONITOR(ATMswitch[switch].servedcellcount[port], IStatObj)) TO Reset;
   END FOR;
END FOR;
END METHOD; {NWStats}
   {--------------------}
END OBJECT; {ATMnetworkObj}
{------------end ATM network Object--------------------}
{**********************end ATM network******************}

END MODULE. {IMPLEMENTATION for ATM  network}
```

```
DEFINITION MODULE usersite;
{cyclic server model - new speech model super LB}
{usersite with dual queues and cyclic server - priority service for RT Q}
{expanded ATM network with new routing tables etc}
{adding LB and data throttleback}

FROM SimMod  IMPORT SimTime, StartSimulation;
FROM UtilMod IMPORT ClockTimeSecs;
FROM RandMod IMPORT RandomObj, FetchSeed;
FROM StatMod IMPORT StatObj, RStatObj, IStatObj, ITimedStatObj, TSINTEGER, SINTEGER,
                                                    RTimedStatObj, TSREAL, SREAL;
FROM MathMod IMPORT LOG10;
FROM types IMPORT ALL sourceType, ALL cellType, ALL cell;
FROM atmnw IMPORT ATMnetwork;

CONST linkcapacity = 45.0E6;              {access link speed}
      capacity = 155.52E6;                {network speed}
      cellsize = 424.0;                   {size of an ATM cell}
      slot = cellsize / capacity;         {duration of an ATM cell in secs}
      aslot = 1.0;                        {absolute time - ref 1 slot}

      recipofslot = 1.0 / slot;           {reciprocal of slot - to speed up calcs}
      serviceTime = (cellsize / linkcapacity) / slot;  {service time in slots for this link access (3.456 slots)}

      linkinslots = linkcapacity / cellsize;  {capacity of link in cells}
      capacityinslots = capacity / cellsize;  {convert into slotted time}

      cellpayload = 384.0;                {actual payload of cell - 48 bytes}
      speechpayload = 376.0;              {payload of speech cell - 47 bytes}

      videocellpayload = 352.0;           {payload of video cell - 4 bytes overhead}
      videorefresh = (1.0/30.0);          {1/30th sec in seconds}
      speechrate = 64.0E3;                {rate for speech coding (in Kb/s)}
      speechpacketization = (cellpayload / speechrate) * recipofslot;   {speech packetization is same for all}

      Silence = 1.67 * recipofslot;       {in slots}
      Talkspurt = 1.34;                   {in seconds - must be in secs for coding}

      phonecallduration = 180.0 * recipofslot; {average length of call in slots}

TYPE
   Status = (free, engaged);              {status for phone lines}
   flagType = (idle, busy);               {server flag}

   muxTimeRec = RECORD
        instopTime  : REAL;               {each MUX records the time the server}
        outstopTime : REAL;               {stops serving, for resynchronization}
   END RECORD;

   entry = RECORD                         {address book entry}
        siteNo      : INTEGER;           {number of this site}
        startlabel  : CHAR;              {for the selected destination}
        destination : INTEGER;           {destination site}
   END RECORD;
```

```
  calls = RECORD                    {record for PBX}
        call      : phonecallObj;   {actual phone call}
        callstatus : Status;        {free or engaged}
  END RECORD;

  policingrec = RECORD                          {leaky bucket policing function}
        activesource      : Status;             {is this source active?}
        counter           : INTEGER;            {number of cells currently transmitted}
        threshold         : INTEGER;            {max number of cells before tagging}
        slotdecrement     : REAL;               {wait time for decrement function}
        decrement         : INTEGER;            {amount to decrement by}
        decrementflag     : BOOLEAN;            {decrement in progress}
        TECflag           : BOOLEAN;            {tail end clipping function activated}
        qhead, qtail      : cell;
        qlen              : INTEGER;            {queuing stuff for buffered LB}
        busyflag          : flagType;           {serving flag for buffered LB}
  END RECORD;

{-------}
{OBJECTS}
{-------}

{User site declarations}

  SourceObj = OBJECT; {base type for all sources}
        source                  : sourceType;   {source of cells - video, speech}
        sitenumber              : INTEGER;      {associated user site No}
        meaninterarrivaltime : REAL;            {in slots}

        ASK METHOD Initalise(IN lam : REAL; IN s : sourceType; IN numberofthissite : INTEGER);

  END OBJECT;  {sourceObj}
  {--------------------}

  videosourceObj = OBJECT(SourceObj);
        meanburstSize   : REAL;         {calc mean burst size based on peak (in slots)}
        peakrate        : REAL;         {input at run time}
        calculatedmean  : REAL;         {theoretical mean rate for 8 state model}
        meanvideorate   : SREAL;        {Stat variable - remove after testing}
        numberofslices  : SINTEGER;     {counts the number of slices}

        ASK METHOD Createvideosources(IN videosource : INTEGER; IN rate : REAL);
        TELL METHOD Generate(IN ID : INTEGER);

  END OBJECT;  {videosourceObj}
  {--------------------}

datasourceObj = OBJECT(SourceObj);
        startTime       : REAL;         {time the source started up}
        rate            : REAL;         {data rate in bits/sec}
        duration        : REAL;         {time duration for this source in secs}
        activesources   : INTEGER;      {number of active data sources}

        TELL METHOD Generate(IN maxNodatafiles : INTEGER);
        ASK METHOD Dataparams(IN source : sourceType);
        TELL METHOD Sendfile(IN rate : REAL; IN duration : REAL; IN source : sourceType;
```

```
                                                                IN dataID : INTEGER);
  END OBJECT;  {datasourceObj}
  {---------------------}

  speechsourceObj = OBJECT(SourceObj);

        TELL METHOD Generate;

  END OBJECT; {speechsourceObj}
  {---------------------}

  phonecallObj = OBJECT;
        silence, talkspurt    : REAL;        {duration of talksurts and silences}
        source                : sourceType;  {this variable will always be speech}

        TELL METHOD Makecall(IN linenumber : INTEGER; IN sitenumber : INTEGER);

END OBJECT; {phonecallObj}
  {---------------------}

  muxObj = OBJECT;
        sitenumber        : INTEGER;
        attachedswitch    : INTEGER;
        port              : INTEGER;
        PSflag            : BOOLEAN;                    {RT flag - Priority Service flag}
        dropLPcells       : BOOLEAN;                    {RT flag - drop LP speech cells flag}
        throttledataflag  : BOOLEAN;                    {nonRT flag - slow data cells from LB}
        blockdataflag     : BOOLEAN;                    {onoRT falg - to stop data cells}
        {stopdataflag     : BOOLEAN;}                   {nonRT flag - stop data cells from LB}
        servingflag       : ARRAY INTEGER OF flagType;  {idle or busy}
        blockeddatacell       : SINTEGER;  {count blocked data cells (from T1)}
        droppedLPvideocell    : SINTEGER;  {count LP video cells dropped by MUX}
        droppedLPspeechcell   : SINTEGER;  {count LP speech cells dropped by MUX}
        droppedLPdatacell     : SINTEGER;  {count LP data cells dropped by MUX}
        qhead             : ARRAY INTEGER OF cell;      {queue stuff}
        qtail             : ARRAY INTEGER OF cell;
        qlen              : ARRAY INTEGER OF INTEGER;   {actual counter for queue}
        resetqlen         : ARRAY INTEGER OF TSINTEGER; {Stat counter for queue reset by stats}

        ASK METHOD Initalise(IN switchNo : INTEGER; IN useport : INTEGER;
                                                        IN sitenumber : INTEGER);
        ASK METHOD AddtomuxQ(IN ATMcell : cell; IN queueNo : INTEGER);   {put on queue}
        TELL METHOD ServemuxQ(IN queue : INTEGER);                       {incoming Q server}
        TELL METHOD CyclicServer(IN queue : INTEGER);                    {outgoing Q server}

END OBJECT; {muxObj}
{---------------------}

usersiteObj = OBJECT;
        Video             : videosourceObj;     {source objects}
        Speech            : speechsourceObj;
        Data              : datasourceObj;
        mux               : muxObj;             {multiplexer object}
        PBX           : ARRAY INTEGER OF calls; {telephone exchange}
        sitenumber        : INTEGER;            {number of this site}
        numberofcalls     : SINTEGER;           {Stats for number of phone calls}
```

```
maxallowedcalls     : INTEGER;          {max number of phone calls allowed}
maxvideosources     : INTEGER;          {max number of video sources allowed}
maxdata             : INTEGER;          {max data files}
totalcapacity       : REAL;             {total link capacity in use}
policing            : ARRAY sourceType, INTEGER OF policingrec;
clippedslice        : INTEGER;          {count number of damaged slices}

{variables to analyse LB tagging function}
taggedvideo, taggingspeech, taggingdata : INTEGER;

speechTotal, videoTotal             : INTEGER;      {generated}
data1Total, data2Total, data3Total  : INTEGER;      {cells}
data4Total, data5Total, data6Total  : INTEGER;

ASK METHOD Initalise(IN numberofthissite : INTEGER);
ASK METHOD InitaliseLeakybucket(IN ID : INTEGER; IN sourcetype : sourceType;
                                IN meanburst : REAL; IN rate : REAL);
ASK METHOD AddtoLBbuffer(IN ATMcell : cell; IN ID : INTEGER);   {adds data to the LB queue}
TELL METHOD Leakybucket(IN ID : INTEGER; IN source : sourceType);   {queuing LB}
ASK METHOD VirtualLeakybucket(IN ATMcell : cell);                   {RT LB}
ASK METHOD PasstoMux(IN source : sourceType; IN ATMcell : cell);
TELL METHOD Decrementfunction(IN ID : INTEGER; IN sourcetype : sourceType);
ASK METHOD Increment(IN source : sourceType);                       {count cells}
ASK METHOD IncBlocks(IN block : INTEGER; IN source : sourceType);
ASK METHOD PBXfreeline(IN linenumber : INTEGER; IN callTimestamp : REAL);
                                                                    {release PBX line}
ASK METHOD AddtoPBX(IN linenumber : INTEGER);                       {add new call to PBX}
ASK METHOD CAC(IN newrate : REAL; IN source : sourceType; IN ID : INTEGER;
                        OUT accepted : BOOLEAN);    {reserve bandwidth – is call allowed?}
ASK METHOD ReleaseResources(IN rate : REAL; IN source : sourceType; IN ID : INTEGER);
                                                    {update current capacity and release LB}
ASK METHOD Receivecell(IN receivedcell : cell);     {count cells received}
ASK METHOD SiteStats;                               {output the statistics}

  END OBJECT; {usersiteObj}
  {---------------------}

EXPORTTYPE
  usersitemanagerObj = OBJECT; FORWARD;     {resolving cyclic dependencies}
  TYPE
  usersitemanagerObj = OBJECT;

      ASK METHOD Initalise(IN Numofsites : INTEGER; IN randomseed : INTEGER);
      ASK METHOD Accessusersite(IN ATMcell : cell; IN siteNo : INTEGER);
      TELL METHOD Monitor(IN runlen : REAL; IN interval : REAL);
      ASK METHOD UsersiteStats;

  END OBJECT;  {usersite manager object}
{******end of user site declarations***************}

{end of type declarations}
```

```
VAR {global variables}
        usersitemanager     : usersitemanagerObj;                  {user site manager object}
        usersite            : ARRAY INTEGER OF usersiteObj;        {array of user sites}
        rand                : RandomObj;                           {random object for all usersites}
        runlength           : REAL;                                {length of simulation run}
        numberofsites       : INTEGER;                             {number of user sites}
        numberofswitches    : REAL;                                {number of switches in N/W}
        endrun              : BOOLEAN;                             {flag for end of run}
        realtimeQ           : INTEGER;                             {MUX realtime outgoing queue = 3}
        outgoingMUXQ        : INTEGER;                             {MUX outgoing queue = 2}
        incomingMUXQ        : INTEGER;                             {MUX incoming queue = 1}
        RT1, RT2            : INTEGER;                             {thresholds for RT traffic}
        T1, T2              : INTEGER;                             {thresholds for non-RT traffic}
        maxentry            : INTEGER;   {number of entries in the address book}
        addressbook         : ARRAY INTEGER OF entry;              {usersite address book}
        muxTime             : ARRAY INTEGER OF muxTimeRec;         {stop times for MUX's}
        tdRTout             : ARRAY INTEGER OF SREAL;              {delay for out MUX queue}
        tdRTin              : ARRAY INTEGER OF SREAL;              {delay for in MUX queue}
        tdnonRTout          : ARRAY INTEGER OF SREAL;              {delay for out MUX queue}
        tdnonRTin           : ARRAY INTEGER OF SREAL;              {delay for in MUX queue}
        tdvideo             : ARRAY INTEGER OF SREAL;              {delay for video cells}
        tdspeech            : ARRAY INTEGER OF SREAL;              {delay for speech cells}
        tddata              : ARRAY INTEGER OF SREAL;              {time delay for data cells}
        burstStats          : ARRAY INTEGER OF SINTEGER;           {burst sizes}
        bursttimeStats      : ARRAY INTEGER OF SREAL;              {delays for last cell}
        callStats           : ARRAY INTEGER OF SREAL;              {stats for calls}
        utaccesslink        : ARRAY INTEGER OF TSINTEGER;          {user link utilization}

{cells received by site 3 ONLY}
{site 1}
        received1speech, received1video    : INTEGER;       {received cells from 1 by 3}
        received1data1, received1data2, received1data3,
        received1data4, received1data5, received1data6 : INTEGER;

{site 2}
        received2speech, received2video : INTEGER;          {received cells from 2 by 3}
        received2data1, received2data2, received2data3,
        received2data4, received2data5, received2data6 : INTEGER;

{site 4}
        received4speech, received4video : INTEGER;          {received cells from 4 by 3}
        received4data1, received4data2, received4data3,
        received4data4, received4data5, received4data6 : INTEGER;

{cells received by all sites to tally with total cells generated}
{site 1}
        receive1speech, receive1video : INTEGER;            {cells from 1 - all sites}
        receive1data1, receive1data2, receive1data3,
        receive1data4, receive1data5, receive1data6 : INTEGER;

{site 2}
        receive2speech, receive2video : INTEGER;            {cells from 2 - all sites}
        receive2data1, receive2data2, receive2data3,
        receive2data4, receive2data5, receive2data6 : INTEGER;
```

```
{site 4}
        receive4speech, receive4video : INTEGER;  {cells from 4 - all sites}
        receive4data1, receive4data2, receive4data3,
        receive4data4, receive4data5, receive4data6 : INTEGER;

{-------------------------}

PROCEDURE PDdelay(IN rate : REAL; IN source : sourceType; OUT PD : REAL);
PROCEDURE idleslots(IN stoptime : REAL; OUT idleslot : INTEGER);
PROCEDURE findfreelinenumber(IN sitenumber : INTEGER; IN maxallowed : INTEGER;
                                                          OUT lineNo : INTEGER);
PROCEDURE findVCIlabel(IN sitenumber : INTEGER; IN calledsite : INTEGER; OUT VCIlabel : CHAR);
PROCEDURE finddestination(IN thissite : INTEGER; OUT calledsite : INTEGER);

END MODULE. {Definition module for usersite}
```

```
IMPLEMENTATION MODULE usersite;              {super LB version 2x burst}

{usersite with dual queues and cyclic server - adding TEC function}
{priority service for RT queue (at 2 levels) and limited non-RT queue}
{expanded ATM network with new routing tables etc 5 switches}
{time is in slots related to the network duration of a slot - 2.7 microsecs}
{access to the link is 45Mb/s, so service times are increased accordingly}
{adding virtual leaky bucket for all sources - tagged cells are passed to}
{network - MUX deletes tagged cells if queues are too long - burst lenght used}

FROM SimMod  IMPORT SimTime;
FROM RandMod IMPORT RandomObj, FetchSeed;
FROM StatMod IMPORT StatObj, RStatObj, IStatObj, ITimedStatObj, TSINTEGER, SINTEGER,
                                                          RTimedStatObj, TSREAL, SREAL;
FROM MathMod IMPORT LOG10;
FROM types IMPORT ALL sourceType, ALL cellType, ALL cell;
FROM atmnw IMPORT ATMnetwork;

{----Procedure Declarations-------}
{----------------------------}

PROCEDURE idleslots(IN stoptime : REAL; OUT idleslot : INTEGER);
{calculate idle slots}
   VAR idlecells : REAL;

   BEGIN
         idlecells := SimTime() - stoptime;
         idleslot := ROUND(idlecells + 0.5);
END PROCEDURE; {idleslots}
{----------------------------}

PROCEDURE PDdelay(IN rate : REAL; IN source : sourceType; OUT PD : REAL);
{convert the current rate (in b/s) into slot intervals}

   VAR cellrate          : REAL;          {local variables for calulations}
         payload         : REAL;

   BEGIN
         IF (source = video) {then} payload := videocellpayload;
         ELSE payload := cellpayload; END IF;
         cellrate := rate / payload;
         PD := capacityinslots / cellrate;  {use slotted time}

END PROCEDURE; {PDdelay}
{----------------------------}

PROCEDURE findlinenumber(IN sitenumber : INTEGER; IN maxallowed : INTEGER;
                         OUT lineNo : INTEGER);          {find a free line number at the user site}

VAR  found  : BOOLEAN;          {stop search when TRUE}

  BEGIN          {usersitemanager.}
         found := FALSE;
         lineNo := 1;
         WHILE (lineNo <= maxallowed) AND (found = FALSE)
           IF (usersite[sitenumber].PBX[lineNo].callstatus = free) {then}
```

```
                found := TRUE;
            ELSE
                INC(lineNo);
            END IF;

        END WHILE;
        IF found = FALSE {then} lineNo := 0; END IF; {trap for errors}
END PROCEDURE; {findlinenumber}
{------------------------------}

PROCEDURE findVCIlabel(IN sitenumber : INTEGER; IN calledsite : INTEGER;
                                                    OUT VCIlabel : CHAR);
{look up the usersite address book and get the VCI label for the destination}

   VAR endsearch : BOOLEAN;
       count    : INTEGER;

   BEGIN
       endsearch := FALSE;
       count := 1;
       WHILE (endsearch = FALSE) AND (count <= maxentry);
        IF (sitenumber = addressbook[count].siteNo) AND
                        (calledsite = addressbook[count].destination); {then}
            endsearch := TRUE;
            VCIlabel := addressbook[count].startlabel;
         ELSE
            INC(count);
         END IF;
        END WHILE; {search for VCI label}

END PROCEDURE;  {findVCIlabel}
{------------------------------}

PROCEDURE finddestination(IN thissite : INTEGER; OUT calledsite : INTEGER);
{sites 1 and 4 can send to all other sites (2, 3 or 4)}
{5 does not send any and 2 only sends to 3}
{50% chance of sending to 3 and 25% to either 5 or 2 from sites 1 and 4}

VAR temp : INTEGER;

BEGIN
        {fixed destinations for testing - to maintain utilization}
        IF ((thissite = 1) OR (thissite = 4) ) {then}
          temp := ASK rand UniformInt(1, 4);
          IF   (temp = 1) OR (temp = 4) {then} calledsite := 3;
          ELSIF (temp = 2) {then} calledsite := 5;
          ELSIF (temp = 3) {then} calledsite := 2;
          END IF;
        ELSIF (thissite = 2) {then}
          calledsite := 3;
        END IF;

END PROCEDURE; {sitetocall}

{********Code for Objects*****************************************************}
{---------code for sources-------------}
```

```
OBJECT SourceObj;

   ASK METHOD Initalise(IN lam : REAL; IN s : sourceType; IN numberofthissite : INTEGER);

   BEGIN
         sitenumber := numberofthissite;
         source := s;
         meaninterarrivaltime := lam * recipofslot; {convert to slots}
   END METHOD; {Initalise}
   {----------------------}
END OBJECT; {SourceObj}

{------------end Object--------------------}

OBJECT  videosourceObj;
{the source object generates the cells and places each cell in the queue}

    {----------------------}
   ASK METHOD Createvideosources(IN videosources : INTEGER; IN rate : REAL);
   {calculates mean rate based on No of states and peak rate and mean burst size from calculated mean}

   VAR count   : INTEGER;
         states   : REAL;                                  {temporary variable}

   BEGIN
         peakrate := rate;                                 {initalise peak rate for video}
         states := 8.0;                                    {8 states}

         FOR count := 1 TO TRUNC(states)                   {calculate mean rate for video source}
               rate := rate + (peakrate / FLOAT(count));
         END FOR;
         calculatedmean := rate / states;                  {calculate the mean rate}
         meanburstSize := (calculatedmean * videorefresh) / cellpayload; {calculate and initalise the burst size}

