Concurrent Cell Rate Simulation of ATM Telecommunications Networks

by

Matthew Bocci

Submitted for the degree of Doctor of Philosophy

Department of Electronic Engineering

Queen Mary & Westfield College

University of London

February 1997

to Laurissa

Abstract

This thesis presents a new approach to the accelerated simulation of Asynchronous Transfer Mode (ATM) telecommunications networks. The union of a parallel computing architecture with cell rate traffic modelling is investigated. The results of this work are also applied to improving sequential cell rate simulators.

In order to obtain statistically significant network performance results, the flow of very large numbers of cells through the network must be modelled. This results in long simulation times. A number of techniques have been studied elsewhere for accelerating the simulation. Concurrent (parallel) simulation exploits the inherent parallelism in a telecommunications network. Cell rate modelling accelerates the simulation by reducing the total number of events that have to be processed. The use of cell rate modelling in parallel simulation could result in enhanced speedup over traditional cell level ATM simulators. Cell rate simulation imposes a new set of requirements on simulation platforms. It is the effect of these in both parallel and sequential computing environments that forms the focus of this research.

In this thesis, a study of the efficiency and accuracy of the Timestepping approach for parallel cell rate ATM simulation is presented. Experiments, based on an ATM simulator developed by the author, demonstrate that Timestepping is only effective if the network is heavily loaded. However, the distributed nature of the event list in the Timestepping scheme can be incorporated into sequential cell rate simulators with significant speed advantages. An experimental study of such an event list scheme demonstrates that maximum speedup is achieved when the event list is spatially decomposed and also when multiple simultaneous event generation during burst scale queueing is correctly handled. Therefore, in

order to efficiently support cell rate modelling an event list algorithm should cope with the spatially distributed nature of events and multiple simultaneous events.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to acknowledge the support of my supervisor Professor Laurie Cuthbert, and also that of Dr Eric Scharf and Dr Jonathan Pitts. I would also like to thank my parents for their encouragement.

1. Introduction

Asynchronous Transfer Mode (ATM) has been identified as the target transfer mode for future Broadband Integrated Services Digital Networks (B-ISDN). Traditionally, simulation techniques have been applied to the specification and design of ATM equipment, and to enable experience of ATM technology to be gained before the introduction of real ATM networks. More recently, the use of simulators in the development of network management systems has emerged.

Despite the fact that real networks are now in existence and experience of ATM is now more widespread, the justification for simulation is expected to remain because these networks are often unsuitable for experimentation. Furthermore, off-line modelling of these networks will be important even for network management systems that are connected to real networks in order to assess the impact of management decisions before they are made.

Conventional simulation of ATM involves the use of discrete event simulators that model the flow of each individual cell through the network (cell level simulation). However, whilst this approach can give very accurate results, statistical considerations mean that very large numbers of cells have to be simulated in order to guarantee the accuracy of those results. Simulators that simply model the set-up and clear-down of calls or connections in the network (call level simulation) can simulate at much faster rates than cell level simulators, but because none of the traffic is modelled during the duration of the call it is not possible to accurately measure many significant cell scale statistics (such as cell loss ratio). Faster simulators that can generate cell scale statistics are required, both to bring down simulation run times and to increase the scope for coupling simulators to telecommunications management networks (TMN) who's

perception of time may be closer to real time than the simulator's internal simulation time.

Accelerated simulation techniques aim to improve the speed of simulation and can be classified into three broad areas:

- implementation techniques
- statistical techniques
- modelling techniques.

This thesis considers the cell rate modelling technique, and in particular, the acceleration of simulators through the exploitation of concurrently occurring events. In particular, two possible paths for accelerating cell rate simulators are investigated: the implementation technique of parallel simulation, and the optimisation of event list management schemes to efficiently cope with the requirements of cell rate modelling.