         OUTPUT("site No ", sitenumber, " No video sources ", videosources, " peak rate ", peakrate, "
                              burstsize ", meanburstSize, " calc meanrate ", calculatedmean);
         OUTPUT("video currently uses peak rate and 2 x burst to initalise LB");
         FOR count := 1 TO videosources
               Generate(count);                            {activate video sources}
         END FOR;

END METHOD; {Createvideosources}
         {----------------------}

TELL METHOD Generate(IN ID : INTEGER);
{generates a burst of ATMcells, with a mean inter-arrival time between bursts - no gaps between bursts}

VAR wait                 : REAL;      {delay between bursts}
         burst              : INTEGER;   {burst related variables}
         burstsize          : REAL;      {exponential burst size}
         bursttimestamp     : REAL;      {start of a burst time stamp}
         videopacketization : REAL;      {delay for cells in a burst}
         calledsite         : INTEGER;   {destination site}
         VCIlabel           : CHAR;      {VCI label to use for destination}
```

```
        burstiness          : INTEGER;   {burstiness factor}
        currentrate         : REAL;      {current rate based on burstiness}
        cellnumber          : INTEGER;
        ATMcell             : cell;      {video cell}
        accepted            : BOOLEAN;   {accepted by CAC function}
        TScounter           : INTEGER;   {counter for transport stream}
        TSsize              : INTEGER;   {size of current TS}

BEGIN
        accepted := FALSE;                      {accepted by LB}
        currentrate := calculatedmean * (1.0 - LOG10(calculatedmean / peakrate));
        {using calculated mean for CAC and LB initalisation for the moment}
        ASK usersite[sitenumber] CAC(currentrate, source, ID, accepted);

        IF (accepted = TRUE) {then start generating}

        {start up policing function}
        ASK usersite [sitenumber] TO InitaliseLeakybucket(ID, source,
                                        (2.0 * meanburstSize), peakrate); {initalise LB}

        IF (ID = 1) {then set destination to 3}
                calledsite := 3;
        ELSE
                finddestination(sitenumber, calledsite);  {find a site to call}
        END IF;

        findVCIlabel(sitenumber, calledsite, VCIlabel); {get VCI label}
        PDdelay(peakrate, source, videopacketization);      {calc packetisation delay}
        currentrate := peakrate;                            {output 1st burst at peak rate}

        {****begin to output bursts of video cells*****}
        LOOP
        IF (endrun = TRUE)      {STOP SIMULATION TIME}
                EXIT;
        END IF;

        meanvideorate := currentrate;    {update stats for video rate}
        TScounter := 0; {initialise counter for transport stream}
        {generate next burst}
        burstsize := ASK rand Exponential(meanburstSize);
        burst := ROUND(burstsize + 0.5);                    {convert to integer}
        ASK usersite[sitenumber] TO IncBlocks(burst, source);   {update stats}
        bursttimestamp := SimTime();                        {time stamp start of burst}

        FOR cellnumber := 1 TO burst                        {loop to generate bursts of cells}
                                                            {divide burst into slices}
          IF (TSsize = TScounter)                           {then reset counter and get next TS size}
                TScounter := 0;
                TSsize := ASK rand UniformInt(5, 100);
          END IF;
          INC(TScounter);                                   {increment the counter for TS pkts}
          WAIT DURATION  videopacketization; END WAIT;      {packetization delay}
          NEW(ATMcell);                                     {generate next cell}
          IF (TScounter = 1)                                {then reset TEC function - start of burst}
                ATMcell.GFCfield := 1;
                INC(numberofslices);                        {count slices}
```

```
    ELSE
          ATMcell.GFCfield := 0;
    END IF;

        {Initalise ATM cell with a timestamp, source}
    ATMcell.cellstartTime := SimTime();                    {cell time stamp}
    ATMcell.source        := source;                       {video cell}
    ATMcell.typeofcell    := I;                            {information cell}
    ATMcell.VCIlabel      := VCIlabel;                     {VCI label to use cell}
    ATMcell.origin        := sitenumber;                   {this site number}
    ATMcell.CLP           := 0;                            {high priority cell}
    ATMcell.VCid          := ID;                           {video ID number}
    ATMcell.cellcount     := cellnumber;                   {temp field for testing}

    IF (cellnumber = burst) {then}
        ATMcell.msgstartTime := bursttimestamp;  {message time stamp the last cell in burst only}
    ELSE
        ATMcell.msgstartTime := 0.0;
    END IF;

    ASK usersite[sitenumber] TO Increment(source);          {update cell stats}
    ASK usersite[sitenumber] TO VirtualLeakybucket(ATMcell);      {pass to LB}

  END FOR;      {end of burst}

  {get next packetization delay}
  burstiness := ASK rand UniformInt(1, 8);                          {burstiness of 1 - 8}
  currentrate := peakrate / FLOAT(burstiness);                      {calc current rate}
  PDdelay(currentrate, source, videopacketization);                 {call procedure to calc PD}

END LOOP; {output video traffic for duration of simulation}
END IF;
          ASK usersite[sitenumber] TO ReleaseResources(calculatedmean, source, ID);
          {send disconnect cell}
          OUTPUT("end of video model ID ", ID, " site ", sitenumber);

    END METHOD; {Generate video}

{-------------------------}

END OBJECT;  {video source}

{------------end Object video-----------------------------------------------}

{----------------SPEECH-------------------}

OBJECT  speechsourceObj;
{the source object generates phone call objects - max 40 allowed}
{Phone calls are generated with an exponentially distributed wait between calls}
{Each new phone call is given a call length and a timestamp.}
{All times are converted to slots when used for "waits", otherwise they are}
{seconds. }

  {  ASK METHOD Initalize; uses method from source}
    TELL METHOD Generate;
```

```
VAR
        timetonextcall    : REAL;          {actual time between phone calls}
        phonecall         : REAL;          {length of a phone call}
        call              : phonecallObj;  {a phone call}
        calltimestamp     : REAL;          {time call starts}
        linenumber        : INTEGER;       {free line number for the call}
        callstatus        : Status;        {line status – engaged or free}
        maxcallsallowed   : INTEGER;       {max number of phone calls allowed}
        accepted          : BOOLEAN;       {is the call accepted?}

BEGIN
        accepted := FALSE;
        maxcallsallowed := ASK usersite[sitenumber] maxallowedcalls;
        OUTPUT("SPEECH Site No ", sitenumber, " maxcalls ", maxcallsallowed);

        LOOP    {to create phone calls}
         IF (endrun = TRUE) TERMINATE;  END IF;
         timetonextcall := ASK rand Exponential(meaninterarrivaltime); {generate time between calls in slots}
         WAIT DURATION timetonextcall; END WAIT;
         IF (ASK usersite[sitenumber] numberofcalls < maxcallsallowed)        {is call allowed? }
                                                {find a free line number identifier for this phone call}
          findlinenumber(sitenumber, maxcallsallowed, linenumber);
          IF (linenumber = 0) {then error finding line number}
                OUTPUT("this line is engaged error in linenumber");          {error trap}
          ELSE
            ASK usersite[sitenumber] TO CAC((speechrate * 0.5), source, linenumber, accepted);
            IF (accepted = TRUE)
              ASK usersite[sitenumber] TO AddtoPBX(linenumber);    {add to PBX and Inc number of calls}
              TELL usersite[sitenumber].PBX[linenumber].call TO Makecall(linenumber, sitenumber);
            END IF;
          END IF;  {linenumber = 0}
         END IF;  {number of calls < max}
        END LOOP; {next phone call}
        OUTPUT("end of simulation - speech model");

   END METHOD; {Generate speech}
   {------------------------}

END OBJECT;  {source for speech}

{--------------------end Object speech-------------------------------------}
OBJECT phonecallObj;
{a new phonecall object is created every time a call is made (max 40 allowed)}
{a search is made for a vacant line number identifier and the total number of}
{phone calls is incremented. The speaker then alternates between talkspurts }
{and silences for the duration of the call. The duration of the call is }
{determined exponentially before the call begins, but the actual call length}
{is usually slightly longer, due to the program loops used to generate the}
{talkspurts and silences. Talkspurt generates a burst of speech. Cells in a}
{burst have an intercell spacing equal to a maximum packetization delay of}
{6.0 milli seconds which is the time to transmit a cell at a constant rate of 64Kbit/s.}

   {------------------------}
   TELL METHOD Makecall(IN linenumber  : INTEGER; IN sitenumber : INTEGER);
```

```
{call duration is in slots, and is generated by the call itself}
{linenumber is the number of the calling phone line, from the sitenumber}

VAR endofcall        : REAL;       {time to stop simulation}
    count, newcount  : INTEGER;    {temporary counters for loops}
    numberofcells    : INTEGER;    {number of cells to be output}
    cells            : REAL;       {variable used to calculate number of cells}
    ATMcell          : cell;       {instance of an ATM cell}
    phonenumber      : INTEGER;    {the number dialled}
    calledsite       : INTEGER;    {destination usersite}
    VCIlabel         : CHAR;       {start VCI label for this destination}
    source           : sourceType; {speech cell identifier}
    calllength       : REAL;       {time the call lasts}
    callTimestamp    : REAL;       {time stamp that the call begins}
    otherPBXmax      : INTEGER;    {max calls allowed at the other site}
    realQ            : INTEGER;    {number of the RT queue to use at MUX}

BEGIN
 realQ := 3;                                  {output to realtime Q at MUX}
 source := speech;
 calllength := ASK rand Exponential(phonecallduration);   {get call duration - in slots}
 endofcall  := SimTime() + calllength;                    {calculate time to stop}

 IF (linenumber = 1)                                      {then set destination to site 3}
     calledsite := 3;
 ELSE
  finddestination(sitenumber, calledsite);                {procedure to find called site}
 END IF;
 cells := 0.0;                                            {cells is blank to make the param list correct}
 ASK usersite[sitenumber] TO InitaliseLeakybucket(linenumber, source, cells, speechrate);
 findVCIlabel(sitenumber, calledsite, VCIlabel);          {find VCI label to use}

{begin the call with a silence - wait for someone to answer}
silence := ASK rand Exponential(Silence);                 {generate slience in slots}

WAIT DURATION silence;  END WAIT;

WHILE (SimTime() < endofcall)          {continue generating talkspurt and silence periods until call ends}
      IF (endrun = TRUE) EXIT; END IF;
      {generate length of talkspurt and convert to cells}
       talkspurt := ASK rand Exponential(Talkspurt); {in seconds}
       cells := (speechrate * talkspurt) / speechpayload;
       numberofcells := ROUND((cells + 0.5));

       FOR count := 1 TO (numberofcells) BY 2;   {put pairs of speech cells into queue back-to-back}
            WAIT DURATION (speechpacketization * 2.0); END WAIT;
            FOR newcount := 1 TO 2
               NEW(ATMcell);
               ATMcell.source        := source;              {speech source}
               ATMcell.cellstartTime := SimTime();           {time stamp cell}
               ATMcell.VCIlabel      := VCIlabel;            {VCI label for cell}
               ATMcell.origin        := sitenumber;          {number of this site}
               ATMcell.msgstartTime := 0.0;                  {not last cell}
               ATMcell.VCid          := linenumber;
               ATMcell.cellcount     := count;               {temp field for testing}
               IF (newcount = 1) {then}
```

```
                        WAIT DURATION aslot; END WAIT;      {slot space between each cell}
                        ATMcell.CLP := 0;                   {high priority cell}
                    ELSE
                        ATMcell.CLP := 1;                   {low priority cell}
                    END IF;
                    ASK usersite[sitenumber] TO Increment(source); {increment speech cells generated}
                    ASK usersite[sitenumber] TO VirtualLeakybucket(ATMcell);
                END FOR; {outputing 2 cells back-to-back}
            IF (endrun = TRUE) TERMINATE; END IF; {stop simulation}
        END FOR;

        silence := ASK rand Exponential(Silence);{in slots}
        WAIT DURATION silence; END WAIT;
    END WHILE; {end of call}

{send disconnect request}
    ASK usersite[sitenumber] TO ReleaseResources((speechrate * 0.5), source, linenumber);
                                {release capacity and LB decrement function}
    ASK usersite[sitenumber] PBXfreeline(linenumber, callTimestamp);
                                {releases line, decrements No of calls and also disposes of this call}
  {end of call}
  END METHOD; {makecall}
  {---------------------}
END OBJECT; {phonecallObj}

{------------end Object phone call----------------------------------------}

{--------------DATA----------------------}

OBJECT datasourceObj;
{the source object generates the cells and places each cell in the queue}

  {---------------------}
TELL METHOD Generate (IN maxNodatafiles : INTEGER);
{generates a data model type (currently 4 types available), and then}
{given max. and min. rates and durations, a bit rate and a time duration}
{are obtained. The packetization delays and the number of ATM cells are}
{calculated.}

VAR wait        : REAL;         {delay between files}
    result      : INTEGER;      {result of dice throw for file type}
    found       : BOOLEAN;      {search variable}
    dataID      : INTEGER;      {index for array}
    sourcetype  : sourceType;   {index for policing array - data}
    accepted    : BOOLEAN;      {is the call allowed}
    blank       : REAL;         {to fill LB param list - CBR not used}

BEGIN
    startTime := SimTime();
    OUTPUT("data model at site number", sitenumber, " Max ", maxNodatafiles);
    sourcetype := data;
    IF (maxNodatafiles > 5)                 {then load up 5 sources}
      {loop to send one data type of each kind to start}
      FOR dataID := 1 TO 5
        IF   (dataID = 1) {then} source := data1;
        ELSIF (dataID = 2) {then} source := data2;
```

```
        ELSIF (dataID = 3) {then} source := data3;
        ELSIF (dataID = 4) {then} source := data4;
        ELSIF (dataID = 5) {then} source := data5;        END IF;
        Dataparams(source);
        ASK usersite[sitenumber] TO CAC(rate, sourcetype, dataID, accepted); {request bandwidth}
        IF (accepted = TRUE) {then}
            ASK usersite[sitenumber] TO InitaliseLeakybucket(dataID, sourcetype, blank, rate);
            Sendfile(rate, duration, source, dataID); {output file to mux}
            INC(activesources);
        END IF;  {if accepted is FALSE, then the capacity was not updated}
      END FOR;
    END IF; {maxNodatafiles > 5}

LOOP
    found    := FALSE;                                    {reset all params}
    dataID   := 1;                                        {and search for a free line}
    accepted := FALSE;
    IF (endrun = TRUE)                                    {STOP SIMULATION TIME}
        EXIT
    END IF;
    wait := ASK rand Exponential(meaninterarrivaltime);   {generate next inter-arrival time}
    WAIT DURATION wait; END WAIT;                         {wait for next data file}

    IF (activesources <  maxNodatafiles)                  {then allow data source to begin}

    WHILE (dataID <= maxNodatafiles) AND (found = FALSE)

      IF (usersite[sitenumber].policing[sourcetype, dataID].activesource = free)  {then}
        found := TRUE;
      ELSE
        INC(dataID);                                      {not found, so check next line}
      END IF;
    END WHILE;    {found a free line}

    IF (found = TRUE)                                     {then choose a data type, get params & send file}
      result := ASK rand UniformInt(1, 5);                {generate random integer}
      IF result = 1                                       {then source is voiceband data}
          source := data1;
      ELSIF result = 2                                    {then source is videotext/teletex}
          source := data2;
      ELSIF result = 3                                    {then source is telemetry}
          source := data3;
      ELSIF result = 4                                    {then source is facsimile}
          source := data4;
      ELSIF result = 5                                    {then source is transaction time sharing}
          source := data5;
      END IF;                                             {result}

      Dataparams(source);                                 {get rate and duration for data type}

     {CAC function sets active source to engaged}
      ASK usersite[sitenumber] TO CAC(rate, sourcetype, dataID, accepted);
      IF (accepted = TRUE)      {then start LB, line becomes engaged, send the file
                                 and increment number of sources active}
        ASK usersite[sitenumber] TO InitaliseLeakybucket(dataID, sourcetype, blank, rate);
```

```
            Sendfile(rate, duration, source, dataID);        {output file to mux}
            INC(activesources);                              {only INC if call is allowed}

          END IF;                                            { accepted = TRUE}
        END IF;                                              {found = TRUE}
      END IF;
 END LOOP;
 OUTPUT("end of simulation - data model");
END METHOD; {Generate data}

{------------------------}

ASK METHOD Dataparams(IN source : sourceType);
{calculate rate in b/s and time duration in seconds.}

VAR
      minrate, maxrate          : REAL;      {range of bit rates}
      minduration, maxduration : REAL;       {range of time durations (seconds)}
      y1, y2, y                 : REAL;      {used for calculations of durations}

BEGIN
{simple data models}
IF source = data1                            {then set global variables for voice band data model}
      minrate  := 300.0;                     {b/s}
      maxrate  := 30.0E3;
      minduration  := 55.0;                  {seconds}
      maxduration  := 3000.0;

ELSIF source = data2                         {then set global variables for videotex/teletex model}
      minrate  := 600.0;                     {b/s}
      maxrate  := 90.0E3;
      minduration  := 600.0;                 {seconds}
      maxduration  := 2000.0;

ELSIF source = data3                         {then set global variables for telemetry model}
      minrate  := 2.0;                       {b/s}
      maxrate  := 200.0;
      minduration := 1.0;                    {seconds}
      maxduration := 60.0;

{complex data models}
{data4 and data5 require the rate for calculation of min & max times}
ELSIF source = data4                         {then set global variables for facsimile}
      minrate := 3.0E3;                      {b/s}
      maxrate := 3.0E6;                      {Mb/s}
      rate := ASK rand UniformReal(minrate, maxrate);    {find bit rate for the complex model}

      {calculate max and min durations for the bit rate selected}
      {maxduration}
      y1 := 1000.0;                          {seconds}
      y2 := 100.0;
      y  := (y1 - y2) * ((rate - minrate) / (maxrate - minrate));
      maxduration := y1 - y;

      {minduration}
      y1 := 200.0;                           {seconds}
```

```
        y2 :=  3.0;
        y  := (y1 - y2) * ((maxrate - rate) / (maxrate - minrate));
        minduration := y2 + y;

    ELSIF source = data5                          {then set global variables for trans timesharing}
        minrate := 60.0;                          {b/s}
        maxrate := 6.0E3;                         {Kb/s}

        {find bit rate for the complex model}
        rate := ASK rand UniformReal(minrate, maxrate);

        {calculate max and min durations for the bit rate selected}

        IF rate > 1000.0
          minduration := 20.0;
          maxduration := 3600.0;
        ELSE
        {maxduration}
          y1 := 3600.0;                           {seconds}
          y2 := 80.0;
          y  := (y1 - y2) * ((maxrate - rate) / (maxrate - minrate));
          maxduration := y1 - y;
          minduration := 20.0;
        END IF;

END IF;

    IF (source = data1) OR (source = data2) OR (source = data3) {then}
      rate := ASK rand UniformReal(minrate, maxrate);
    END IF;    {find bit rate for simple data models}

    duration := ASK rand UniformReal(minduration, maxduration);
    END METHOD; {Data}

    {----------------------}

    TELL METHOD Sendfile (IN rate : REAL; IN duration : REAL; IN source : sourceType;
                                                                  IN dataID : INTEGER);
    VAR cellcounter       : INTEGER;          {total number of cells to transmit}
        msgsize           : REAL;             {size of file to send}
        msgtimestamp      : REAL;             {start of a msg time stamp}
        cellnumber        : INTEGER;
        temp              : REAL;             {variables used to calculate the}
        temp1             : INTEGER;          {number of cells for this source}
        datapacketization : REAL;             {packetization delay for this data source}
        ATMcell           : cell;
        destination       : INTEGER;          {destination user site number}
        VCIlabel          : CHAR;             {VCI label to use}

    BEGIN
{set up connection}
        msgtimestamp := SimTime();            {time stamp data file}
        msgsize := rate * duration;           {calculate message size}

        {convert message into a whole number of cells - rounding up if nec}
```

```
temp := msgsize / cellpayload;
temp1 := TRUNC(temp);

IF (temp - FLOAT(temp1) = 0.0)  {then}
   cellcounter := temp1;                              {no rounding required - filled cells}
ELSE
   cellcounter := ROUND(temp + 0.5);                  {round up - part filled cell}
END IF;

{convert duration into slots and calculate packetization delay}
datapacketization := (duration * recipofslot) / (FLOAT(cellcounter));

IF (dataID = 1)                                       {then set destination to site 3}
  destination := 3;
ELSE
  finddestination(sitenumber, destination);           {find site for data}
END IF;

findVCIlabel(sitenumber, destination, VCIlabel);

  {loop to output cells to queue}
  FOR cellnumber := 1 TO cellcounter
    WAIT DURATION datapacketization; END WAIT;        {packetization delay}
    NEW(ATMcell);                                     {generate next cell}

    {Initalise ATM cell with a timestamp, source, etc}
    ATMcell.cellstartTime := SimTime();
    ATMcell.source        := source;
    ATMcell.typeofcell    := I;
    ATMcell.origin        := sitenumber;
    ATMcell.VCIlabel      := VCIlabel;
    ATMcell.CLP           := 0;                       {high priority cell}
    ATMcell.VCid          := dataID;
    ATMcell.cellcount     := cellnumber;              {temp field for testing}
    ATMcell.dest := destination;
    IF cellnumber = cellcounter;                      {then last cell}
        ATMcell.msgstartTime := msgtimestamp;         {time stamp last cell}
    ELSE                                              {not last cell}
        ATMcell.msgstartTime := 0.0;
    END IF;
    ASK usersite[sitenumber] TO Increment(source);
    ASK usersite[sitenumber] TO AddtoLBbuffer(ATMcell, dataID);
    IF (endrun = TRUE) TERMINATE; END IF; {stop simulation}
  END FOR;      {end of file}

  {release all resources for this call}
  source := data;
  ASK usersite[sitenumber] TO ReleaseResources(rate, source, dataID);
  DEC(activesources);                                 {decrement active data sources}

END METHOD; {Sendfile}

{----------------------}

END OBJECT;  {data source}
```

```
{----------end Object data----------------------------------------------}

OBJECT muxObj;

{Server inspects the queue. If queue is not empty, then the next item at }
{the head of the queue is removed. If the queue is empty, then the server }
{suspends, and when the next item is added to the queue, the number of }
{idle slots is calculated. The server waits the length of a cell slot, (time}
{to transmit a cell) and then begins inspecting the queue again. If the }
{queue is empty the server suspends, recording the time, which is used to}
{calculate the resynchronisation adjustment for the next slot.}

   ASK METHOD Initalise(IN switchNo : INTEGER; IN useport : INTEGER;
                                                    IN numberofthissite : INTEGER);
   VAR count : INTEGER;

   BEGIN
         sitenumber      := numberofthissite;        {number of this user site}
         attachedswitch  := switchNo;                {number of the nearest switch}
         port            := useport;                 {output port to use}

         {threshold is now a global variable}

         OUTPUT("MUX initalizing at site No ", sitenumber, " with RT1 ", RT1, " RT2 ", RT2);
         NEW(qhead, 1 .. 3);                         {1 = input Q, 2 = output Q}
         NEW(qtail, 1 .. 3);                         {queue stuff}
         NEW(qlen, 1 .. 3);
         NEW(resetqlen, 1 .. 3);                     {queue stats}

         NEW(servingflag, 1 .. 2);                   {create array of flags and set to idle}
         FOR count := 1 TO 2
           servingflag[count] := idle;               {initalising all flags to idle}
         END FOR;

         PSflag          := FALSE;                   {priority service flag}
         dropLPcells     := FALSE;                   {drop low priority cells flag}
         throttledataflag := FALSE;                  {slow data cells flag}
         blockdataflag   := FALSE;                   {stop data cells flag}

END METHOD; {Initalise}

{-----------------------------}

ASK METHOD AddtomuxQ(IN ATMcell : cell; IN queueNo : INTEGER);
{add next cell to end of queue and call the server if flag is idle}
{set flags for different service when adding cells to queues}
{First threshold causes server to switch to priority service}
{second threshold causes server to drop low priority cells}
{LP cells will always be RT cells}

BEGIN
 IF (queueNo = outgoingMUXQ) AND (qlen[outgoingMUXQ] >= T2)      {then can not accept this cell}
       DISPOSE(ATMcell);                                        {discard this cell}
       INC(blockeddatacell);                                    {count blocked data cell}
 ELSE                                                  {add the cell to the queue}
       INC(resetqlen[queueNo]);                       {Q counter reset at each Stat call}
```

```
INC(qlen[queueNo]);                                  {Q counter for actual queue}
IF (qlen[queueNo] = 1)                               {then put 1st item on queue and call server}
   qtail[queueNo] := ATMcell;                        {qtail points to 1st cell}
   qhead[queueNo] := ATMcell;                        {qhead points to 1st cell}
   ATMcell.prev   := NILREC;
   ATMcell.next   := NILREC;                         {only item in queue}

   IF (queueNo = 1) AND (servingflag[queueNo] = idle) {then}
         servingflag[queueNo] := busy;               {set flag to busy and then}
         ServemuxQ(queueNo);                         {activate server for incoming Q}
   ELSIF (queueNo = 2) OR (queueNo = 3)
         IF (servingflag[outgoingMUXQ] = idle)       {then}
          servingflag[outgoingMUXQ] := busy;
          CyclicServer(queueNo);                     {activate server for the outgoing Q}
         END IF;
   END IF;

ELSE {flag is already busy and there are already cells in the queue, so add to end of queue}
   ATMcell.next        := NILREC;
   qtail[queueNo].next := ATMcell;                   {last cell in queue}
   ATMcell.prev        := qtail[queueNo];            {point tail at last cell}
   qtail[queueNo]      := ATMcell;

END IF;                                              {first in queue or not}

IF (qlen[realtimeQ] > RT1) AND (PSflag = FALSE) {then}
         PSflag := TRUE;                             {set priority service flag}
END IF;

IF (qlen[realtimeQ] > RT2) AND (dropLPcells = FALSE) {then}
         dropLPcells := TRUE;                        {set the drop low priority cells flag}
END IF;

IF (qlen[outgoingMUXQ] > T1) AND (throttledataflag = FALSE) {then}
         throttledataflag := TRUE;
END IF;

IF (qlen[outgoingMUXQ] > T2) AND (blockdataflag = FALSE) {then}
         blockdataflag := TRUE;                      {setting block data flag}
END IF;

 {Release flags as queue length falls below the threshold}
 IF (qlen[outgoingMUXQ] < T1) AND (throttledataflag = TRUE) {then}
         throttledataflag := FALSE;                  {release throttle back data flag}
 END IF;

 IF (qlen[outgoingMUXQ] < T2) AND (blockdataflag = TRUE) {then}
         blockdataflag := FALSE;                     {release block data flag}
 END IF;

 IF (qlen[realtimeQ] < RT2) AND (dropLPcells = TRUE) {then}
         dropLPcells := FALSE;                       {release drop low priority cells flag}
 END IF;

 IF (qlen[realtimeQ] < RT1) AND (PSflag = TRUE) {then}
         PSflag := FALSE;                            {release priority service flag}
```

```
        END IF;

 END IF; {nonRT queue delete}

END METHOD;  {addtomuxQ}
{-----------------------------}

TELL METHOD ServemuxQ(IN queueNo : INTEGER);

 {Incoming MUX Q single server - service time is a constant}
 {the number of idle slots is calculated, and the next available slot }
 {is assigned. The server loops round and serves queue until there is }
 {nothing in the queue. The server flag is set to idle and the stopping }
 {time recorded}

 VAR servedATMcell   : cell;          {ATM cell removed from queue to transmit}
       idlecells     : REAL;          {resynchronise server}
       idleslots     : REAL;
       source        : sourceType;
       resynchronization : REAL;
       stopTime      : REAL;

 BEGIN

  stopTime          := muxTime[sitenumber].instopTime;   {incoming queue}
  idleslots         := SimTime() - stopTime;
  idlecells         := FLOAT( ROUND(idleslots + 0.5));
  resynchronization := idlecells - idleslots;

  WAIT DURATION resynchronization; END WAIT;        {wait to start of next slot}

  WHILE (qlen[queueNo] > 0)                         {server serves while there are cells in the queue}
       IF (endrun = TRUE) EXIT; END IF;             {stop server when simulation ends}
       {remove cell from queue and update delay stats}
       servedATMcell := qhead[queueNo];             {remove 1st item in queue}
       qhead[queueNo] := qhead[queueNo].next;       {point head to next item}
       DEC(qlen[queueNo]);
       DEC(resetqlen[queueNo]);                     {update queue Stats}

       ASK usersite[sitenumber] TO Receivecell(servedATMcell);

  END WHILE;  {loops around if there is another cell to serve}

  servingflag[queueNo] := idle;                     {release the flag}

 muxTime[sitenumber].instopTime := SimTime();       {time the server stops serving in queue}

END METHOD;  {ServemuxQ}
{-----------------------}

 TELL METHOD CyclicServer(IN queueNo : INTEGER);
       {Outgoing MUX queue server - same as above but with a}
       {cyclic server and priority service when thresholds are reached}
       {priority service flags are released when the Q length falls below}
       {the various thresholds}
```

```
VAR  servedATMcell   : cell;                    {ATM cell removed from queue to transmit}
     idlecells       : REAL;                    {INTEGER;}
     idleslots       : REAL;                    {No of idle slots to wait}
     source          : sourceType;
     resynchronization : REAL;
     stopTime        : REAL;

BEGIN
     {resynchronise the server after idle time}
     stopTime := muxTime[sitenumber].outstopTime;      {outgoing queue}
     idleslots      := SimTime() - stopTime;
     idlecells      := FLOAT( ROUND(idleslots + 0.5));
     resynchronization := idlecells - idleslots;

     WAIT DURATION resynchronization; END WAIT; {wait to start of next slot}

     INC(utaccesslink[sitenumber]);

     WHILE (qlen[2] > 0) OR (qlen[3] > 0)       {continue if there are cells to serve in either queue}

      IF (endrun = TRUE) EXIT; END IF; {check stop flag}

      IF (qlen[realtimeQ] > RT1)  AND (PSflag = FALSE) {then}
            PSflag := TRUE;                     {set flag if greater than the threshold}
      END IF;

      IF (qlen[queueNo] > 0) {remove cell from queue and update delay stats}
        servedATMcell := qhead[queueNo];        {remove 1st item in queue}
        qhead[queueNo] := qhead[queueNo].next; {point head to next item}
        DEC(qlen[queueNo]);
        DEC(resetqlen[queueNo]);                {update queue Stats}
        IF (queueNo = 3) {then calculate delay in RT queue}
          tdRTout[sitenumber] := SimTime() - servedATMcell.cellstartTime;
      ELSE                                      {calculate delay in non-RT queue}
            tdnonRTout[sitenumber] := SimTime() - servedATMcell.cellstartTime;
      END IF;
      IF (dropLPcells = TRUE) AND (servedATMcell.CLP = 1)
            IF (servedATMcell.source = video)           {then count video cells}
              INC(droppedLPvideocell)                   {no waiting - serve next cell}
            ELSIF (servedATMcell.source = speech)       {then count speech cells}
              INC(droppedLPspeechcell)                  {no waiting - serve next cell}
            ELSE {must be data cell}
              INC(droppedLPdatacell);                   {should not be able to do this}
            END IF;
            DISPOSE(servedATMcell);                     {delete LP cell}
      ELSE
            WAIT DURATION serviceTime; END WAIT;        {transmit cell}
            ASK ATMnetwork TO AccessNW(servedATMcell, attachedswitch, port);
      END IF;
     END IF;

     {Release flags as queue length falls below the threshold}
    IF (qlen[outgoingMUXQ] < T1) AND (throttledataflag = TRUE) {then}
            throttledataflag := FALSE;              {release throttle back data flag}
    END IF;
```

```
IF (qlen[outgoingMUXQ] < T2) AND (blockdataflag = TRUE) {then}
        blockdataflag := FALSE;                    {release throttle back data flag}
END IF;

IF (qlen[realtimeQ] < RT2) AND (dropLPcells = TRUE) {then}
        dropLPcells := FALSE;                      {release drop low priority cells flag}
END IF;

IF (qlen[realtimeQ] < RT1) AND (PSflag = TRUE) {then}
        PSflag := FALSE;                           {release priority service flag}
END IF;

{Select service required for this queue}
IF (PSflag = TRUE) {THEN}
        queueNo := 3;                              {serve RT queue only}
ELSE
        IF  (queueNo = 3)                          {then serve alternately}
           queueNo := 2;
        ELSE
           queueNo := 3;
        END IF;
END IF;

END WHILE;                                         {loop to check if there are more cells to serve}

PSflag := FALSE;                                   {finished so release all service flags}
dropLPcells := FALSE;
throttledataflag := FALSE;
blockdataflag := FALSE;
servingflag[outgoingMUXQ] := idle;                 {release the serving flag}
muxTime[sitenumber].outstopTime := SimTime();      {save stop time}
DEC(utaccesslink[sitenumber]);                     {update utilization of MUX}
  END METHOD; {CyclicServer}

END OBJECT; {server MUX}

{------------end Object MUX-----------------------------------------------------}

{---------code for user sites----------}

OBJECT usersiteObj;

  ASK METHOD Initalise(IN numberofthissite : INTEGER);

  VAR switchnum        : INTEGER;      {attached switch number for each site}
      useport          : INTEGER;      {ATM port on the switch for site to use}
      lam              : REAL;         {interarrival time for sources}
      peakrate         : REAL;         {peak rate for video}
      burstsize        : REAL;         {input param for video method Generate}
      percentage       : REAL;         {input param for batch traffic}
      phonecall        : phonecallObj; {temp phone call obj to initalise array}
      maxNodatafiles : INTEGER;        {max No of datafiles allowed}
      counter  : INTEGER;              {loop counter for arrays}
      tempPBXrec       : calls;        {temporary record for initalizing the PBX}
```

```
        tempLBrec       : policingrec;      {temporary record for initalising LB}
        sourcecounter   : sourceType;       {source type for sources}

  BEGIN
        sitenumber := numberofthissite; {user site number}
        OUTPUT("****Enter params for usersite number ", sitenumber, " ***");

        {assign switches and ports for each user site to use}
        IF   (sitenumber = 1) {then} useport := 2; switchnum := 1;
                NEW(Speech); NEW(Video); NEW(Data); NEW(mux);
        ELSIF (sitenumber = 2) {then} useport := 4; switchnum := 5;
                NEW(Speech); NEW(Video); NEW(Data); NEW(mux);
        ELSIF (sitenumber = 3) {then} useport := 9; switchnum := 9;
                NEW(mux);
        ELSIF  (sitenumber = 4) {then} useport := 2; switchnum := 3;
                NEW(Speech); NEW(Video); NEW(Data); NEW(mux);
        ELSE  {sitenumber = 5}  useport := 99; switchnum := 99;
                NEW(mux);
        END IF;

        {new objects for this user site}

IF (sitenumber = 1) OR (sitenumber = 2) OR (sitenumber = 4) {then}
{****Phone call data ****}
        {OUTPUT("enter max number of phone calls allowed");}

        INPUT(maxallowedcalls);
        INPUT(lam);

        NEW(PBX, 1 .. maxallowedcalls);                 {create PBX}
        FOR counter := 1 TO maxallowedcalls;            {and initalize}
                NEW(tempPBXrec);
                PBX[counter] := tempPBXrec;
                PBX[counter].callstatus := free;
                NEW(phonecall);
                PBX[counter].call := phonecall;
        END FOR;

        {Initalize the LB}
        NEW(policing, video .. data, 1 .. maxallowedcalls);

        sourcecounter := speech;
        FOR counter := 1 TO maxallowedcalls;                    {LB for speech}
                NEW(tempLBrec);
                policing[sourcecounter, counter] := tempLBrec;
        END FOR;

        ASK Speech TO Initalise(lam, sourcecounter, sitenumber);

{*****Video data ******}

        lam := videorefresh; {halved in Video Generate}
        INPUT(peakrate);
        INPUT(maxvideosources);
        sourcecounter := video;
        ASK Video TO Initalise(lam, sourcecounter, sitenumber);
```

```
        FOR counter := 1 TO maxvideosources;                    {Initalising LB for video}
                NEW(templBrec);
                policing[sourcecounter, counter] := tempLBrec;
        END FOR;

{******Data data******}
        INPUT(maxNodatafiles);

{Initalise LB for data}
        sourcecounter := data;
        FOR counter := 1 TO maxNodatafiles;                     {Initalise LB for data}
          NEW(tempLBrec);
          policing[sourcecounter, counter] := tempLBrec;
          NEW(policing[sourcecounter, counter].qhead);
          NEW(policing[sourcecounter, counter].qtail);
          policing[sourcecounter, counter].busyflag := idle;
        END FOR;

        INPUT(lam);                                             {enter the time between data files}
        ASK Data TO Initalise(lam, sourcecounter, sitenumber);
END IF;

{******MUX data *****}
        ASK mux TO Initalise(switchnum, useport, sitenumber);

{**Loop to generate source objects***}
IF (sitenumber = 1) {then}
        TELL Speech TO Generate;
        ASK Video TO Createvideosources(maxvideosources, peakrate);
        TELL Data TO Generate(maxNodatafiles);
ELSIF (sitenumber = 2) {then}
        TELL Speech TO Generate;
        ASK Video TO Createvideosources(maxvideosources, peakrate);
        TELL Data TO Generate(maxNodatafiles);
ELSIF (sitenumber = 4) {then}
        TELL Speech TO Generate;
        ASK Video TO Createvideosources(maxvideosources, peakrate);
        TELL Data TO Generate(maxNodatafiles);
END IF;

  END METHOD; {Initalise usersite}
  {--------------------}

  ASK METHOD InitaliseLeakybucket(IN ID : INTEGER; IN sourcetype : sourceType;
                                  IN meanburst : REAL; IN rate : REAL);

  {when new source starts up it calls this method to initalise params}

  VAR delta : REAL;                                             {delay jitter}
      d     : INTEGER;                                          {decrement value}

  BEGIN
        policing[sourcetype, ID].TECflag := FALSE;
        policing[sourcetype, ID].decrementflag := FALSE;
        policing[sourcetype, ID].counter := 0;
        policing[sourcetype, ID].activesource := engaged;
```

```
        {use larger decrement period for higher rate sources}

        IF (rate <= 1.0E6) {then} delta := 20.0E-3; d := 1;
        ELSE {rate > 1 Mb/s} delta := 0.2E-3;
                IF (sourcetype = video) {then}
                        d := ROUND(meanburst);
                ELSE    d := 16; END IF;
        END IF;

        policing[sourcetype, ID].decrement := d;
        policing[sourcetype, ID].threshold := 1 + d + ROUND((delta * rate) / cellpayload);
        policing[sourcetype, ID].slotdecrement := ((FLOAT(d) / rate) * cellpayload) * recipofslot;

END METHOD; {Initalise Leaky Bucket}
{--------------------}

ASK METHOD AddtoLBbuffer(IN ATMcell : cell; IN ID : INTEGER);
        {adds data to the LB buffer queue}

VAR source : sourceType;

BEGIN
        IF (ATMcell.source = video) OR (ATMcell.source = speech) {then}
           source := ATMcell.source;
        ELSE
           source := data;
        END IF;                                         {put source type into source variable}

        INC(policing[source, ID].qlen);                 {increment queue length}

        IF (policing[source, ID].qlen = 1)              {then 1st cell in queue}
          policing[source, ID].qtail := ATMcell;
          policing[source, ID].qhead := ATMcell;
          ATMcell.prev := NILREC;
          ATMcell.next := NILREC;
        ELSE                                            {add to end of queue}
          ATMcell.next := NILREC;
          policing[source, ID].qtail.next := ATMcell;
          ATMcell.prev := policing[source, ID].qtail;
          policing[source, ID].qtail := ATMcell;
        END IF;

        IF (policing[source, ID].busyflag <> busy) {then}
                policing[source, ID].busyflag := busy;
                TELL usersite[sitenumber] TO Leakybucket(ID, source);
        END IF;

END METHOD;  {AddtoLBbuffer}

{--------------------}

TELL METHOD Leakybucket(IN ID : INTEGER; IN source : sourceType);
        {policing method for data - includes buffer}

VAR   nextcell : cell;                                  {next ATM cell removed from q and served}
```

```
        service : REAL;                                  {service time for a data cell - may be 2X}

BEGIN
      IF (source <> video) AND (source <> speech)        {then data cells}
            source := data;
      END IF;
      IF (policing[source, ID].decrementflag = FALSE)    {then}
        policing[source, ID].decrementflag := TRUE;
        Decrementfunction(ID, source) ;                  {call decrement function}
      END IF;
      IF (policing[source, ID].busyflag <> busy)         {then}
            policing[source, ID].busyflag := busy;
      END IF;