Whilst cell rate modelling and parallel simulation can give significant speedup, they still run at much less than real time i.e. for a given real time duration as perceived by the user of the simulator, the simulator's internal clock will advance by a much smaller amount. A number of research projects (such as CEC RACE ICM) have attempted to obtain further speedup by combining both of these approaches in a single simulator. The main contribution of the work relating to parallel simulation described in this thesis is in identifying and solving some of the problems in attempting to simulate ATM networks using cell rate modelling on a parallel computing architecture. A novel synchronous time synchronisation scheme for parallel simulation, known as *timestepping*, is described and the results of experiments based on cell rate simulators using this technique are presented. These results demonstrate many of the problems in identifying parallelism in the cell rate model. Following this, the implementation of timestepping in a new cell rate simulator developed by the author is described. This simulator is used to compare

the speed performance of timestepping with the simple linear event list scheme. Through this work, a number of important characteristics of cell rate modelling are identified that have significant new implications for simulator performance.

1.1 Outline of the Thesis

The previous section has outlined the rationale behind the work described in this thesis.

Chapter 2 of this thesis introduces the essentials of ATM before the motivation for and techniques involved in the simulation of this technology are detailed in Chapter 3. The traditional method of cell level simulation is also outlined here.

Chapter 4 describes in detail the need for accelerated simulation techniques. It outlines the principles behind cell rate and parallel simulation and the results of previous studies into these two techniques are reviewed. Parallel computer architectures are also reviewed here because the precise details of this can have a significant bearing on the parallel simulation technique that is chosen.

Chapter 5 details a study by the author into an existing timestepping simulator that attempts to combine cell rate modelling and parallel techniques. This study indicates that the simple synchronous time synchronisation scheme used is not effective in exploiting parallelism in the cell rate model unless the scale of the simulation is large, even when modified to attempt to account for the sparse nature of events in a cell rate simulation. The existing simulator is found to be insufficiently flexible for a wider ranging study of the efficiency of timestepping, in particular for assessing the schemes performance in larger scale network simulations and in comparison with other time synchronisation schemes.

Chapter 6 describes in detail the design of an new object oriented ATM network simulator (SPROG) that was developed by the author in order to overcome some of shortcomings of the existing timestepping simulator. The rationale behind the development of SPROG is outlined, together with a description of how the properties of object oriented programming languages can be applied to great advantage in network simulation. SPROG is used to compare the performance of timestepping with another basic time synchronisation and event list management scheme when modelling a range of network scales and traffic loads when used in a sequential simulator. Hence a sequential simulator is applied in a novel manner in order to assess the potential performance of timestepping in a parallel computing environment. The results of Chapter 6 suggest that timestepping is only likely to be an effective time synchronisation scheme for parallel cell rate simulators for large, heavily loaded network simulations.

Chapter 7 considers the optimisation of event list management schemes in sequential simulators such that they can efficiently support cell rate modelling. The application of the spatial distribution of events, and their concurrent occurrence, is exploited in the development of novel event list management algorithms that are optimised for cell rate modelling. Experiments that assess the performance of these algorithms are described.

Chapter 8 discusses the implications of the work described in this thesis on the development of both parallel and sequential cell rate ATM network simulators. This chapter concludes with some proposals for further work that follows up the research described in this thesis.

Finally, the conclusions of the research work are presented in Chapter 9.

1.2 Summary of Contribution

The research described in this thesis contributes to two main areas of accelerated simulation of ATM networks:

- parallel simulation of a cell rate model
- optimisation of simulators to maximise their performance when supporting a cell rate model.

Throughout this thesis, a number of references are made to major research projects in the field of ATM and telecommunications network resource management of which the author was a member. These projects were:

- CEC R1022 Technology for ATD
- CEC R2059 Integrated Communications Management (ICM)
- EPSRC/DTI Project ATM Resource Management (ARMAN).