      WHILE (policing[source, ID].qlen > 0)
        IF (mux.throttledataflag = TRUE) {then}
            service := (2.0 * serviceTime);              {serve data cells 2X as slow}
        ELSE
            service := serviceTime;
        END IF;

        WAIT DURATION service; END WAIT;

        IF (policing[source, ID].counter < policing[source, ID].threshold)     {then remove next cell}
            nextcell := policing[source, ID].qhead;
            policing[source, ID].qhead := policing[source, ID].qhead.next;
            DEC(policing[source, ID].qlen);
            PasstoMux(source, nextcell); {send to multiplexer}

        END IF;

        IF (endrun = TRUE) {then} EXIT; END IF;                                {stop if end of run}
      END WHILE;                                                               {stop when queue empty}

      policing[source, ID].busyflag := idle;                                   {release flag}

END METHOD; {leaky bucket queue}

{--------------------}
ASK METHOD VirtualLeakybucket(IN ATMcell : cell);
{virtual leaky bucket - allows cells to pass but does not delay them.}
{Checks to see if decrement function is working - calls it if not and }
{sets flag - checks counter tags cell if counter is above the threshold}
{and pass to Mux}

VAR  ID : INTEGER;                                   {VCI label from cell}
       source : sourceType;                          {source type}

BEGIN
      ID := ATMcell.VCid;                            {extract index for policing}
      source := ATMcell.source;

      {start of new burst, so reset TEC function}
      IF (source = video) AND (policing[source, ID].TECflag = TRUE) AND (ATMcell.GFCfield = 1);
            policing[source, ID].TECflag  := FALSE;
```

```
END IF;

{simplify source identifier for data}
IF (source = data1) OR (source = data2) OR (source = data3) OR
  (source = data4) OR (source = data5) OR (source = data6)
{then change to data}  source := data;
END IF;

{check that the decrement function has been activated for this VC}
IF (policing[source, ID].decrementflag  = FALSE)     {then start it}
   policing[source, ID].decrementflag := TRUE;
   Decrementfunction(ID, source);                    {call the decrementing function}
END IF;

{****Leaky bucket bit****}
IF (policing[source, ID].counter < policing[source, ID].threshold)
  {then cell valid, so count and pass to mux, unless TEC function}
  {is activated, in which case delete cell}

   IF (policing[source, ID].TECflag = TRUE)          {then clipping video tail}
         DISPOSE(ATMcell);                           {dispose of video cell}
         INC(taggedvideo);                           {count deleted cell}
   ELSE
         INC(policing[source, ID].counter);          {increment counter}
         PasstoMux(source, ATMcell);

   END IF;
ELSE                                {counter > threshold so tag cells and activate TEC function}
    IF (source = video)             {then must delete it}

         IF (policing[source, ID].TECflag = FALSE)          {then start TEC function}
           policing[source, ID].TECflag := TRUE;
           INC(clippedslice);                               {counting the number of slices clipped}
         END IF;
         DISPOSE(ATMcell);                                  {dispose of video cell}
         INC(taggedvideo);                                  {count deleted video cell}
      ELSE  {tag cell and pass to Mux}
         IF (source = speech) {then}
               INC(taggingspeech);                          {counting tagged cells}
         ELSE    INC(taggingdata);
         END IF;
         ATMcell.CLP := 1;
         PasstoMux(source, ATMcell);

      END IF;

   END IF; {leaky bucket bit}

END METHOD; {Leakybucket}
{----------------------}

ASK METHOD PasstoMux(IN source : sourceType; IN ATMcell : cell);

BEGIN
   IF (source = speech) OR (source = video)                 {then pass to RT traffic}
      ASK usersite[sitenumber].mux TO AddtomuxQ(ATMcell, realtimeQ);
```

```
        ELSE     {pass to non-RT traffic}
            ASK usersite[sitenumber].mux TO AddtomuxQ(ATMcell, outgoingMUXQ);
        END IF;

END METHOD;  {pass to Mux from LB}
{----------------------}

 TELL METHOD Decrementfunction(IN ID : INTEGER; IN sourcetype : sourceType);
 {decrements counter periodically}

 BEGIN

 WHILE (policing[sourcetype,ID].activesource = engaged) {continue}

        IF (endrun = TRUE) EXIT; END IF;

        WAIT DURATION (policing[sourcetype,ID].slotdecrement);   END WAIT;

        policing[sourcetype, ID].counter :=
        policing[sourcetype, ID].counter - policing[sourcetype,ID].decrement;

        IF (policing[sourcetype,ID].counter < 0)                    {then}
          policing[sourcetype,ID].counter := 0;                     {negative counter not allowed}
        END IF;

 END WHILE;

  policing[sourcetype, ID].decrementflag := FALSE;                  {release flag}

 END METHOD; {Decrementfunction}
{----------------------}

ASK METHOD Increment(IN source : sourceType);
{The source is video, speech or data. Used to count cells of each type.}

BEGIN

        CASE source

                WHEN video:  INC(videoTotal);                       {cells generated by each source}
                WHEN speech: INC(speechTotal);

                WHEN data1:  INC(data1Total);
                WHEN data2:  INC(data2Total);
                WHEN data3:  INC(data3Total);
                WHEN data4:  INC(data4Total);
                WHEN data5:  INC(data5Total);

        END CASE;

END METHOD; {Increment}
{---------------------}

ASK METHOD IncBlocks(IN block : INTEGER; IN source : sourceType);
{update stats on burst sizes}
```

```
BEGIN

        burstStats[sitenumber] := block;

END METHOD; {IncBlocks}
{--------------------}

ASK METHOD PBXfreeline(IN linenumber : INTEGER; IN callTimestamp : REAL);
{change the status of a phone line at the exchange to free and update Stats}

BEGIN
        PBX[linenumber].callstatus := free;                    {change line status within PBX}
        DISPOSE(PBX[linenumber].call);                         {release line}
        DEC(numberofcalls);                                    {update stats on numbers of calls}
        callStats[sitenumber] := SimTime() - callTimestamp;    {end of call Stats}
END METHOD;  {PBXfreeline}
{----------------------}


ASK METHOD AddtoPBX(IN linenumber : INTEGER);
        {add a new phone call to the PBX}

VAR call : phonecallObj;

BEGIN
        INC(numberofcalls);                                    {stats on total numbers}
        NEW(call);
        PBX[linenumber].call := call;
        PBX[linenumber].callstatus := engaged;
END METHOD; {add to PBX}
{----------------------}

ASK METHOD CAC(IN newrate : REAL; IN source : sourceType; IN ID : INTEGER;
                                                    OUT accepted : BOOLEAN);
        {reserves network resources and activates policing line}

BEGIN

        IF ((totalcapacity + newrate) < linkcapacity)          {the call allowed}
                accepted := TRUE;
                totalcapacity := newrate + totalcapacity;
                policing[source, ID].activesource := engaged;
        ELSE
                accepted := FALSE;                             {call blocked, capacity not updated}
        END IF;
END METHOD;  {CAC method}
{----------------------}

ASK METHOD ReleaseResources(IN rate : REAL; IN source : sourceType;
                                    IN ID : INTEGER);
        {updates the current capacity of the link and frees policing function}

BEGIN
```

```
        totalcapacity := totalcapacity - rate;                    {release capacity}
        policing[source, ID].activesource := free;                {this turns off LB function}

END METHOD;  {Release Resources}
{----------------------}

ASK METHOD Receivecell(IN receivedATMcell : cell);
{cells received at a usersite are counted, and the end-to-end delay for}
{all cells and for individual types of cell are calculated}
{the delay for the last cell of each type are also calculated}

VAR source : sourceType;
        origin : INTEGER;

BEGIN

  source := receivedATMcell.source;
  origin := receivedATMcell.origin;

  {count all cells received regardless of destination}
  CASE source
        WHEN video:
     CASE origin
           WHEN 1: INC(receive1video);                    {from all sites}
           WHEN 2: INC(receive2video);
           WHEN 4: INC(receive4video);
         END CASE;

        WHEN speech:
     CASE origin
           WHEN 1: INC(receive1speech);
           WHEN 2: INC(receive2speech);
           WHEN 4: INC(receive4speech);
     END CASE;

     WHEN data1:
           CASE origin
              WHEN 1: INC(receive1data1);
              WHEN 2: INC(receive2data1);
              WHEN 4: INC(receive4data1);
           END CASE;

     WHEN data2:
           CASE origin
              WHEN 1: INC(receive1data2);
              WHEN 2: INC(receive2data2);
              WHEN 4: INC(receive4data2);
           END CASE;

     WHEN data3:
           CASE origin
              WHEN 1: INC(receive1data3);
              WHEN 2: INC(receive2data3);
              WHEN 4: INC(receive4data3);
           END CASE;
```

```
            WHEN data4:
                    CASE origin
                       WHEN 1: INC(receive1data4);
                       WHEN 2: INC(receive2data4);
                       WHEN 4: INC(receive4data4);
                    END CASE;

            WHEN data5:
                    CASE origin
                       WHEN 1: INC(receive1data5);
                       WHEN 2: INC(receive2data5);
                       WHEN 4: INC(receive4data5);
                    END CASE;
          END CASE; {counting all cells}

{count ONLY cells received by site 3 that have crossed the network}
    IF (sitenumber = 3)                                    {then count the cells received at site 3}
    CASE source
         WHEN video:
      CASE origin
           WHEN 1: INC(received1video);                    {at site 3 only}
           WHEN 2: INC(received2video);
           WHEN 4: INC(received4video);
         END CASE;
      IF (receivedATMcell.msgstartTime > 0.0)              {then}
          bursttimeStats[origin] := SimTime() - receivedATMcell.msgstartTime;
      END IF;                                              {delay for a video burst}

        WHEN speech:
      CASE origin
           WHEN 1: INC(received1speech);
           WHEN 2: INC(received2speech);
           WHEN 4: INC(received4speech);
      END CASE;

      WHEN data1:
            CASE origin
               WHEN 1: INC(received1data1);
               WHEN 2: INC(received2data1);
               WHEN 4: INC(received4data1);
            END CASE;

      WHEN data2:
             CASE origin
               WHEN 1: INC(received1data2);
               WHEN 2: INC(received2data2);
               WHEN 4: INC(received4data2);
            END CASE;

      WHEN data3:
            CASE origin
               WHEN 1: INC(received1data3);
               WHEN 2: INC(received2data3);
               WHEN 4: INC(received4data3);
            END CASE;
```

```
        WHEN data4:
                CASE origin
                   WHEN 1: INC(received1data4);
                   WHEN 2: INC(received2data4);
                   WHEN 4: INC(received4data4);
                END CASE;

        WHEN data5:
                CASE origin
                   WHEN 1: INC(received1data5);
                   WHEN 2: INC(received2data5);
                   WHEN 4: INC(received4data5);
                END CASE;

      END CASE;

END IF;       {site No 3 only}

IF (source = data1) OR (source = data2) OR (source = data3) OR (source = data4) OR (source = data5)
                                                                                    {then}
      source := data;
END IF;

{only count end-to-end delays destined for site 3}
IF (sitenumber = 3)

      {origin = 1 or 4 across whole N/W or 2 using only 2 switches}

       CASE source
        WHEN speech :
                tdspeech[origin] := SimTime() -  receivedATMcell.cellstartTime;
                tdRTin[origin] := SimTime() - receivedATMcell.cellstartTime;
                                                    {end-to-end delay in slots for RT cells}
          WHEN video :
            tdvideo[origin] := SimTime() -  receivedATMcell.cellstartTime;

                tdRTin[origin] := SimTime() - receivedATMcell.cellstartTime;
                                                    {end-to-end delay in slots for RT cells}
          WHEN data :
                tddata[origin] := SimTime() -  receivedATMcell.cellstartTime;
                                                    {delay for a data cell}
                tdnonRTin[origin] := SimTime() - receivedATMcell.cellstartTime;
                                                    {end-to-end delay in slots for non-RT cells}
          END CASE;

      END IF;{destination site = No 3}

      DISPOSE(receivedATMcell);

 END METHOD; {Receivedcell}
 {----------------------}

ASK METHOD SiteStats;

BEGIN
 OUTPUT("**************");
```

```
OUTPUT("VIDEO cell STATS for cells to site 3");
OUTPUT();
OUTPUT("mean td each cell     ",  ASK(GETMONITOR(tdvideo[sitenumber], RStatObj)) Mean());
OUTPUT("Max td each cell      ",  ASK(GETMONITOR(tdvideo[sitenumber], RStatObj)) Maximum);
OUTPUT("burst size mean       ",  ASK(GETMONITOR(burstStats[sitenumber], IStatObj)) Mean());
OUTPUT("burst size max        ",  ASK(GETMONITOR(burstStats[sitenumber], IStatObj)) Maximum);
OUTPUT("mean burst delay      ",  ASK(GETMONITOR(bursttimeStats[sitenumber], RStatObj)) Mean());
OUTPUT("max burst delay       ", ASK(GETMONITOR(bursttimeStats[sitenumber], RStatObj)) Maximum);
OUTPUT();
OUTPUT("Total bursts sent =   ", ASK(GETMONITOR(burstStats[sitenumber], IStatObj)) Count);
OUTPUT("bursts received       ", ASK(GETMONITOR(bursttimeStats[sitenumber], RStatObj)) Count);
OUTPUT("Current No of slices  ",  ASK(GETMONITOR(Video.numberofslices, IStatObj)) Count);
OUTPUT("No. of clipped slices ", clippedslice);
OUTPUT("mean video rate       ", ASK(GETMONITOR(Video.meanvideorate, RStatObj)) Mean());
OUTPUT();
OUTPUT("DATA cell STATS for cells to site 3");
OUTPUT();
OUTPUT("Active data sources   ", usersite[sitenumber].Data.activesources);
OUTPUT("mean td each cell     ", ASK(GETMONITOR(tddata[sitenumber], RStatObj)) Mean());
OUTPUT("Max td each cell      ", ASK(GETMONITOR(tddata[sitenumber], RStatObj)) Maximum);
OUTPUT();
OUTPUT("SPEECH STATS for cells to site 3");
OUTPUT();
OUTPUT("mean td sp cells      ", ASK(GETMONITOR(tdspeech[sitenumber], RStatObj)) Mean());
OUTPUT("max td sp cells       ", ASK(GETMONITOR(tdspeech[sitenumber], RStatObj)) Maximum);
OUTPUT();
OUTPUT("Actual calls now      ", numberofcalls);
OUTPUT("PHONE CALLS - Mean No. ", ASK(GETMONITOR(numberofcalls, IStatObj)) Mean());
OUTPUT();
OUTPUT("Outgoing MUX queue for site ", sitenumber );
OUTPUT();
OUTPUT("mean OUT length non-RT ",
                    ASK(GETMONITOR(mux.resetqlen[outgoingMUXQ], ITimedStatObj)) Mean() );
OUTPUT("Max Out qlen non-RT    ",
                    ASK(GETMONITOR(mux.resetqlen[outgoingMUXQ], ITimedStatObj)) Maximum);
OUTPUT("Var =                  ",
                    ASK(GETMONITOR(mux.resetqlen[outgoingMUXQ], ITimedStatObj))Variance );
OUTPUT("StdDev =               ",
                    ASK(GETMONITOR(mux.resetqlen[outgoingMUXQ], ITimedStatObj)) StdDev);
OUTPUT("No still in non-RT Q   ", mux.qlen[outgoingMUXQ]);
OUTPUT();
OUTPUT("mean OUT length RT     ",
                    ASK(GETMONITOR(mux.resetqlen[3], ITimedStatObj)) Mean() );
OUTPUT("Max Out qlen RT        ",
                    ASK(GETMONITOR(mux.resetqlen[3], ITimedStatObj)) Maximum);
OUTPUT("Var =                  ",
                    ASK(GETMONITOR(mux.resetqlen[3], ITimedStatObj))Variance );
OUTPUT("StdDev =               ",
                    ASK(GETMONITOR(mux.resetqlen[3], ITimedStatObj)) StdDev);
OUTPUT("No still in RT Q       ", mux.qlen[3]);
OUTPUT();
OUTPUT("VLB stats");
OUTPUT("Video deleted by TEC   ", taggedvideo );
OUTPUT("Speech tagged by LB    ", taggingspeech);
OUTPUT("data tagged by LB      ", taggingdata);
OUTPUT();
```

```
OUTPUT("At Mux");
OUTPUT("Dropped speech cells   ", ASK(GETMONITOR(mux.droppedLPspeechcell, IStatObj)) Count);
OUTPUT("Dropped video  cells   ", ASK(GETMONITOR(mux.droppedLPvideocell, IStatObj)) Count);
OUTPUT("Dropped data cells     ", ASK(GETMONITOR(mux.droppedLPdatacell, IStatObj)) Count);
OUTPUT("blocked data cell (T1) ", ASK(GETMONITOR(mux.blockeddatacell, IStatObj)) Count);
OUTPUT();
OUTPUT("Total capacity now is  ", totalcapacity);

        {resetting Stats}
        ASK(GETMONITOR(mux.resetqlen[2], ITimedStatObj)) TO Reset;
        ASK(GETMONITOR(mux.resetqlen[3], ITimedStatObj)) TO Reset;
        ASK(GETMONITOR(mux.droppedLPspeechcell, IStatObj)) TO Reset;
        ASK(GETMONITOR(mux.droppedLPvideocell, IStatObj)) TO Reset;
        ASK(GETMONITOR(mux.droppedLPdatacell, IStatObj)) TO Reset;
        ASK(GETMONITOR(mux.blockeddatacell, IStatObj)) TO Reset;
        ASK(GETMONITOR(Speech.blockedspeechcalls, IStatObj)) TO Reset;
        ASK(GETMONITOR(Video.meanvideorate, RStatObj)) TO Reset;
        ASK(GETMONITOR(Video.numberofslices, IStatObj)) TO Reset;

        speechTotal := 0;
        videoTotal  := 0;  {resetting generated cell totals - all sources}
        data1Total := 0;
        data2Total := 0;
        data3Total := 0;
        data4Total := 0;
        data5Total := 0;
        data6Total := 0;
        clippedslice := 0;   {resetting count for slices tagged by LB at site}
        taggingspeech := 0;  {resetting counts for cells tagged by LB at site}
        taggingdata := 0;
        taggedvideo := 0;
   END METHOD; {siteStats}

END OBJECT; {usersitObj}

{------------end Object-----------------------------------------------------}

OBJECT usersitemanagerObj;

   ASK METHOD Initalise(IN Numofsites : INTEGER; IN randomseed : INTEGER);

   VAR sitenumber : INTEGER;
        site       : usersiteObj;
        temprec    : muxTimeRec;
        addrec     : entry;
        index      : INTEGER;

   BEGIN
OUTPUT("QL thresholds at mux. NW-link capacity = ", linkcapacity, " used in CAC ");
OUTPUT("Virtual leaky bucket version - cells are tagged and passed to mux");
OUTPUT("TEC function is working - LB to MUX");
OUTPUT("video uses 2x burst for LB params");
        NEW(rand);                                          {random object for all user sites}
        ASK rand TO SetSeed(randomseed);
        OUTPUT("random seed for usersitemanager ", randomseed);
```

```
        INPUT(RT1);                                  {for QLT scheme}
        INPUT(RT2);                                  {  "       }
OUTPUT("Priority Service - RT queue - threshold improves service RT1 = ", RT1);
OUTPUT("Mux deletes cells if Q length is > RT2 threshold = ", RT2);
        INPUT(T1);                                   {thresholds for nonRT queue}
        INPUT(T2);
        OUTPUT("non-RT thresholds - T1 = ", T1, " T2 = ", T2);
        endrun := FALSE;
        numberofsites := Numofsites;
        realtimeQ    := 3;                           {outgoing real-time queue is always 3}
        outgoingMUXQ := 2;                           {outgoing queue at MUX is always 2}
        incomingMUXQ := 1;                           {incoming queue at MUX is always 1}
        NEW(tdRTout, 1 .. numberofsites);                    {delay to access NW - RT cells}
        NEW(tdnonRTout, 1 .. numberofsites);                 {delay to access NW - nonRT cells}
        NEW(tdRTin, 1 .. numberofsites);                     {end to end delay - RT cells}
        NEW(tdnonRTin, 1 .. numberofsites);                  {end to end delay - nonRT cells}
        NEW(tdspeech, 1 .. numberofsites);                   {end to end delay - speech cells}
        NEW(tdvideo, 1 .. numberofsites);                    {end to end delay - video cells}
        NEW(tddata, 1 .. numberofsites);                     {end to end delay - data cells}
        NEW(burstStats, 1 .. numberofsites);
        NEW(bursttimeStats, 1 .. numberofsites);
        NEW(callStats, 1 .. numberofsites);
        NEW(utaccesslink, 1 .. numberofsites);               {one for each MUX}
        NEW(muxTime, 1.. numberofsites);                     {record to store stop time}
        FOR sitenumber := 1 TO numberofsites;                {one for each MUX}
          NEW(temprec);
          muxTime[sitenumber] := temprec;
        END FOR;
        maxentry := 7;
        NEW(addressbook, 1..maxentry);                       {address book entries}
        FOR index := 1 TO maxentry;
          NEW(addrec);
          addressbook[index] := addrec;
        END FOR;

        addressbook[1].siteNo := 1 ;
        addressbook[1].startlabel := 'a';
        addressbook[1].destination := 5;

        addressbook[2].siteNo := 1;
        addressbook[2].startlabel := 'A';
        addressbook[2].destination := 3;

        addressbook[7].siteNo := 1 ;
        addressbook[7].startlabel := 'x';
        addressbook[7].destination := 2;

        addressbook[3].siteNo := 4;
        addressbook[3].startlabel := 'm';
        addressbook[3].destination := 5;

        addressbook[4].siteNo := 4;
        addressbook[4].startlabel := 'M';
        addressbook[4].destination := 2;

        addressbook[5].siteNo := 4;
```

```
        addressbook[5].startlabel := 'p';
        addressbook[5].destination := 3;

        addressbook[6].siteNo := 2;
        addressbook[6].startlabel := 'z';
        addressbook[6].destination := 3;

        NEW(usersite, 1.. numberofsites);
        FOR sitenumber := 1 TO numberofsites;
           NEW(site);
           usersite[sitenumber] := site;                    {setting up array of usersites}
           ASK usersite[sitenumber] TO Initalise(sitenumber);

        END FOR;

OUTPUT();

  END METHOD; {Initalise for user site manager}
   {--------------------}

   ASK METHOD Accessusersite(IN ATMcell : cell; IN siteNo : INTEGER);
   {method used by the network to access the users site MUX}

   VAR queueNo : INTEGER;

   BEGIN
        queueNo := incomingMUXQ;                            {input queue at MUX}
        ASK usersite[siteNo].mux TO AddtomuxQ(ATMcell, queueNo);

   END METHOD; {access usersite}
   {-----------------------}

   TELL METHOD Monitor(IN runlength : REAL; IN interval : REAL);
   {run length and interval are in seconds - convert to slots}

    BEGIN
OUTPUT("monitor in user site manager starting");
        runlength := runlength * recipofslot;
        interval := interval * recipofslot;
        endrun := FALSE;
        WHILE (SimTime() < runlength)
          WAIT DURATION interval; END WAIT;
          OUTPUT("**********************");
          OUTPUT("asking all user sites to output their Stats");
          OUTPUT();
          ASK SELF UsersiteStats;
          OUTPUT("**********************");
        END WHILE;
        endrun := TRUE;
        TERMINATE;
    END METHOD; {monitor}

    {-----------------------}

   ASK METHOD UsersiteStats;
   VAR count : INTEGER;
```

```
        realtime : REAL;
        r3speech, rspeech, gspeech : INTEGER;
        r3video, rvideo, gvideo : INTEGER;

{comment end comment}
        r3data1, rdata1, gdata1 : INTEGER;
        r3data2, rdata2, gdata2 : INTEGER;
        r3data3, rdata3, gdata3 : INTEGER;
        r3data4, rdata4, gdata4 : INTEGER;
        r3data5, rdata5, gdata5 : INTEGER;

  BEGIN
        {call all user site objects to give their stats}

        OUTPUT("****USER SITE STATS*****");
        realtime := SimTime();
        OUTPUT("SimTime = ", (realtime * slot), " seconds");

  FOR count := 1 TO numberofsites

   OUTPUT();
   OUTPUT("   USER SITE NUMBER ", count);
IF count = 1;
  rspeech := receive1speech;                          {received by all sites}
  rvideo := receive1video;
  r3speech := received1speech;                        {received by site3 only}
  r3video := received1video;
  rdata1 := receive1data1;        r3data1 := received1data1;
  rdata2 := receive1data2;        r3data2 := received1data2 ;
  rdata3 := receive1data3;        r3data3 := received1data3;
  rdata4 := receive1data4;        r3data4 := received1data4;
  rdata5 := receive1data5;        r3data5 := received1data5;
ELSIF count = 2
  rspeech := receive2speech;                          {received by all sites}
  rvideo := receive2video;
  r3speech := received2speech;                        {received by site3 only}
  r3video := received2video;
  rdata1 := receive2data1;        r3data1 := received2data1;
  rdata2 := receive2data2;        r3data2 := received2data2;
  rdata3 := receive2data3;        r3data3 := received2data3 ;
  rdata4 := receive2data4;        r3data4 := received2data4;
  rdata5 := receive2data5;        r3data5 := received2data5;
ELSIF (count = 4)
  rspeech := receive4speech;                          {received by all sites}
  rvideo := receive4video;
  r3speech := received4speech;                        {received by site3 only}
  r3video := received4video;
  rdata1 := receive4data1;  r3data1 := received4data1;
  rdata2 := receive4data2;  r3data2 := received4data2;
  rdata3 := receive4data3;  r3data3 := received4data3;
  rdata4 := receive4data4;  r3data4 := received4data4;
  rdata5 := receive4data5;  r3data5 := received4data5;
END IF;

IF (count = 1) OR (count = 2) OR (count = 4);
   gspeech := usersite[count].speechTotal;
```

```
    gvideo :=  usersite[count].videoTotal;

    gdata1 :=  usersite[count].data1Total;
    gdata2 := usersite[count].data2Total;
    gdata3 := usersite[count].data3Total;
    gdata4 := usersite[count].data4Total;
    gdata5 := usersite[count].data5Total;

    OUTPUT("received speech at 3  ", r3speech);
    OUTPUT("received video  at 3  ", r3video);
    OUTPUT("received data 1 at 3  ", r3data1);
    OUTPUT("received data 2 at 3  ", r3data2);
    OUTPUT("received data 3 at 3  ", r3data3);
    OUTPUT("received data 4 at 3  ", r3data4);
    OUTPUT("received data 5 at 3  ", r3data5);
    OUTPUT();
    OUTPUT("received speech      ", rspeech);
    OUTPUT("received video       ", rvideo);
    OUTPUT("received data 1      ", rdata1);
    OUTPUT("received data 2      ", rdata2);
    OUTPUT("received data 3      ", rdata3);
    OUTPUT("received data 4      ", rdata4);
    OUTPUT("received data 5      ", rdata5);
    OUTPUT("generated speech     ", gspeech );
    OUTPUT("generated video      ", gvideo );
    OUTPUT("generated data 1     ", gdata1);
    OUTPUT("generated data 2     ", gdata2);
    OUTPUT("generated data 3     ", gdata3);
    OUTPUT("generated data 4     ", gdata4);
    OUTPUT("generated data 5     ", gdata5);
OUTPUT();
OUTPUT("Totals received by 3  ", (r3speech + r3video + r3data1 + r3data2 + r3data3 + r3data4 + r3data5));
OUTPUT();
OUTPUT("Totals all received   ", (rspeech + rvideo + rdata1 + rdata2 + rdata3 + rdata4 + rdata5));
OUTPUT("Total generated       ",
                    (gspeech + gvideo + gdata1 + gdata2 + gdata3 + gdata4 + gdata5 ), " at site ", count);
OUTPUT();
OUTPUT("Total RT cells sent   ", (gspeech + gvideo));
OUTPUT("Total RTs received    ", (rspeech + rvideo));
OUTPUT();
OUTPUT("Total nonRTs sent     ", (gdata1 + gdata2 + gdata3 + gdata4 + gdata5));
OUTPUT("Total nonRTs received ", (rdata1 + rdata2 + rdata3 + rdata4 + rdata5));
OUTPUT("**********************************");
OUTPUT("link utilization mean ", ASK(GETMONITOR(utaccesslink[count], ITimedStatObj)) Mean());
OUTPUT("link utilization Max  ", ASK(GETMONITOR(utaccesslink[count], ITimedStatObj)) Maximum);
OUTPUT();
OUTPUT("REALTIME - all cells to access N/W");
OUTPUT("Mean delay to access  ", ASK(GETMONITOR(tdRTout[count], RStatObj)) Mean());
OUTPUT("Max delay to access   ", ASK(GETMONITOR(tdRTout[count], RStatObj)) Maximum);
OUTPUT("delay var to access   ", ASK(GETMONITOR(tdRTout[count], RStatObj)) Variance);
OUTPUT("SD of delay to access ", ASK(GETMONITOR(tdRTout[count], RStatObj)) StdDev);

IF (count = 2)  {then output this stuff}
OUTPUT("These cells are not crossing whole network");
END IF;
OUTPUT("REALTIME - arriving at site 3 only");
```

```
OUTPUT("Mean delay to cross   ", ASK(GETMONITOR(tdRTin[count], RStatObj)) Mean());
OUTPUT("Max delay to cross    ", ASK(GETMONITOR(tdRTin[count], RStatObj)) Maximum);
OUTPUT("delay var to cross    ", ASK(GETMONITOR(tdRTin[count], RStatObj)) Variance );
OUTPUT("SD of delay to cross  ", ASK(GETMONITOR(tdRTin[count], RStatObj)) StdDev);
OUTPUT();
OUTPUT("NON-REALTIME - all cells to access N/W");
OUTPUT("Mean delay to access  ", ASK(GETMONITOR(tdnonRTout[count], RStatObj)) Mean());
OUTPUT("Max delay to access   ", ASK(GETMONITOR(tdnonRTout[count], RStatObj)) Maximum);
OUTPUT("delay var to access   ", ASK(GETMONITOR(tdnonRTout[count], RStatObj)) Variance);
OUTPUT("SD of delay to access ", ASK(GETMONITOR(tdnonRTout[count], RStatObj)) StdDev);

IF (count = 2)  {then output this stuff}
OUTPUT("These cells are not crossing whole network");
END IF;
OUTPUT("NON-REALTIME - arriving at site 3 only");
OUTPUT("Mean delay to cross   ", ASK(GETMONITOR(tdnonRTin[count], RStatObj)) Mean());
OUTPUT("Max delay to cross    ", ASK(GETMONITOR(tdnonRTin[count], RStatObj)) Maximum);
OUTPUT("delay var to cross    ", ASK(GETMONITOR(tdnonRTin[count], RStatObj)) Variance );
OUTPUT("SD of delay to cross  ", ASK(GETMONITOR(tdnonRTin[count], RStatObj)) StdDev);
OUTPUT();
ASK usersite[count] {for} SiteStats;
OUTPUT("**************************");
END IF; {if count = 1, 2 or 4}
END FOR; {for each usersite}

{RESETTING STATS}
FOR count := 1 TO numberofsites
        ASK(GETMONITOR(tdRTout[count], RStatObj)) TO Reset;
        ASK(GETMONITOR(tdnonRTout[count], RStatObj)) TO Reset;
        ASK(GETMONITOR(tdRTin[count], RStatObj)) TO Reset;
        ASK(GETMONITOR(tdnonRTin[count], RStatObj)) TO Reset;
        ASK(GETMONITOR(utaccesslink[count], ITimedStatObj)) TO Reset;
        ASK(GETMONITOR(tdspeech[count], RStatObj)) TO Reset;
        ASK(GETMONITOR(tdvideo[count], RStatObj)) TO Reset;
        ASK(GETMONITOR(tddata[count], RStatObj)) TO Reset;
        ASK(GETMONITOR(burstStats[count], IStatObj))TO Reset;
        ASK(GETMONITOR(bursttimeStats[count], RStatObj)) TO Reset;
        ASK(GETMONITOR(callStats[count], RStatObj)) TO Reset;
END FOR;
        received1speech := 0;
        received1video := 0;                {received cells from 1 - destination 3 only}
        received1data1 := 0;
        received1data2 := 0;
        received1data3 := 0;
        received1data4 := 0;
        received1data5 := 0;
        received1data6 := 0;
        received2speech := 0;
        received2video := 0;                {received cells from 2 - destination 3 only}
        received2data1 := 0;
        received2data2 := 0;
        received2data3 := 0;
        received2data4 := 0;
        received2data5 := 0;
        received2data6 := 0;
        received4speech := 0;
```