These projects provided the inspiration for, some of the background material pertaining to, and a realistic context for the research described herein. However, this research was conducted entirely by the author as a separate exercise and makes a number of contributions that are beyond the scope of these projects. In particular, the new areas that the research focuses on are:

- the identification of potential parallelism in cell rate ATM models and solving some of the problems of parallelising a cell rate model
- the investigation of timestepping as a time synchronisation scheme for parallel cell rate ATM simulators
- the development of an object oriented cell rate ATM simulator for the purpose of studying event list management algorithm performance
- the optimisation of event list management algorithms for sequential cell rate ATM network simulators.

2. Asynchronous Transfer Mode

The International Telecommunications Union (ITU-T) has identified Asynchronous Transfer Mode (ATM) as the target transfer mode solution for the Broadband Integrated Services Digital Network (B-ISDN) [I.150]. ATM is potentially able to carry all of the anticipated broadband services, while maintaining efficient use of the available network resources. Since ATM has already been described in detail in the literature, for example [I.150][Pryk91] and [Cuth93], only a brief outline of the main principles is given here.

In ATM, user (or *source*) information is transmitted in fixed size packets, or *cells*. These are 53 octets long, of which 48 octets is the actual user information (the *payload*) and 5 octets the cell header, which contains network information such as routeing. ATM is basically a connection oriented transfer mode in which cells associated with a number of connections are multiplexed onto a single link using asynchronous time division (ATD) multiplexing techniques. These connections are logical and each cell header includes a Virtual Channel Identifier (VCI) and a Virtual Path Identifier (VPI) that associate it with a unique Virtual Channel Connection (VCC). This simple labelling scheme allows network complexity to be kept to a minimum and hence allows very fast switching and routeing of cells.

ATM is capable of carrying information generated by both Constant Bit Rate (CBR) sources, such as simple voice traffic, and Variable Bit Rate (VBR) sources such as video coder/decoders (CODECS). Services for which a guaranteed quality of service (QoS) is required, such as video, can be carried, as well as those for which the QoS requirements are less stringent, for example Internet Protocol (IP) data.

Four transfer capabilities are defined [I.371]:

- Deterministic Bit Rate (DBR)
- Statistical Bit Rate (SBR)
- Available Bit Rate (ABR)
- ATM Block Transfer (ABT)

DBR is used for connections that require a static amount of bandwidth that is continuously available during the connection lifetime. SBR, on the other hand, is used for connections for which any characteristic variation in the traffic is known at connection set-up time. In ABT, bandwidth is allocated in terms of a ATM block, each block being essentially equivalent to a DBR connection. ABR is intended to support traffic from sources that are able to reduce their information transfer rate to zero if the network requires them to do so, but may also increase their information transfer rate if there is extra bandwidth available. In this thesis, only those characteristics of ATM relevant to DBR and SBR are considered.

ATM Cells are carried in *slots* and they are only generated when there is information for transfer. When there is no information to transfer, unassigned cells are carried in the slots. ATM is also able to guarantee cell sequence integrity i.e. cells on a particular VCC will always arrive at their destination in the same order that they were sent out from the source. Because ATM networks must operate at very high speeds, network simplicity is essential. Therefore, ATM includes no provision for error recovery, which must instead be performed on an end-to-end basis.

2.1 ATM Protocol Reference Model

Figure 1 shows the ATM protocol reference model [I.121]. It consists of a user plane, a control plane, and a management plane. The user and

control planes are divided into layers and sub-layers, as shown in Table 1. The lowest of these, the Physical Layer, is responsible for details such as bit synchronisation, framing, and the physical transmission medium itself. Above this, the ATM Layer provides cell transfer for all services, while the ATM Adaptation Layer (AAL) provides service dependent functions to the higher layers. The management plane provides network supervision functions.