```
        received4video := 0;                {received cells from 4 - destination 3 only}
        received4data1 := 0;
        received4data2 := 0;
        received4data3 := 0;
        received4data4 := 0;
        received4data5 := 0;
        received4data6 := 0;
        receive1speech := 0;
        receive1video := 0;                 {received cells from 1 - all destinations}
        receive1data1 := 0;
        receive1data2 := 0;
        receive1data3 := 0;
        receive1data4 := 0;
        receive1data5 := 0;
        receive1data6 := 0;
        receive2speech := 0;
        receive2video := 0;                 {received cells from 2 - all destinations}
        receive2data1 := 0;
        receive2data2 := 0;
        receive2data3 := 0;
        receive2data4 := 0;
        receive2data5 := 0;
        receive2data6 := 0;
        receive4speech := 0;
        receive4video := 0;                 {received cells from 4 - all destinations}
        receive4data1 := 0;
        receive4data2 := 0;
        receive4data3 := 0;
        receive4data4 := 0;
        receive4data5 := 0;
        receive4data6 := 0;
END METHOD; {usersiteStats}
   {--------------------}
END OBJECT; {usersitemanagerObj}
{------------end Object-----------------------------------------------------}
{*******************end user site *************}
END MODULE. {Implementation module for usersite}
```

# Appendix III – Conference Papers associated with this work

# Source Modelling for B-ISDN Networks with ATM Switching

DIANE GAN and SATI MCKENZIE

School of Computing and Information Technology,
Greenwich University, Wellington Street, London SE18 6PF.

## 1. Introduction

B-ISDN/ATM networks are designed to transport different traffic types - digitised speech, real-time video, graphics images and 'traditional' data such as file transfers and transaction processing. These have very different arrival characteristics (rates, burstiness) and service requirements (loss tolerance, acceptable delay & delay variation). Data traffic can generally be described by Poisson arrivals and exponential (or general) service time distributions. Video traffic tends to be bursty, the bursts corresponding to sudden scene changes. Voice traffic consists of calls with alternating talkspurts and silences. Data is loss sensitive but can tolerate queuing delay. Video and voice require real time delivery but can tolerate a small amount of loss. Multimedia traffic may require, in addition, synchronisation of say sound and video.

Typically, all these sources will be present at a single user site. Traffic generated must be formatted into fixed length ATM cells and multiplexed into a single stream for transmission over the local access link to the first ATM switching node. The cells are then routed across the network to the destination site(s). A connection mode service is envisaged. Traffic parameters, bandwidth requirements (peak, mean) and quality of service (cell loss, cell delay variation) requirements are negotiated at connection setup.

Use of a high speed (Gbps) integrated network for multimedia traffic poses a number of interesting design questions.

The first relates to traffic arrival rates. Cells entering the network at an ATM node are made up of a superimposition of traffic from several subscriber sites, each with multiple active sources with widely differing arrival patterns. The resulting cell stream will not in general correspond to a simple Poisson process. Accurate source modelling is required for reliable performance evaluation.

The second relates to quality of service (QOS) provision, measured by cell loss probability and cell delay variation, to different users. Ensuring the negotiated QOS requires efficient congestion control and bandwidth allocation. Congestion control needs to be preventive rather than reactive, usually by restricting connection access dynamically, depending on the state of the network. Bandwidth allocation has to be made on the basis of peak rate requirements rather than mean rates, and some means of bandwidth enforcement has to be provided (e.g. leaky bucket).

The third relates to the design of the ATM switch. Given the high transmission speeds available, switching times become important in determining overall cell end-to-end delays. Several switch designs have been proposed, with different strategies for buffering and contention resolution. It would be interesting to compare them under realistic traffic loads.

The aim of this research is to develop a flexible simulation model to study the above questions. As a minimum, the system must include a small scale backbone ATM switching network (say five nodes) with associated subscriber sites each generating voice, video and data traffic which is then multiplexed and input to the backbone network. This should be

sufficient to analyse performance using different design options under varying loads.

The system model is being implemented using MODSIM - an object oriented simulation language from CACI. The relevant system components (sources, multiplexors, switches) are modelled as objects with precise interface definitions which are accessible to other objects and internal implementation details which are not. This has two advantages. Once the basic objects have been implemented and tested, the system can be built up incrementally by creating new object instances. Also, alternative designs (e.g. switching strategies) can be compared by changing the implementation details without affecting the interfaces with other objects.

The first phase of the project was to model the sources at a single subscriber site and the multiplexor and access link to the ATM node. This is described in Section 2. Preliminary simulation results are presented in Section 3. Section 4 summarises objectives for the next phase.

## 2 Simulation Model

The subscriber-site has three traffic sources connected to a multiplexor, which is in turn connected to an ATM switch through an access link. The source models generate speech, video and data traffic, respectively, which is queued at the multiplexor prior to being transmitted. The output link at the multiplexor operates in discrete slot intervals. A cell arriving when the queue is empty must wait until the start of the next slot boundary. The speed of the access link (typically 155.5 Mbps, Sonet OC-3) has been reduced to 20 Mb/s to allow reasonable loading of the access link with the traffic sources currently in place. Statistics are collected for each source type and for the queue length and utilization of the link from the multiplexor.

### 2.1 Voice Model

Voice is characterised by an ON-OFF speech model, representing alternating talkspurts and silences. Each state has an exponentially distributed time duration, and different papers give different values for the mean durations of these periods, [1],[2],[3],[4]. In this work, a talkspurt has been assigned a mean of 1.34 seconds and a silence 1.67 seconds. Telephone calls have a mean duration of 3.3 minutes and a mean inter-arrival time of 20 s. The time duration of each call and the time to the next call are exponentially distributed. Speech cells are generated only during a talkspurt, with a packetisation delay of 6 ms. Up to 40 simultaneous telephone calls are allowed, representing a small scale PBX. Presently every 10th cell is tagged as a low priority cell, representing periods of low speech intensity i.e. background noise during pauses in speech. Such low priority cells may be dropped in cases of network congestion.

### 2.2 Video Model

Video sources generate bursts of highly correlated cells, which are statistically different from a speech source. B-ISDN allows variable bit rate (VBR) traffic (bandwidth on demand), so that cells may be output on to the network as they are created, with no delays at the codec [6], [7]. The use of compression techniques for video coding means that only a change of scene or movement within a frame actually causes data to be output onto the network [5]. This results in a burst of cells being output to the network periodically.

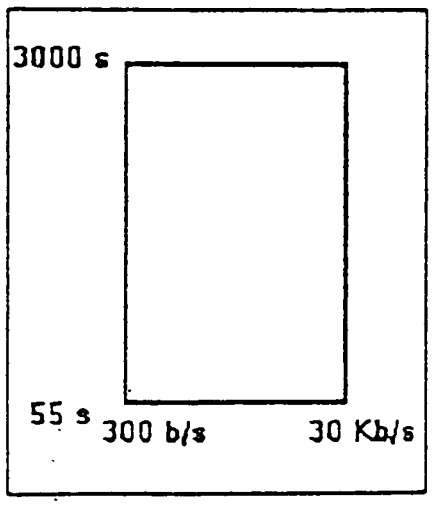
A burst of video data generates cells at a peak rate of, typically, 47 Kcells/s for the duration

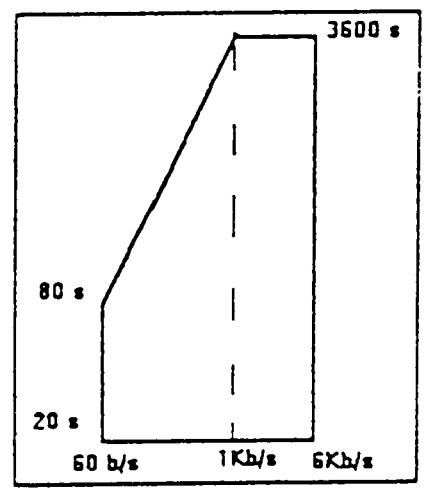
of a burst. Each burst has a variable inter-burst arrival time, which is exponentially distributed with a mean of 2 ms and the mean burst size is set at 10 cells. The peak rate is used to calculate the packetization delay, which is fixed for the duration of the burst. The peak cell rate is given by the ratio of peak bit rate to cell payload. To determine how frequently a video cell will occur in the output stream we calculate the ratio of link capacity to the peak rate. The packetization delay, based on the 155.52 Mb/s link capacity, is therefore eight times the slot duration, or 21.8 micro seconds [8]. For the current model, the peak rate is 15 Mb/s and hence in the scaled down link, three out of four cells may be occupied by a video cell.

Over the whole cycle (idle time between bursts + burst), an *average* rate is obtained which varies from burst to burst. Each such rate could be modelled as a *state* of the system, with a corresponding average bit rate as seen by the network. If the burstiness factor of 4 is assumed, then the mean rate for the video source is 3.75 Mb/s.

## 2.3 Data Model

Data traffic requires accurate delivery of cells, and queuing delays are less important [5]. Examples of data types are telemetry, teletext, voiceband data, facsimile and transaction timesharing. Each of these types is characterised by a range of allowed bit rates and a range of message durations [10]. A time duration and a bit rate are chosen at random within the appropriate limits. Figure 1 shows the bit rates and time durations limits for voiceband data, while Figures 2 and 3 correspond to those for transaction time sharing and facsimile, respectively.



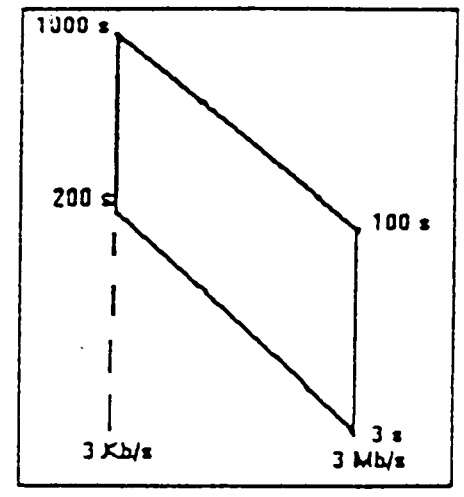




Fig.1 Voiceband Data

Fig.2 Transaction Time-Sharing

Fig.3 Facsimile

The size of the data message is given by bit-rate x time-duration. The number of ATM cells can then be determined as the ratio of message size in bits to ATM payload size. The packetization delay is obtained from the time duration divided by the number of ATM cells. This provides a steady flow of cells onto the multiplexor queue for the time duration selected. Statistics are collected for the different data models separately, as their characteristics are very different.

## 2.4 Multiplexor

Access to the network is currently regulated by a simple admission control scheme. The total buffer space is divided into 4 parts - a reserved allocation for each source and one shared area. A threshold is also defined, restricting access to the shared buffer. Each source must request permission to access the network and is always allowed access if there is space in its own area. Data sources are blocked if that is not the case. Access may still be allowed for speech calls and video bursts, provided the amount of unused shared buffer is below the

threshold [11]. For a speech call using the shared buffer space, all low priority cells are dropped. In the case of video, insufficient buffer space results in a lost burst, while for data and speech the whole message or call is blocked. It is assumed that blocked calls will be tried again and that blocked data messages will be retransmitted by a higher protocol layer.

When a cell has been accepted into the buffer a first-come-first-served policy applies and all cells are treated identically. As explained in section 1, the access control scheme adopted here may be easily modified, and it is intended that other methods will be explored in due course.

## 3 Simulation Results

The capacity of the link was scaled down to 20 Mb/s to ensure that the multiplexor could be heavily loaded in a reasonable simulation time. A maximum of 40 voice calls are allowed and this is kept constant for all runs. The data model currently generates 5 different data types, as described in section 2.3, and these are selected at random. The maximum number of active data sources is restricted and the number is varied for different simulation runs. It was found that the number of active data sources tended to the maximum, but it was the combination of the different data types which was the limiting factor, rather than the total number.

The simulation was run with the video source scaled down in the same proportion as the link. Peak rate video traffic occupies approx. 10% of the 155.52Mb/s link, so by scaling down the video source on the 20 Mb/s link, the same proportion of capacity, for video traffic, was maintained. It was found that the queue was well behaved and no cells were lost, by any source, due to blocking at the multiplexor. Cell delays were fairly constant for all traffic types.

- Table I - Utilization of the link (ut) and mean cell delay for all cells

| Max Time | Scaled down video source (10 data sources) ut | Mean cell delay (s) | Scaled down video source (15 data sources) ut | Mean cell delay (s) | 15 Mb/s video source (10 data sources) ut | Mean cell delay (s) | 15 Mb/s video source (15 data sources) ut | Mean cell delay (s) |
|---|---|---|---|---|---|---|---|---|
| 1000 | .0082 | $32\times10^{-6}$ | .0084 | $32\times10^{-6}$ | .158 | $1.2\times10^{-3}$ | .172 | $9.6\times10^{-3}$ |
| 2000 | .0089 | $32\times10^{-6}$ | .0091 | $33\times10^{-6}$ | .118 | $35\times10^{-6}$ | .136 | $42\times10^{-6}$ |
| 3000 | .0080 | $32\times10^{-6}$ | .0081 | $32\times10^{-6}$ | .117 | $35\times10^{-6}$ | .126 | $39\times10^{-6}$ |
| 4000 | .0072 | $32\times10^{-6}$ | .0075 | $32\times10^{-6}$ | .12 | $35\times10^{-6}$ | .133 | $39\times10^{-6}$ |
| 5000 | .0072 | $32\times10^{-6}$ | .0073 | $32\times10^{-6}$ | .165 | $25.5\times10^{-3}$ | .129 | $39\times10^{-6}$ |

To increase the load on the link, the video source packetization delay was assumed as for the 155.52 Mb/s link. This was equivalent to having several video sources on the standard capacity link. Since the video source has a peak rate of 15 Mb/s, when a burst of video cells arrives at the multiplexor, 3/4 of the link capacity is utilized for the duration of the burst. This only presents a problem when the combined speech and data traffic exceeds the remaining capacity. The queue length increases and cell delays become unacceptable for real-time traffic.

Table II - Details of Queue Lengths for Different Loads

| Max | Scaled down video source (10 data sources) | | | (15 data sources) | | | 15 Mb/s video source (10 data sources) | | | (15 data sources) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time | Max | Mean | STD | Max | Mean | STD | Max | Mean | STD | Max | Mean | SID |
| 1000 | 4 | 0.043 | .21 | 5 | .044 | .21 | 2425 | 14.2 | 146.9 | 257897 | 2256 | - |
| 2000 | 4 | 0.045 | .21 | 4 | .049 | .22 | " | 7.1 | 104.1 | " | 1128 | 1079 |
| 3000 | 5 | 0.42 | .21 | 4 | .043 | .21 | " | 4.7 | 85.1 | " | 752 | 1030 |
| 4000 | 5 | 0.037 | .19 | 4 | .039 | .20 | " | 3.5 | 73.7 | " | 564 | 949 |
| 5000 | 5 | 0.041 | .20 | 4 | .037 | .19 | 76952 | 42.5 | 526.5 | " | 451 | 878 |

Table II indicates that for the heavily loaded link (15 Mb/s video source, with both 10 and 15 data sources), the queue can become very large, even though the average utilization for the same period is still quite low ( see Table I). For a Poisson arrival stream, utilizations of 70% can be achieved. Similar utilizations with bursty sources can lead to queues overflowing, since queues at the multiplexor are very sensitive to momentary fluctuations in the arrival rates of the various sources.

Table III - Numbers of cells of each type (Mcell) served by multiplexor - cumulative

| Max | Scaled down video source (10 data sources) | | | (15 data sources) | | | 15 Mb/s video source (10 data sources) | | | (15 data sources) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time (sec) | M cells (cum.) Video | Speech | Data | M cells (cum.) Video | Speech | Data | M cells (cum.) Video | Speech | Data | M cells (cum.) Video | Speech | Data |
| 1000 | 2.78 | 0.84 | 0.26 | 2.78 | 0.82 | 0.36 | 4.67 | 0.67 | 2.13 | 4.46 | 0.65 | 5.95 |
| 2000 | 5.55 | 1.87 | 0.66 | 5.55 | 1.81 | 0.95 | 9.37 | 1.45 | 2.22 | 9.16 | 1.42 | 6.67 |
| 3000 | 8.33 | 2.70 | 0.82 | 8.33 | 2.65 | 1.11 | 14.09 | 2.17 | 2.30 | 13.88 | 2.14 | 7.60 |
| 4000 | 11.10 | 3.30 | 0.86 | 11.11 | 3.21 | 1.34 | 18.78 | 3.05 | 2.40 | 18.57 | 3.02 | 8.20 |
| 5000 | 13.88 | 3.87 | 0.90 | 13.88 | 3.79 | 1.40 | 23.45 | 3.75 | 4.80 | 23.28 | 3.73 | 8.86 |

The cumulative total of cells, for each source type, is shown in Table III. The usage by data on the scaled down link is quite small in comparison with the 15 Mb/s video results. Speech and video are penalised as the data gets more of the capacity.

Table IV - Percentage Occupancy of the Buffer by Source Type

| | Scaled down video source (10 data sources) | | | (15 data sources) | | | 15 Mb/s video source (10 data sources) | | | (15 data sources) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Time (sec) | Percentage Video | Speech | Data | Percentage Video | Speech | Data | Percentage Video | Speech | Data | Percentage Video | Speech | Data |
| 1000 | 71.6 | 21.7 | 6.7 | 70.3 | 20.7 | 9.0 | 62.5 | 9.0 | 28.5 | 40.6 | 5.9 | 53.5 |
| 2000 | 68.6 | 23.1 | 8.2 | 67.1 | 21.9 | 11.1 | 71.9 | 11.1 | 17.0 | 53.6 | 8.3 | 38.1 |
| 3000 | 70.2 | 22.9 | 6.9 | 68.9 | 21.9 | 9.2 | 75.9 | 11.7 | 12.4 | 59.5 | 9.2 | 31.3 |
| 4000 | 72.8 | 21.5 | 5.6 | 71.0 | 20.5 | 8.5 | 77.6 | 12.6 | 9.9 | 62.9 | 10.2 | 26.8 |
| 5000 | 74.5 | 20.7 | 4.8 | 72.8 | 19.9 | 7.3 | 73.3 | 11.7 | 15.0 | 65.6 | 10.5 | 23.9 |

As the percentage occupancy by the data traffic increases, for the 15 Mb/s video source,

speech and video occupancy of the buffer is forced down. Times of highest data occupancy correspond to the largest numbers of blocked cells.

The preliminary results indicate control of the data access to the line requires stricter control to prevent degradation of the quality of service to the real-time traffic.

## 4 Future Work

The models have been checked for internal consistency. The source models have been validated against previous results from other papers.

The next phase is to model the ATM switch and create a small scale (5-node) backbone network with associated subscriber sites. Realistic traffic can then be sent across the network and performance evaluated. The emphasis will be on end-to-end delays at cell, message, call and burst level, cell loss rates for individual connections and cell delay variation. Throughput and overall performance of individual switches and the whole network can also be studied.

The following questions will be addressed:

(1) Existing work on cell loss probability treats the access link as a whole, rather than individual connections. This is important to assess performance as seen by the end user.

(2) Comparison of contention strategies within switches. Existing work on switch performance is based on single switches. There is a need to consider more general topologies.

(3) Congestion measured through queue length is not valid for ATM networks, as the traffic is bursty and can cause rapid, temporary rises in queue length causing congestion to be wrongly detected. Alternative ways of measuring and controlling congestion are needed.

## References


[1] Brady P.T, "A Statistical Analysis of ON-OFF Patterns in 16 Conversations ", The Bell System Technical Journal, Jan 1968, pp 73

[2] Brady P.T., " A Model for Generating ON-OFF Speech Patterns in 2-way Conversation", The Bell System Technical Journal, Sept 1969, pp 2445

[3] Brady P.T., "A Technique for Investigating ON-OFF Patterns of Speech", The Bell System Technical Journal, Jan 1965, pp 1

[4] Gruber J.G., "A Comparison of Measured and Calculated Speech Temporal Parameters Relevant to Speech Activity Detection", IEEE Trans on Comms. Vol. Com-30, No. 4 April 1982, pp 728

[5] Bae J.J., "Survey of Traffic Control Schemes and Protocols in ATM Networks", IEEE Vol. 79, No. 2, Feb 1991,pp 170

[6] Nomura M., Fujii T., Ohta N., "Basic Characteristics of Variable Rate Video Coding in an ATM Environment", IEEE Journal on Selected Areas in Comms. Vol. 7, No.5, June 1989, pp 752

[7] Zhang Y.Q., Wu W.W., Kim K.S., Pickholtz R.L., Ramasastry J., "Variable Bit-rate Video Transmission in the Broadband ISDN Environment", Proc. of the IEEE, Vol.79, No. 2, Feb 1991, pp 214

[8] Odin A., Cosma J.P., "The Impact of Time Series and Counting Process Approaches to Video Source Modelling", Teletraffic Symposium 1992

[9] Handel R., Huber M.N., " Integrated Broadband Networks, An Introduction to ATM-based Networks", ISBN 0-201-54444-X, p52-58

[10] Weinstein S.B., "Telecommunications in the coming decades", IEEE Spectrum, Nov. 1987 pp62

[11] Kim B.G., Towsley D., "Dynamic Flow Control Protocols for Packet-Switching Multiplexers Servicing Real-Time Multipacket Messages", IEEE Trans. Comm. COM-34 (4) p348-356

# PERFORMANCE OF AN ATM NETWORK WITH MULTIMEDIA TRAFFIC - A SIMULATION STUDY

D E Gan, S McKenzie

University of Greenwich, UK

**Abstract.** Asynchronous Transfer Mode (ATM) allows real-time (RT) and non-real-time (non-RT) traffic to share the same network. These traffic types have different characteristics and requirements. This paper investigates different network access policies for an ATM network with heterogeneous traffic. The performance of a single queue model is examined and compared to the performance of a dual queue model, with different priorities of service. The goal is to provide flexible bandwidth allocation to benefit RT traffic without reducing the level of service to non-RT traffic significantly.

## INTRODUCTION

The introduction of Broadband Integrated Services Digital Networks (B-ISDN) will support the integration of such diverse services as data, voice and video over a single network. Asynchronous Transfer Mode (ATM) is the recommended multiplexing and switching technique for these new multi-service networks. ATM is a connection orientated, high-speed packet switching technique with bit rates of the order of 155 - 622 Mb/s, and small fixed length packets called cells. A multimedia call may require audio, data, still images or full motion video, or any combination of these, to be transported. Hence, multimedia applications e.g. video conferencing, must support the broad range of bit rates demanded by connections. A company based in London recently launched the UK's first public multimedia service. However, a new study recently revealed that video will be the most important driving force in the development of ATM networks. This will include compound documents containing text, spreadsheets, graphics files and scanned images.

A network transporting multimedia traffic must support connections with different delay requirements. For example, real-time traffic (RT) has stringent delay and loss requirements. The loss of a single video cell can cause a video stream to become de-synchronised and result in many subsequent video cells being discarded on arrival at the destination. Non-real-time traffic (non-RT), such as data retrieval requires guaranteed accurate delivery, but is less sensitive to delays. Multimedia traffic has additional requirements. For example, to achieve good lip synchronisation during video conferencing, the delay difference between the audio and video components must be less than 100 ms, Key (1).

International Broadcasting Convention, 14–18 September 1995
Conference Publication No. 413, © IEE 1995.

ATM networks must be able to transport the various types of traffic, each with different characteristics, while maintaining the quality of service (QoS) required by each user. The interaction of these streams of traffic at multiplexers and within switches can have a significant impact on the overall performance. The aim of this simulation is to investigate an optimal way of sharing resources (eg bandwidth), so as to minimise cell loss for all users and reduce delays for RT cells. This is done by developing realistic models for voice, video and some types of data, and using them to simulate the flow of traffic from a typical user-site across a small scale ATM network. All traffic is packetised into ATM cells and queued at an ATM multiplexer, prior to being passed to the first ATM switch. Cells are transported via virtual channel connections, using a simplified, dynamic addressing scheme. The ATM switches are modelled as self-routing, 4x4 with output buffering.

Details of the simulation model are discussed in the next section. Simulation results are also presented, and analysed in the subsequent sections. Conclusions and further work are detailed in the final section.

## THE SIMULATION MODEL

The simulation model has been implemented using the language MODSIM (CACI Inc.). MODSIM is an object orientated, event-driven simulation language which runs on a Sun work-station. System components (traffic sources, multiplexers, switches) are modelled as objects, with precise interface definitions.

### The User-Site

Voice, video and data traffic are generated at a typical user-site and are queued at a multiplexer prior to being transmitted to the first ATM switch in the network. ATM cells are passed across the network to the destination user-site. Statistics are collected at each user-site for the different types of traffic generated and received. End-to-end delays are also collected at the cell and message level.

**The Speech Model.** Speech is the least sensitive of the real-time services, to delays and cell loss. During two-way

speech, delays of up to 250 ms and cell losses as high as 1 in $10^4$ are tolerable. Speech is characterised by an ON-OFF model, which represents talkspurts and silences. Speech is generated at the rate of 64 Kb/s during the talkspurt periods. The gaps and short spurts that represent hesitations and pauses during normal conversation are assumed to be bridged by the speech coder using either fill-in or hang-over, creating the longer talkspurt and silence periods as discussed in Brady (2).

Each user-site has 40 phone lines associated with it, as could be expected in a small scale PBX (Private Branch eXchange). Phone calls are generated with a small inter-arrival time, as would be appropriate in a busy local exchange. Phone calls are only allowed if there is a vacant line in the local PBX and at the remote user-site PBX. During a silence period, the remote speech source generates ATM speech cells, corresponding to a two-way conversation (alternating talkspurts). All generated speech is packetised into ATM cells, with a packetisation delay of 6ms. The cells are queued at the multiplexer prior to being passed to the first ATM switch in the network.

**The Video Model.** The video model generates bursts of highly correlated cells, which are statistically different from speech cells. The characteristics of video depend on the coding scheme used, but typically video must be transmitted with minimal cell loss. The codec reads received video data into a buffer to smooth out any cell delay variation caused by the network. A codec can also take between 90 - 259 ms to decode the signal (1). An overall delay of up to 1 second is tolerable, but if the video is two-way e.g. video-conferencing, then the tolerances are the same as for speech.

The model assumes the variable bit rate (VBR) coding standard, MPEG, is being used to code the video. Bursts of video cells are generated with an exponential distribution and a mean size of 500 cells. The inter-arrival rate has an exponential distribution with a mean of half the video refresh rate (25 frames per second). The burstiness of video can be defined as the Peak rate / mean rate. Each burst has a burstiness factor (1-8), randomly selected, Izquierdo and Reeves, (3). This number is then used to determine the current rate for the burst.

Current video rate = Peak rate / burstiness factor

Video cells are then output to the multiplexer with a packetisation delay corresponding to the current rate. If the burstiness is 1 then video cells are output at the peak rate. A burstiness factor of 1-8 gives an overall mean video rate, during the simulation, approximately one third of the peak rate specified.

**The Data Model.** Data is very sensitive to cell loss. If the loss was due to congestion at a node, then the problem could be exacerbated by retransmitting the data message. Those cells already received may be discarded, on arrival, by the higher layers of the protocol..

There are a number data types included in the data each with different characteristics. These are tele teletext, voiceband data, facsimile and transaction sharing. Each data type has a range of allowed bit rat time durations, Stallings (4). Up to 10 data sourc allowed at any one user-site, and the type is selec random. The bit rate and the time duration are selec random from the permitted ranges for the chosen dat The size of the data message is given by (bit rate x duration) and the number of ATM cells is determined ratio of message size, in bits, to ATM payload siz bytes). The ATM cells are then output to the multiple the appropriate rate for the selected time duration.

**The Batch Model.** A batch source is included to pr background traffic to a specified percentage of cap This proves useful in investigating how a few voice may be affected by a heavy surge of other types of t Increasing the voice traffic alone will not provid loading required, and would also involve an addi computational overhead. The batch size is exponer distributed, with a mean of 1000 cells. Each batch als an exponentially distributed inter-arrival time. Batch are output to the multiplexer in a continuous stream, i packetisation gap between adjacent cells.

**The Multiplexer.** Access to the ATM network is multiplexer at each user-site. The cells from the t sources are queued prior to being transmitted to the ATM switch in the network. Cells access the multip asynchronously, while the network and access links network operate in slotted time, with a slot represent cell transmission time.

Each multiplexer has an associated ATM switch transmits all outgoing cells to a specified port on that sv A cell at the head of a multiplexer queue is served start of the next slot boundary. Queues are served on in first out (FIFO) basis. Statistics are collected for the time to access the network (for all cells) from multiplexer. The multiplexer also has a separate quet cells received from the network. Statistics are collecte the time taken for cells to cross the network and be rec at the remote user-site.

## The ATM Network

The ATM network currently comprises two ATM swit connected by a high-speed link (155.52 Mb/s). The a links from the user-sites are also assumed to be of the link speed. Routing tables are maintained centrall accessed by all switches.

**ATM Switch.** Each ATM switch is modelled as a contentionless, self-routing, banyan type switch. Each switch also has 4 inputs and 4 outputs (4x4), with output queuing. A cell entering a switch at an input port is routed to the correct output port using the virtual channel identifier (VCI). The label in the VCI field of the ATM cell is changed as the cell passes through the switch. The cell is then queued at the output port prior to transmission to the next ATM switch, or destination multiplexer. Statistics on the utilization and queue length at each port are gathered.

## THE SIMULATION EXPERIMENTS

A series of experiments has been completed to determine the optimum method for RT and non-RT traffic to share resources (e.g. multiplexer buffers), while maintaining the QoS required by the various traffic sources. Initially, the simulation was run with a single queue accessing the network at each user-site and no restrictions on queue sizes or access to the ATM network. The length of the queues, and hence the delays caused, at the multiplexers, were observed under various loads, in order to assess buffer space requirements.

It has been suggested in literature that a multiplexer buffer size equivalent to the number of cells which could be served in 1 ms is a reasonable assumption , Haverkort et.al.(6). For a 155 Mbps link this would require a buffer size of 367 cells. However, other sources, Bonomi, et.al. (7), state that buffer sizes of >500 cells, when serving bursty traffic, is more reliable. Imposing buffer restrictions for the single queue model meant that RT cells arriving at a multiplexer may find a nearly full buffer and be excessively delayed or even discarded, with subsequent degradation of QoS.

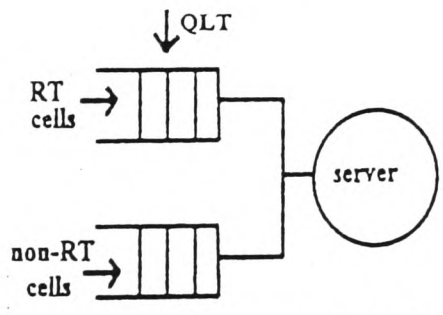



**Figure 1:** Cyclic Server with QLT = 2

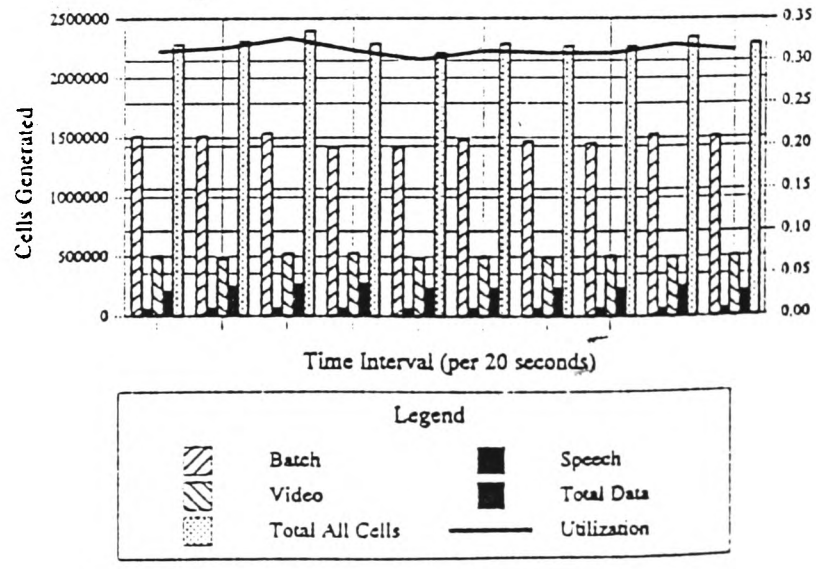
The multiplexer model was then changed to include separate queues for RT and non-RT traffic accessing the network. The queues were served alternately by a single (cyclic) server . A burst of RT cells (speech or video) arriving at the multiplexer would not be delayed by non-RT cells already queued, as is the case in the single queue model. This gives preferential treatment to the RT traffic, while still maintaining a level of service to the non-RT traffic. The dual queue model was again changed to give priority to RT cells. There are two possible strategies, priority service and exhaustive priority service, (6). In the case of priority service, if the RT queue was greater than a queue length threshold (QLT), then that queue was served exclusively until it was less than the threshold, see Fig For the exhaustive priority service, the RT queue is ass to be occupied by a burst of RT cells and the RT qu served until it is empty. In both cases, the server resumes alternating service between the RT and no queues. There is also the special case of priority service QLT = 0. This gives absolute non-pre-emptive prior the RT queue, i.e. on arrival at the multiplexer, a R will always be served before a non-RT cell.

The priority service was run with QLT = 2 and QLT The exhaustive non-pre-emptive priority service (exQ was also run with exQLT = 2 and exQLT = 4. The priority strategies and the alternating cyclic server compared to the single server case. These results analysed in the next section.

## SIMULATION RESULTS

All the simulations were run for 200 seconds using the s random seeds. This ensured that any variation in delays queue lengths are attributable to the access strategy a multiplexer, and not caused by any variation in the num of cells generated. Reports were generated at 20 sec intervals, and the statistics were reset after each report generated. The batch model was 20% of capacity and video model was 75 Mb/s peak rate. The mean numbe cells generated during the simulation are indicated in Fi 2. The mean utilization throughout the simulations approximately 31%, and this fluctuated by $\pm$1%.

Figure 2:Cells Generated vs Utilization




The delays caused by queuing at the multiplexer can l significant contribution to the overall delays encountere cells. Multiplexing a large number of bursty sources lead to long queues building up and subsequent cell los which is undesirable for RT traffic.

A single server at a multiplexer, used by a high propor of RT traffic, can be a bottleneck and cause serious del

which could affect the QoS for RT traffic. Queuing the RT traffic in a separate queue, reduces the delays experienced by RT cells significantly. The cyclic server (alternating between RT and non-RT queues), has hardly any impact on the non-RT cells, which experience the same delays as encountered in the single queue case. Figures 3 and 4 show the queue lengths at the multiplexer for RT and non-RT cells compared to the single queue for all cells, respectively.

Figure 3: Multiplexer Mean Queue

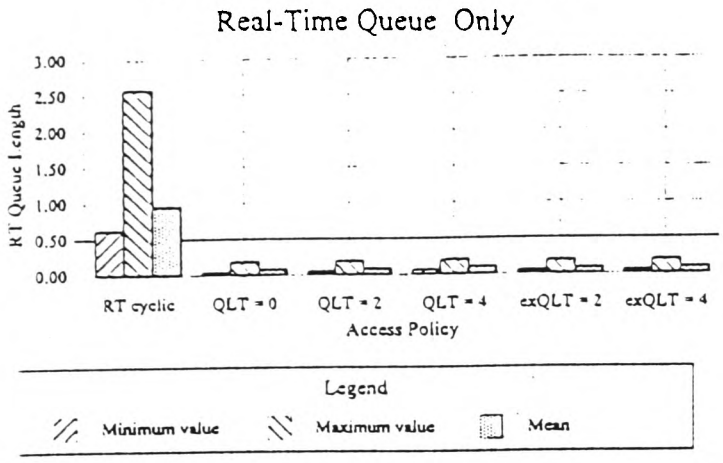



Figure 3 shows that the mean queue length for RT traffic is reduced when the RT cells are queued separately. The single server case (all cells) has a mean queue size of 94 cells. The mean queue size is reduced to less than 1 for the cyclic server, and to less than 0.1 for the other priority servers.

Figure 4: Multiplexer Mean Queue

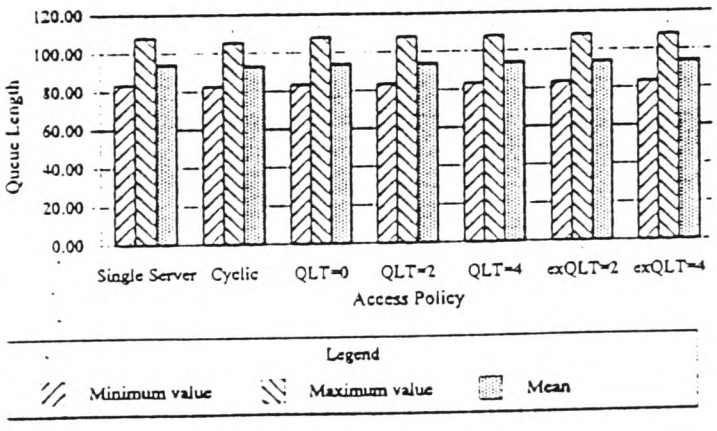
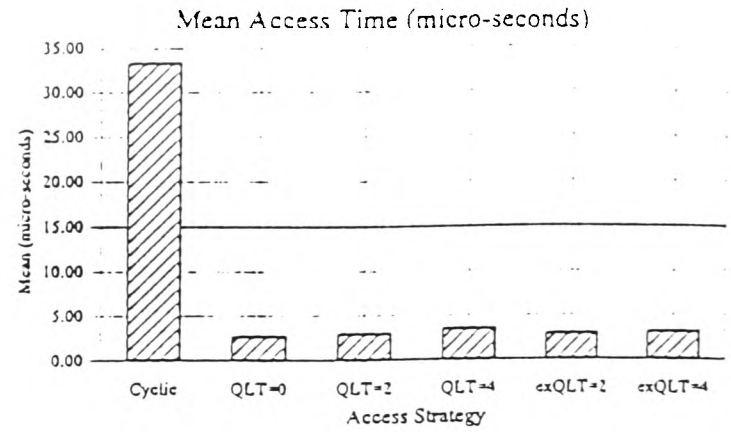



Figure 4 for non-RT traffic, shows that the mean queue length during the simulation, was 94.02 cells for the single server, 93 cells for the cyclic server and 93.9 cells for the other queues, regardless of which priority service strategy is used. This means that separate service for the RT cells has little impact on the queue length for non-RT cells, but significant improvement in the service to RT cells.

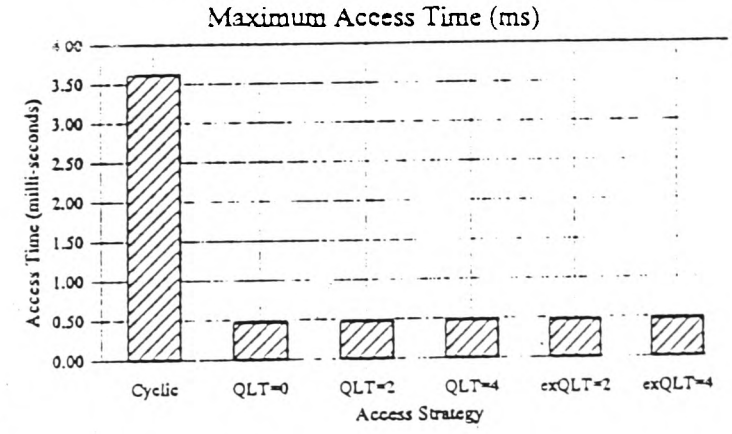
The time that a cell spends waiting in the multiplexer queue, prior to being served, is called the access time. Figure 5 shows the mean access times, for RT cells and is displayed in micro-seconds. Figure 6 illustrates the maximum access times, also for RT cells in milli-seconds. In both cases, (Figures 5 and 6) these are averages taken over the whole simulation.

Figure 5: Access Time for RT Cells




The mean access time using the single serve approximately 0.8ms. This is reduced to 33.3$\mu$s, for the traffic, using the cyclic server (Figure 5). The access tim further reduced for the priority strategies to a mean of 2. for QLT = 2 and exQLT = 2, 3.5$\mu$s for QLT = 4, and 3. using exQLT = 4. For the absolute non-pre-emptive (QLT = 0) there is further improvement to 2.7$\mu$s. The m access time for the priority queues (QLT and exQLT) considerably improved, compared to the single server c

Figure 6: Access Time for RT Cells



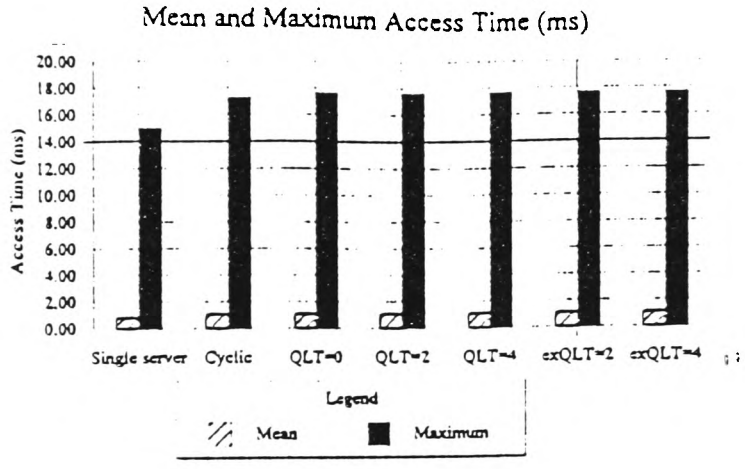

The maximum access time over the simulation, using single server, is 15ms, for all cells, and this includes cells. The cyclic server (Figure 6) has a maximum ac time, for RT cells, of 3.6ms during the simulation. Howe both QLT and exQLT have considerably better maxir access times for RT cells. The worst case maximum ac time for all QLT strategies is 1.46ms. The sma maximum access time is 54$\mu$s using QLT = 0. For QI 2 and exQLT = 2 it is 56$\mu$s, and for QLT = 4 this ris 65$\mu$s, while for exQLT = 4 it is 60$\mu$s. The mean fc priority strategies is 0.47ms. This is expected, since a l queue length threshold would begin giving priority se to RT cells sooner, and this is particularly true whe QLT = 0. This is also characteristic of RT traffic whic give rise to large fluctuations in the numbers of presented at a multiplexer.

Figure 7 shows the mean and maximum access time for

RT cells, in milli-seconds. The single server gives a mean access time over the whole simulation of 0.8ms. All other cyclic server strategies show a mean access time of approximately 1ms, for non-RT cells. Although this is slightly worse than the single server case for all cells, the delay is not excessive for non-RT cells.

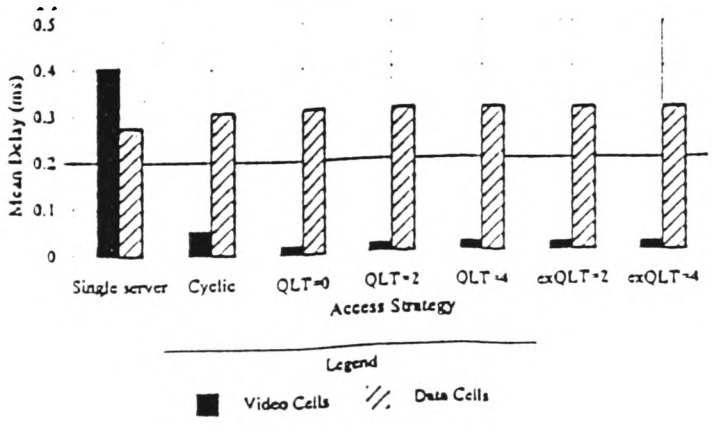
The maximum access times, in milli-seconds, for non-RT cells, are also shown in Figure 7. In this case, the single server queue has slightly smaller delays (15ms), than the other service strategies. The maximum access delay for the cyclic server has a mean of 17.2ms, while for the other strategies this is slightly worse at 17.6ms. For all the cyclic strategies the minimum and maximum range fluctuates between 13.8ms and 24ms, while the single server range is 12ms to 22ms The priority servers are marginally worse than the cyclic server, but still within acceptable limits for data traffic.

Figure 7: Access Time for non-RT Cells



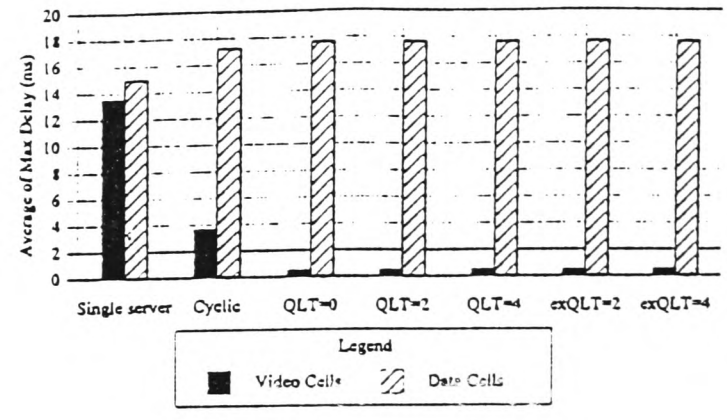

A single cell has a minimum end-to-end delay to cross the network of 13.6$\mu$s, with no delays caused by queuing. These are transmission delays and the delays imposed to traverse ATM switches, etc.. Figure 8 shows the mean end-to-end delays, in milli-seconds, experienced by video cells compared to that for data cells.

Figure 8: Mean End-to-End Delays




The mean end-to-end delays as seen by video cells are 405$\mu$s for the single server, and 52.8$\mu$s using the cyclic server. The priority service strategies reduce the mean delay for video cells to 19$\mu$s for QLT = 2 and exQLT = 2 and 19.6$\mu$s and 19.2$\mu$s for QLT = 4 and exQLT = 4, respectively. The mean access delay over the v simulations is approximately 19$\mu$s for all the pr strategies, except QLT = 0, which shows a s improvement, with a mean delay of 18.7$\mu$s. The QL exQLT strategies also benefit speech cells, w experience similar improvements in end-to-end delays The maximum delays experienced by video and data are shown in Figure 9, in milli-seconds. Video experience a maximum delay of 13.5ms, when usin single server. The cyclic server improves on this, w maximum delay of 3.6ms. However, the maximum d are considerably reduced with QLT = 0, to 0.484ms, = 2 and exQLT = 2 are only marginally worse with a r value of 0.486ms. For QLT = 4 the maximum del 0.492ms, and for exQLT = 4 this is 0.490ms, but this is considerably better than the single server case.