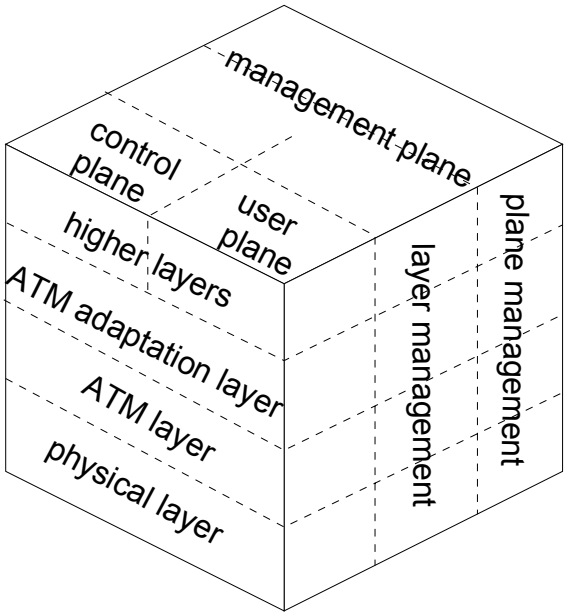

Figure 1 The Protocol Reference Model for ATM

Layer/Sub-layer		Function
ATM Adaptation Layer	Convergence sub-layer Segmentation and re-assembly sub-layer	Convergence Segmentation and re-assembly
ATM layer		generic flow control cell header generation/extraction cell VCI/VPI translation cell multiplex and de-multiplex
Physical Layer	Transmission convergence sub-layer Physical medium sub-layer	cell rate decoupling HEC header generation/verification cell delineation transmission frame adaptation transmission frame generation & recovery bit timing physical medium

Table 1 ATM Protocol Layer and Sub-layer Functions

2.2 Statistical and Deterministic Multiplexing

In ATM, each connection is allocated a certain amount of bandwidth at call set-up time. However, because variable bit rate as well as constant bit rate sources can be accommodated, a scheme for allocating the required bandwidth while guaranteeing a certain quality of service to the user is required.

If deterministic multiplexing is used, then the peak source bit rate is allocated to each connection. Whilst this can lead to a very high quality of service because cell level congestion is minimised, it is wasteful of the available bandwidth for 'bursty' VBR sources. This is because, with VBR sources, the full allocated bandwidth will not be used all of the time, but

any spare bandwidth will not be available to other currently more active connections because it has already been allocated.

In statistical multiplexing, the bandwidth allocated to each connection is less than the peak bit rate, but generally greater than the mean bit rate. This means that more of the network resources will be utilised at any given time and leads to the interesting situation where the sum of the peak bandwidths of all the VCs on a link (or VP) can exceed the total available bandwidth. Statistical multiplexing is effective provided that that the sources are sufficiently 'bursty' and that they are uncorrelated i.e. the peak bit rates of the different sources do not occur simultaneously.

2.3 Quality of Service in ATM

The concept of Quality of Service (QoS) in ATM is very complex and relates to the satisfaction gained by a user from the service provided by a network. The formal definition given by ITU-T in Recommendation I.350 [I.350] is: *the collective effect of service performances which determine the degree of satisfaction of a user of a service.* Whilst this definition is useful to the user of a network, it is highly subjective and is less meaningful to a network provider. Therefore, the ITU-T has also defined a further measure known as *Network Performance* (NP) which is characterised by measurable and calculable parameters. The formal definition of NP, also given in Recommendation I.350, is: *the ability of a network or network portion to provide the functions related to communications between users.* This is characterised by a number of generic parameters that are applicable to all digital networks, including access speed, information transfer speed, information transfer accuracy and dependency, and disengagement speed. These map to specific network performance parameters in ATM including connection set-up time, bandwidth, cell loss ratio (CLR), cell delay and cell delay variation (CDV), and connection clear-down time.

When a connection is set-up across the ATM network it is done so in accordance with a contract that is made between the users and the network provider. This contract specifies, amongst other things, the QoS that the provider must provide to the user. This will, at least in part, be interpreted by the network provider as a set of guaranteed network performance parameters.