Figure 9: Maximum End-to-End Delays




It can be seen that both the mean and maximum end-to-delays for data cells are slightly worse when RT cells queued separately, when compared to the single se case. The mean delay (Figure 8) for data cells is 27 using the single server. It is 306$\mu$s for the cyclic server approximately 311$\mu$s for the other priority servers.

The maximum delays data cells experience using any queue access method (Figure 9) are only marginally w than for the single server. This is 15ms for the single ser 17.2ms for the cyclic service and for all other ser strategies the maximum end-to-end delay is 17.6ms. choice of cyclic priority strategy used makes no impac the maximum end-to-end delays for data cells.

It has been found that QLT = 2 and exQLT = 2 give similar improvements in the access times and end-to-delays experienced by RT cells, with exQLT = 2 b marginally better than QLT = 2. QLT = 4 and exQLT are slightly worse, mainly due to the additional wait imp while the RT queue builds up to the threshold. As expe QLT = 0 gives a better service than the other pri strategies. This is because absolute non-pre-emptive ser is always given to RT cells and this is reflected in results. In all cases, cyclic service improves the perform of RT cells compared to a single queue.

## CONCLUSIONS

The aim of this paper has been to investigate various strategies for optimally sharing the bandwidth of an ATM access link between RT and non-RT traffic. The single queue model has been used as a baseline for comparison with other strategies. Separate queues for RT and non-RT traffic have been studied, with the same traffic load. The performance of the multiplexer and cell level statistics for the different traffic sources have been analysed.

The results show that separate queues for RT and non-RT traffic benefits the RT traffic, and this includes multimedia traffic. It has also been found that giving a biased service to the RT traffic, as in the case of the cyclic server, does not significantly affect the QoS for the non-RT traffic. However, giving a priority service to the RT queue, as in the case of the priority service and the exhaustive priority policies, has some effect on the non-RT traffic, but not very large. There is however, considerable improvement to the delays experienced by RT traffic.

The buffers at multiplexers serving RT traffic need to be carefully dimensioned to ensure that RT cells are not lost due to buffer overflow. Setting maximum buffer limits close to the mean burst size of a real-time source, means that a single burst of cells could fill the buffer and cause other traffic using that same link to have cells dropped. The main RT source in this simulation is the video model, and the mean burst size is 500 cells. The batch data model also has a mean burst size of 1000 cells, which could present problems to other users, and particularly RT traffic, trying to share the same buffer.

There is no congestion present within the ATM network, during the simulations, so any delays are caused by the queues at the multiplexer. If further delays were encountered, for example, due to congestion in the ATM switches, as could be the case at times of peak traffic flow, RT cells could suffer serious degradation in QoS. It is therefore reasonable to reduce delays to RT cells, if possible, to allow more flexibility when congestion is encountered. This would ensure that RT cells are delivered within the strict delay limits required.

## REFERENCES


1. Key, P., 1994. 2nd Performance Modelling & Evaluation of ATM Networks 1/1-28

2. Brady P.T.1965. The Bell Syst. Tech J. 44, 1-22

3. Izquierdo, M.R., Reeves, D. 1994.TR94-17mri Tech Rep CCSP

4. Stallings, W., 1992. "ISDN & Broadband ISDN", Maxwell MacMillan International Editions

5. Saito, H., 1993. "Teletraffic Technologies in Networks", Artech House, Boston

6. Haverkort B.R. Idzenga H.P. Kim B.G. 1994 Performance Modelling & Evaluation of ATM Net 17/1-12

7. Bonomi F., Montagna S., Paglino R., 1993. Com Networks & ISDN Systems 26 119-138

# References


[ABBAS92] M.Abbas, Z.A.Ahmad
Perfomance Evaluation of Selective Cell Discarding Control in ATM Networks
Singapore ICCS/ISITA Conference Vol.1 (1992) p142-146

[AMBR92] M.D'Ambrosio, R.Melen
Performance Analysis of ATM Switching Architectures: A Review
CSELT Technical Papers Vol.XX No.3 (June1992) p265-281

[ANAG91] M.E. Anagnostou, P.D.Sparaggis
Analysis of Asynchronous Multiplexers for Periodic Traffic in Integrated Broadband Networks
IEE Proceedings-1 Vol.138 No.6 (Dec 1991) p481-486

[ARN95] J. Arnold, GPT Ltd
ATM Networks Traffic Conditioning & Flow-control
3rd ATM Workshop on Performance Modelling, Tutorial Paper 1, (July 1995)

[ARV95] A.Arvidsson, V.Lind
On the Validity of Some Markovian Models in ATM Traffic Modelling
3rd ATM Workshop on Performance Modelling, Technical Paper 36, (July 1995)

[BADR94] H.F.Badran, H.T.Mouftah
ATM Switch Architectures with Input-Output-Buffering: Effect of Input Traffic Correlation, Contention Resolution Policies, Buffer Allocation Strategies and Delay in Backpressure Signal
Computer Networks and ISDN Systems 26 (1994) p1187-1213

[BAE91] J.J.Bae, T. Suda
Survey of Traffic Control Schemes and Protocols in ATM Networks
Proceedings of the IEEE Vol.79, No.2 (Feb 1991) p170-187

[BONO92] F.Bono, S.Montagna, R.Paglino
Performance Analysis of an ATM Statistical Multiplexer with Heterogeneous Bursty Traffic
11th Conference on Computer Communication, Genova, Italy Vol.2, (1992) p737-742

[BONO93] F.Bono, S.Montagna, R.Paglino
A Further Look at Statistical Multiplexing in ATM Networks
Computer Networks and ISDN Systems 26 (1993) p119-137

[BRAD65] P.T.Brady
A Technique for Investigating On-Off Patterns of Speech
The Bell System Technical Journal Vol.XLIV, No.1 (Jan 1965) p1-23

[BRAD68] P.T.Brady
A Statistical Analysis of On-Off Patterns in 16 Conversations
The Bell System Technical Journal (Jan 1968) p73-91

[BRAD69] P.T.Brady
A Model for Generating On-Off Speech Patterns in Two-Way Traffic
The Bell System Technical Journal (Sept 1969) p2445-2471

[BUTT91] M.Butto, E.Cavallero, A.Tonietti
Effectiveness of the Leaky Bucket Policing Mechanism in ATM Networks
IEEE Journal on Selected Areas in Communications Vol.9, No.3 (April 1991) p335-342

[CALL92] F.Callegati, C.Raffaelli
Performance Analysis of an ATM Multiplexer with Different Classes of Bursty Sources
Singapore ICCS/ISITA Conf. Vol.1 (1992) p167-172

[CAS94] O.Casals
Traffic Control Functions in ATM: Usage/Network Parameter Control and Traffic Shaping
2rd ATM Workshop on Performance Modelling, Tutorial Paper 4, (July 1994)

[CHAN94] E.Chan, J.M.Ng, T.Y.Liang, B.Ho, V.Lee
Effectiveness of LB as Source Policing Schemes for ATM N/Ws.
2rd ATM Workshop on Performance Modelling, Paper 56, (July 1994)

[CHANG94] C.J. Chang, P.C. Lin, J.M. Chen
Study on Optimal Queue-length-threshold Scheduling Policy for an ATM Multiplexer with Finite Buffers and Batch Poisson Arrivals
Computer Networks and ISDN Systems 26 (1994) p525-539

[CHEN94] L.Cheng, P.Yegani
Priority Scheduling in an ATM Network
Proceedings of 1994 Summer Computer Simulation Conference, ISBN 1-56555-029-3, (1994) p169-174

[CHIP89] R.Chipalkatti, J.F.Kurose, D.Towsley
Scheduling Policies for Real-time and Non-Real-time Traffic in a Statistical Mulitplexer
IEEE INFOCOM 89 8th Annual Joint Conf. Ottawa Cat No. 89CH2702-9 (1989) p774-783

[CHOW94] S.Chowbury, K.Sohraby
Bandwidth Allocation Algorithms for Packet Video in ATM N/Ws
Computer N/Ws and ISDN Systems 26 (1994) p1215-1223

[CLARK96] M.P. Clark
ATM Networks, Principles and Use
Pub Wiley-Teubner (1996) Chapter 8, ISBN 0-471-96701-7

[CONT95] M.Conti, E.Gregpori, M.Nava
Analysis and Modelling of MPEG-1 Video Sources
3rd ATM Workshop on Performance Modelling, Technical Paper 5, (July 1995)

[COSM94] J.P.Cosmas, G.H.Petit, T.Lehnert, C.Blondia, K.Kontovassilis, O.Casals, T.Theimer
A Review of Voice, Data and Video Traffic Models for ATM
ETT, Vol 5, No.2 (Mar - Apr. 1994) pp11/139

[DAG95] A. Dagiuklas & M. Ghanbari
Modelling of a Compatible H.261 2-layer Video Codec at the Cell Level
3rd ATM Workshop on Performance Modelling, Technical Paper 3, (July 1995)

[DAIL96] A. Dailianas, A.Bovopoulos
Real-Time Admission Control Algorithms with Delay and Loss Guarantees in ATM Networks
Computer Communications 19 (1996) p169-179

[DITTM91] L.Dittman, S.B.Jacobsen, K.Moth
Flow Enforcement Algorithms for ATM Networks
IEEE Journal on Selected Areas in Communications Vol.9, No.3 (April 1991) p343-350

[FOW91] H.J.Fowler, W.E.Leland
Local Area Network Traffic Characterisations, with Implications for Broadband Network Congstion Management
IEEE Journal on Selected Areas in Communications Vol.9 No.7 (Sept 1991) p1139-1149

[GALL89] G.Gallassi, G.Rigolio, L.Fratta
ATM: Bandwidth Assignment and Bandwidth Enforcement Policies
IEEE Globecom-89 Dallas Texas, (1989) p1788-1793

[GAN94] D.E.Gan, S.McKenzie
Source Modelling for B-ISDN Networks with ATM Switching
IEE Multimedia Communication Systems Colloquium (March 1994) Paper 8

[GAN95] D.E.Gan, S.McKenzie
Performance of an ATM Network with Multimedia Traffic - A Simulation Study
IEE International Broadcatsing Convention, (14-18 Sept. 1995), pub. No 413, p263-268

[GANO94] P.Ganos, M.Koulkias, G.Kokkinanis, S.Kotsopoulos
A Novel Dynamic Priority Scheduling Method for Mulitple Classes of ATM Traffic in an ATM Statistical Multiplexer
2rd ATM Workshop on Performance Modelling, Technical Paper 48, (July 1994)

[GERZ91] G.Y. Gerzon, L.F. Turner
An Overload Control Scheme for Time Division Multiplexed Packet Voice
Proceedings of 1991 Singapore International Conference on Networks (1991) p382-387

[GRAH96] G. Graham
ATM & Digital Video
1996 Digital Video Test Symposium - The Moving Image Society (Sept 1996) Paper 8 p13-19

[GRUB82] J.G. Gruber
A Comparision of Measured and Calculated Speech Temporal Parameters Relevant to Speech Activity Detection
IEEE Transactions on Communications, COM-30, No.4 (April 1982) p728-738

[GUN93] L.Gun, R.Guerin
Bandwidth Management & Congestion Control Framework of the Broadband N/W Architecture
Computer N/Ws and ISDN Systems 26 (1993) p61-78

[GUPT93] A.K.Gupta, L.O.Barbosa, N.D.Georganas
Switching Modules for ATM Switching Systems and their Interconnection Networks
Computer Networks and ISDN Systems 26 (1993) p433-445

[HABIB92] I.W.Habib, T.N.Saadawi
Access Flow Control Algorithms in Broadband Networks
Computer Communications Vol.15,No.5 (June 1992) p326-332

[HAL96] F. Halsall
Data Communication, Computer Networks and Open Systems 4th Ed
Addison Wesley ISBN 0-201-42293-X (1996) Chapter 10

[HART91] F.Hartanto, H.R.Sirisena, K.Pawlikowski, W.K.Kennedy
Performance Study of Dual Queues with Limited Cyclic Service in ATM Switching
Proc. of S'pore International Conference on Networks - (1991) p253-258

[HAV94] B.R.Haverkort, H.P.Idzenga, B.G. Kim
Performance Evaluation of ATM Cell Scheduling Policies using Stochastic Petri Nets
2rd ATM Workshop on Performance Modelling, Technical Paper 17, (July 1994)

[HLUC88] M.G.Hluchyj, M.J.Karol
Queueing in High-Performance Packet Switching
IEEE Journal on Selected Areas in Communications, Vol.6, No.9 (Dec 1988) p1587-1597

[HUAN96] C.H.Huang, R.Y.Lee
Achieving Multimedia Synchronisation Between Live Video and Live Audio Streams using QoS Controls
Computer Communications Vol.19 (1996) p456-467

[IZQUI94] M.R.Izquierdo, D.S.Reeves
Statistical Characterisation VBR MPEG at the Slice Layer
Techical Report - North Carolina State University, Raleigh, NC27695 (June 1994)

[IZQUI96] M.R.Izquierdo, D.S.Reeves
A Survey Of Source Models for Variable Bit Rate Encoded Video
Techical Report - North Carolina State University, Raleigh, NC27695 (Aug 29 1996)

[JAD95] T. M. Jadoon, D. A. Harle
On mean policing with bursty traffic specification and allocation (BTSA) policer function
3rd ATM Workshop on Performance Modelling, Technical Paper 23, (July 1995)

[JAIN96] R. Jain
Congestion Control & Traffic Management in ATM Networks: Recent Advances and a Survey
Ohio State University, (Aug 1996)

[KARA95] M.Kara, P.M.Dew
A Traffic Characterisation and General Architecture for Distributed Multimedia Applications
2nd Communications Symposium (July 1995) p154 -157

[KARL96] G.Karlsson
Capacity Reservation in ATM Networks
Computer Communications 19 (1996) p180-193

[KEY94] P. Key
An Introduction to ATM Performance Issues & Modelling
2rd ATM Workshop on Performance Modelling, Tutorial Paper 1, (July 1994)

[KIM92] Y.H. Kim, B.C. Shin, C.K.Un
Performance Analysis of Leaky Bucket Bandwidth Enforcement Strategy for Bursty Traffics in an ATM Network
Computer Networks and ISDN Systems 25 (1992) p295-303

[KIM96a] J.B.Kim, R.Simha, T.Suda
Analysis of a Finite Buffer Queue with Heterogeneous Markov Modulated Arrival Processes: a study of traffic burstiness and priority packet discarding
Computer Networks and ISDN Systems 28 (1996) p653-673

[KIM96b] Y.J.Kim. S.C.Chang, C.K.Un, B.C.Shin
UPC/NPC Algorithm for Guaranteed QoS in ATM Networks
Computer Communications 19 (1996) p216-225

[LAET95] G. De Laet, J.Naudts
A Scheme for Multiplexing ATM Sources
3rd ATM Workshop on Performance Modelling, Technical Paper 50, (July 1995)

[LI85] S.Q.Li, J.W.Mark
Performance of Voice/Data Integration on a TDM System
IEEE Transactions on Communications Vol.Com-33 No.12 (Dec 1985) p1265-1273

[LIEW90] S.C.Liew
Performance of Input Buffered and Output Buffered ATM Switches Under Bursty Traffic: Simulation Study Conference (1990) p1919-1925

[LODG92] N. Lodge
Video Compression Techniques
IEE Colloquium, Applications of Video Compression in Broadcasting, (Oct 1992)

[MAS96] L.G.Mason, A.Pelletier, J. Lapointe
Towards Optimal Policing in ATM Networks
Computer Communications 19 (1996) p194-204

[MIT94] N.M.Mitrou
Traffic Control & Congestion Control in ATM Networks: a unified view
2rd ATM Workshop on Performance Modelling, Tutorial Paper 5, (July 1994)

[MOLL94] A.M.Moller
Equivalence of Policing Mechanisms
Integrated Broadband Communication Networks and Services (1994) p253-259

[NIEST90] G. Niestegge
The "Leaky Bucket" Policing Method in the ATM (Asynchronous Transfer Mode) Network
International Journal of Digital and Analog Communications Systems Vol.3 (1990) p187-197

[NITTO92] E.Di Nitto, A.Iera, S.Marano
Avalanche Tagging: A New Proposal for Policing Function in ATM Environment
11th International Conference on Computer Communication Vol.2 Genova, Italy, (1992) p725-730

[ONV94] R.O.Onvural
Asynchronous Transfer Mode Networks – Performance Issues
Pub Artech House, p20, 21, Chapter 2

[ORS95] T.Ors, S.P.W.Jones
Performance Optimisation of ATM Input Control using Multiple Leaky-buckets
3rd ATM Workshop on Performance Modelling, Technical Paper 24, (July 1995)

[PATTA93] A.Pattavina
Nonblocking Architectures for ATM Switching
IEEE Communications Magazine (Feb 1993) p38-48

[PAX94] V.Paxson, S.Floyd
Wide-Area Traffic: The Failure of Poisson Modeling
Proceedings of SIGCOMM (1994) p257-268

[PAXS95] V.Paxson, S.Floyd
Wide-Area Traffic: The Failure of Poisson Modeling
(July 18 1995)

[PLATT94] A. Platt
Congestion Control on Demand
2nd ATM Workshop on Performance Modelling, Technical Paper 55, (July 1995)

[RAD96] S.Radhakrishnan, S.V.Raghavan, A.K.Agrawala
A Flexible Traffic Shaper for High Speed Networks: Design and Comparative Study with Leaky Bucket
Computer Networks and ISDN Systems 28 (1996) p453-469

[RAMA91] G.Ramathurthy, B.Sengupta
Delay Analysis of a Packet Voice Multiplexer by the ΣDi/D/1 Queue
IEEE Transactions on Communications, Vol.39, No.7 (July 1991) p1107-1114

[RASM91] C. Rasmussen, J.H.Sorensen, K.S. Kvols, S.B. Jacobsen
Source-Independent Call Acceptance Procedures in ATM Networks
IEEE Journal on Selected Areas in Communications Vol.9, No.3 (April 1991) p351..

[RATH91] E.P.Rathgeb
Modeling and Performance Comparison of Policing Mechanisms for ATM Networks
IEEE Journal on Selected Areas in Communications Vol.9, No.3 (April 1991) p325-334

[RILE95] M.J.Riley, B.U.Kohler, A.J.Miller, I.E.G.Richardson
Statistical Multiplexing MPEG Video Traffic with FEC
3rd ATM Workshop on Performance Modelling, Technical Paper 53, (July 1995)

[RUIU96] D. Ruiu
An Overview of MPEG
1996 Digital Video Test Symposium - The Moving Image Society (Sept 1996) Paper 9 p20-33

[RUIU96a] D. Ruiu
The Test Challenges of Compressed Digital Video
1996 Digital Video Test Symposium - The Moving Image Society (Sept 1996) Paper 10 p35-55

[SING90] S.Singhal, D.Le Gall, C-T Chen
Source Coding of Speech and Video Signals
Proceeding sof the IEEE, Vol.78, No.7 (July 1990) pp1233

[SKEL93] P.Skelly, M.Schwartz, S.Dixit
A Histogram-Based Model for Video Traffic Behaviour in an ATM Multiplexer
IEEE/ACM Transactions on Networking Vol.1, No.4 (Aug 1993) p446-458

[SMIT94] T.A. Smit
The Poor Gain from Statistically Multiplexing in the Homogenous and the Heterogenous Case
Integrated Broadband Communications Networks and Services (1994) p213-223

[SOLE94] J.Sole-Pareta, J.Domingo-Pascual
Burstiness Characterisation of ATM Cell Streams
Computer Networks and ISDN Systems 26 (1994) p1351-1363

[STAM94] G.D.Stamoulis, M.E.Anagnostou, A.D.Georgantas
Traffic Source Models for ATM Networks: A Survey
Computer Communications Vol.17 No.6 June (1994) p428-438

[SUDA89] T. Suda, T.T. Bradley
Packetized Voice/Data Integrated Transmission on a Token Passing Ring LAN
IEEE Transactions on Communications, Vol.37, No.3 (March 1989) p238-244

[TAN96] A.S. Tanenbaum
Computer Networks 3rd Ed
Prentice Hall ISBN 0-13-394248-1 Chapt.2

[TED93] T.E.Tedijanto, L.Gun
Effectiveness of Dynamic Bandwidth Management Mechanisms in ATM N/Ws
IEEE Computer Society Conf. 12th INFOCOM '93 Conference San Francisco p358-367

[TURN86] J.S.Turner
New Directions in Communications (or which way the information age?)
IEEE Communications 24 (10) (Oct 1986) p8-15

[TURN92] J.S. Turner
Managing Bandwidth in ATM Networks with Bursty Traffic
IEEE Network (Sept 1992)

[WAL93] K. van der Wal, M.Dirksen, D.Brandt
Implementation of a Police Criterion Calculator Based on the Leaky Bucket Algorithm
IEEE Globecom 93 (1993) p713-718

[WALL90] E.Wallmeier
A Connection Acceptance Algorithm for ATM Networks Based on Mean and Peak Bit Rates
International Journal of Digital and Analogue Communication Systems, Vol.3 (1990) p143-153

[WAN92] J.Wang, C.Wang, Y. Wang
Loss Perfomance Analysis of an ATM Mutliplexer
Singapore ICCS/ISITA Conf. Vol.1 (1992) p173-176

[WAN93] W.Wang, F.A.Tobagi
The Christmas Tree Switch: An Output Space-Division Fast Packet Switch Based on Interleaving Distribution and Concentration Functions
Computer Networks and ISDN Systems 25 (1993) p631-644

[WANG93] J.L.Wang, J.P.Zhou,C.Wang Y.H.Fan
Interdeparture Processes of Traffic from ATM Networks
IEEE Computer Society Conference (1993) p1337-1341

[WEINS87] S.B. Weinstein
Telecommunications in the coming decades
IEEE Spectrum, (Nov 1987), p62-67

[WENG91] C.M.Weng , J.J.Li
Solution for Packet Switching of B-ISDN
IEE Proceedings - 1 Vol.38, (Oct 1991) p394-400

[WIDJ93] I.Widjaja, A. Leon-Garcia
The Effect of Cut-through Switching on the Perfomance of Buffered Banyan Networks
Computer Networks and ISDN Systems 26 (1993) p139-159

[XING95] A.Xing, C.McCrosky
Switch Performance with Self-Similar Traffic
3rd ATM Workshop on Performance Modelling, Technical Paper 22, (July 1995)

[YAMA92] N.Yamanaka, Y. Sato, K. Sato
Performance Limitation of Leaky Bucket Algorithm for Usage Parameter Control and Bandwidth Allocation Methods
IEICE Trans. Comms, Vol.E75-B No.2 (Feb 1992) p82-86

[YEGE94] F.Yegenoglu, B. Jabbari
Characterisation and Modelling of Aggregate Traffic for Finite Buffer Statistical Multiplexers
Computer Networks and ISDN Systems 26 (1994) p1169-1185

[YIN90] N. Yin, S-Q. Li, T.E. Stern
Congestion Control for Packet Voice by Selective Packet Discarding
IEEE Transactions on Communications, Vol.38, No.5 (May 1990) p674-683