Dimensioning and *Performance Engineering* are two processes that must be applied by the network provider in order to be able to guarantee a given QoS or NP to a user. Dimensioning is a longer term activity that focuses on the organisation and provisioning of sufficient equipment to meet the expected demands of users, and requires a knowledge of usage patterns and service characteristics [Pitts93]. Performance engineering, however, is a more detailed activity that concentrates both on the detailed design of the equipment and on the management of the low level (i.e. cell, burst and connection level) resources of the network in order to guarantee QoS. This requires detailed knowledge of the performance limits of equipment, the service characteristics and information about typical service mixes.

2.4 Traffic Control and the Role of Network Management

The primary aim of traffic control and congestion control parameters and procedures is to protect the network in order to achieve network performance objectives [I.371] which should be done while optimising the use of network resources. This is necessary in order to guarantee the QoS that is agreed with users at call set-up time.

The ITU-T outlines a number of generic functions to meet the above objectives. These are:

- *Network Resource Management* - this includes provisioning

- *Connection Admission Control (CAC)* - this is defined as the set of actions taken by the network at the call set-up phase in order to establish if a virtual channel or virtual path connection (VCC or VPC) can be accepted. This decision is based on an assessment of the currently available bandwidth and the bandwidth required by the new connection.

- *Usage and Network Parameter Control (UPC and NPC)* - this is the set of actions taken by the network in order to monitor and control traffic at the user access and the network access respectively. The main purpose is to protect network resources from malicious as well as unintentional misbehaviour that can affect the QoS of existing connections by detecting violations of negotiated parameters and taking appropriate actions.

- *Priority Control* - the user may generate different priority traffic flows that may be dealt with differently by network elements. For example, low priority cells could be discarded during periods of congestion in order to protect the QoS of other connections.

- *Congestion Control* - the set of actions taken by the network in order to minimise the intensity, spread and duration of congestion.

The ITU-T has also identified a number of traffic parameters that enable the above procedures to characterise the traffic. These include:

- *Peak cell rate*

- *Mean cell rate*

- *Burstiness*

- *Peak cell rate duration*

- *Source type*

Because of the complex nature of ATM traffic patterns, the precise way in which traffic is characterised can be extremely important. For example, in the case of CAC, the number of connections that can be accepted onto a link and their quality of service is highly dependent on which of the above parameters is used. Generally, a number of parameters will be used and an *effective bandwidth* for a connection, or set of connections, derived from these.

Most traffic control functions operate within the control plane of the ATM protocol model. Network management, however, also operates within the management plane and is responsible for a full range of higher-level, longer-term planning and resource management functions. Network management is implemented using a *telecommunications management network* [TMN], defined in RACE CFS H100 [H100] as *a system which supports the management requirements of administrations to plan, provision, install, maintain, operate and administer telecommunications networks and services.* Whilst simple network management systems are already used in telecommunications networks, the sheer flexibility, range of services, and consequent complexity of the traffic patterns on a large ATM network mean that powerful TMNs will be required to successfully manage the network. TMN functionality is classified in [ICM92.1] into 9 user specific areas (Design, Planning, Installation, Provisioning, Accounting, Customer Query & Control, Maintenance & Fault Management, Performance Management, and Security) and 5 user generic areas (Configuration Management, Test Management, Event Management, Log Control, and Monitoring). Whilst many of these operate at a very high level, many interact with the control plane traffic control functions of the ATM network. For example, performance management will be able to modify CAC parameters based on it's perception of the current traffic load on the network in order to maintain the QoS of current users, while areas such as planning will impact upon dimensioning of the network.

2.5 Summary

This chapter has introduced the basic principles of ATM. ATM has been identified as the target transfer mode for use in the B-ISDN. In ATM, information is encapsulated in fixed length cells and these cells are transmitted in slots. ATM is capable of carrying traffic generated by both CBR and VBR sources while making efficient use of the available network resources. This is possible through the use of statistical multiplexing techniques. When connections are set-up across the ATM network, they are done so in accordance with a contract between the user and the network provider. This guarantees a certain quality of service to the user. In return, the user must comply with restrictions on the offered traffic. Resource management algorithms, such as CAC, ensure that sufficient network resources are allocated to new connections to guarantee the contractually agreed QoS while not compromising the QoS of existing connections. Policing functions such as UPC protect the network against misbehaving traffic sources. Management functions are provided by a Telecommunications Management Network. This is a large distributed computing system that provides services ranging from performance management to longer term services that aid activities such as network planning.

3. Simulation

A *simulation* is a representation of certain features of the behaviour of a physical system or abstract model of a real system [ICM92.2]. This is to be distinguished from an *emulation* which is an implementation of a system that exactly duplicates some of the behaviour of the real system. Simulators are systems comprising either software or a combination of software and hardware, and are developed to apply the simulation process. In general, simulation tools are flexible and can be used for many modelling applications including:

- Making experimental measurements or predicting the behaviour of the simulated system.
- Aiding in system synthesis and analysis for a system that is under construction.
- Testing the system definition.

The process of developing a simulator consists of a number of steps, central to which is an understanding of the original problem. These steps are:

- Problem formulation
- Modelling
- Model implementation
- Verification and validation
- Data collection and analysis

3.1 Motivation for Simulation in ATM

The simulation of ATM networks and network elements has had, and continues to have, many applications ranging from the initial research and development of ATM technology to the testing and validation of network management systems. Traditionally the main application of simulation has been in ATM research and the design of equipment. For example, simulation was used extensively in the initial design of the CEC ACTS EXPERT Test-bed (ETB) [SNH96] ATM demonstrator. More recently, simulation has found application in the development of telecommunications management systems such as those of CEC RACE projects NEMESYS [NEME93] and ICM [ICM92.2][Swift94]. In particular, simulation, as opposed to the use of real networks, has continued to be applied to solving ATM and network management problems for the following reasons:

- The unavailability of commercial scale networks for experimentation and the testing of new equipment, code and algorithms.
- Difficulty and cost of instrumenting real networks i.e. to do realistic experiments on real networks large numbers of ATM traffic generators and analysers, for example, will be required.
- Lack of configuration control: Many existing ATM networks simply do not allow access to the traffic and configuration parameters that experimenters require.
- In order to adequately verify network management procedures, they must be tested in wide range of situations and network topologies. This is not possible on a real network.
- Evaluating new equipment and management algorithms can be achieved much more quickly and at a much lower cost with a simulator.

In TMN research applications, simulators have the following advantages over real networks [Bocci95.2]:

- *Scaling:* Simulators are be able to simulate large high-speed ATM networks. Present day ATM demonstrator networks are usually small scale and laboratory based.

- *Flexibility:* Compared to a real commercial network, simulators can offer flexibility in the following respects:

  Functionality: A variety of network functions, technologies and traffic types can be supported by simulators.

  Measurement: Simulators provide access to a variety of network data and parameters that may be difficult or impossible to obtain in the real system.

  Scenarios: Simulators can model a variety of traffic scenarios as well as eventualities such as node and link failures and buffer overflows.

- *Portability*: In general, simulators are a software product that can be designed to be ported easily between different computing platforms. This means that simulation experiments can performed at many different sites with a minimum of specialist equipment.

The application of simulation in TMN studies imposes some specific requirements on the design of the simulator:

- *TMN Interface:* Simulators for use in TMN studies should provide a standard TMN interface, such as a Q3 adapter [M.3010].

- *Simulation Speed:* Ideally, the simulator and the TMN should have a common perception of time. This can be achieved by either ensuring that the simulator runs in real time, or by introducing a mechanism in the Simulator/TMN interface to pass the simulated time from the

simulator up through the layers of the TMN, thus ensuring that the TMN always uses the simulator for its time reference. The latter approach was used in the ATM simulator developed by ICM [ICM95]. However, this approach requires a specially modified TMN platform that can accept simulated time as real time. The research described in this thesis is aimed at maximising the speed of the simulator so that there is less requirement for a specially modified TMN platform.

Of course, simulation is not the only method for predicting the behaviour of ATM networks without recourse to the use of real networks. Mathematical analysis can also be used. However, stochastic systems are very difficult to analyse mathematically and in order to model all but the most simple situations significant simplifications are required. These can lead to unacceptable inaccuracies in the results obtained. Furthermore, mathematical analysis is particularly unsuitable for network management studies where dynamic interaction is required between the TMN and the system being modelled.

Despite the fact that it is envisaged that ATM networks will become progressively more available to experimenters in the future, and that these networks will provide increased access to the appropriate parameters, they will generally be commercial revenue earning networks and hence the cost and risk factors associated with experimenting on them will remain. Therefore simulation will retain many applications despite the widespread availability of real networks.

3.2 Simulation Principles

Simulation has two basic forms: *discrete* and *continuous* [Phillips92]. In discrete simulation, the changes in a system over time are represented as a series of instantaneous occurrences of *events*. These are reflected by changes in the *state variables* that describe the current *system state*.

Discrete simulation enables a more detailed range of data to be collected than continuous simulation. For example, consider a queue in a bank. In a continuous simulation the movement of customers through the queue would be modelled as a flow or rate, so only parameters such as mean arrival rate would be applicable. However, in a discrete event simulation, the movement of each individual person would be modelled and hence 'rare events' such as the refusal to admit individual customers could be studied.

3.2.1 Time Advancement and the Event List

Discrete event simulations typically model dynamic processes that occur over a particular time duration with events representing state changes that occur at specific points in time. A simulation clock is used to keep a record of the simulated time, and events are generally time-stamped with the simulation time at which that event is scheduled to occur. Furthermore, some way of storing all the foreseeable future scheduled events is required. This is accomplished by the *event list* which normally takes the form of a time ordered sequence of event records with the next scheduled event at the head. The event list can be sited within the logical architecture of the simulator according one of two schemes, the *centralised event list* architecture and *distributed event list* architecture. In a centralised event list architecture, one large event list is used to store all of the events for the whole simulation, whereas with a distributed event list there is a list for each model in the simulation that stores all of the locally scheduled events.

There are two principle schemes for advancing the simulation: *time driven simulation* and *event driven simulation* [Rich89]. In both of these schemes the simulation essentially progresses through a process of the next event being taken from the event list, processed, and any resultant events being put back in their correct time position in the list. Where they differ is in the way that the simulation clock is advanced.

In time driven simulation *fixed-increment* time progression is used. Here, the simulation clock advances by constant amounts, or *ticks*. After each tick, all of the events that were scheduled to occur during that time interval are processed, and the system state updated. This scheme is generally used if events are known to occur at fixed regular intervals but it can be inefficient if the tick is set incorrectly. If the tick is set to small there will be only a low probability that an event will be scheduled for a given tick. Therefore, there will be many ticks in which no useful processing is done in the simulator. However, if the tick value is set too large, then the error resulting from the fact that all events scheduled for that tick are processed at the end of the tick will be unacceptably high.

In event driven simulation a *next-event* time progression scheme is used. This is the most common technique used in ATM simulation because it can cope with random event times. In this scheme, after the processing of an event, the simulation clock is advanced to the time of the next scheduled event in the event list. Therefore, periods of inactivity are skipped and processor time is not wasted waiting for future events to occur.

The sequence of operations that occurs in an event driven simulator is as follows; During system initialisation, the simulator configuration is loaded, simulation parameters and measurements initialised, the simulation clock reset, and each of the components of the system polled to determine when they expect to generate their first event. Based on this the event list is initialised. The simulation then enters a loop in which the earliest event in the event list is processed, the system state updated, and any resulting events placed back in the event list in correct time order. A check is then made to see if some stop criterion has been met which could be some end simulation time, or it could be based on the state of some other system variable. If this criterion has not been met, then the next event in the event list is fetched and processed. Once the stop criterion is satisfied, the loop exits and the final results of the simulation collected.

Figure 2 illustrates, using the ITU-T Specification and Description Language (SDL) [Z.100][Belina89], the sequence of operations that occurs in an event driven simulator:

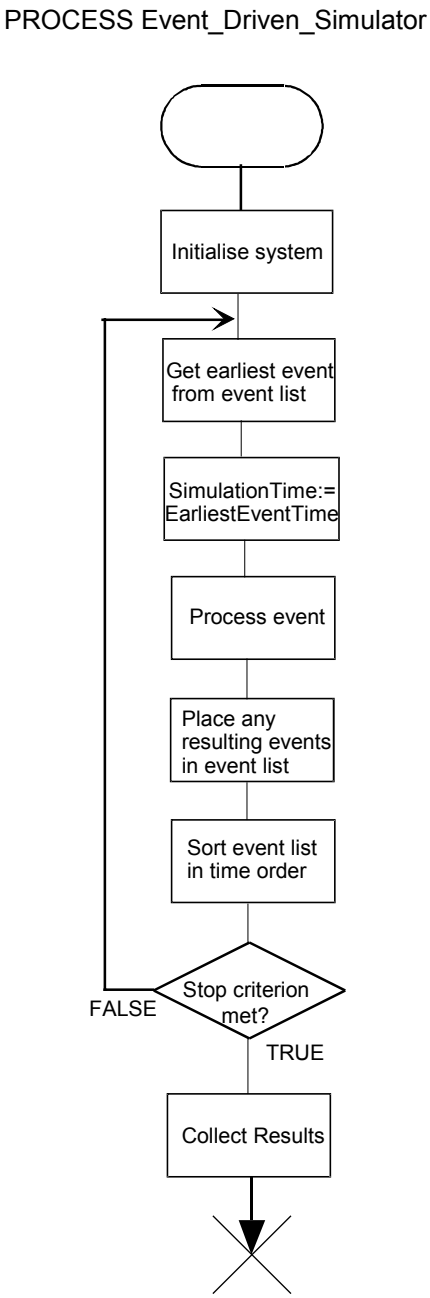

Figure 2 Sequence of Operations in an Event Driven Simulator

3.2.2 Event Generation and Random Number Generators

Event times can be generated in one of two ways. Measurements from a real system can be used to determine the times of the events. However, whilst this method can offer authenticity, it has the disadvantage that for reasonable length simulations a great deal of data must be stored. Furthermore, it implies that real systems already exist to provide such data to experimenters. Clearly, this is a problem in relation to ATM simulation since the real systems do not necessarily already exist, and even if they do access to data from them is often difficult, costly and time consuming.

An alternative method of event generation is to use random number generators coupled with models of the sources (e.g. probability distributions of cell inter-arrival times). From these, particular event times can be calculated. However, it is important to note that random number generators are usually *pseudo-random* and so care must be taken to ensure that these times are adequately realistic. A good pseudo random number generator should produce a sequence of numbers, evenly distributed, which should not exhibit any correlation between each other. The sequence should also be reproducible, which aids in debugging and can be used to increase the precision of results [Pitts93], and should be as long as possible.

A review of random number generator algorithms is given by Law & Kelton [Law91], an in depth discussion of these being beyond the scope of this thesis. The Wichmann-Hill algorithm [Wich82] has shown particular application in both cell rate and cell level simulation of ATM networks. This is a high performance algorithm with a period (sequence length) of about 7×10^{12} and the major advantage that it is highly portable, being suitable for both 8 and 16 bit processors.

3.2.3 Transient and Steady States

When a simulation is first started any measurements made will be partially dependent on the initial states of the system variables. After some period, the simulation will settle down to a steady state at which the distribution of measurements (but not necessarily the actual measurements themselves) will become stable. This initial state is known as a transient state. In ATM simulation it is not just applicable to the period immediately following the start of a simulation run, but also following major occurrences when, for example, new sources start generating traffic immediately following the set-up of calls. The precise nature of that being measured using the simulator will determine whether the steady or the transient state is of interest. In general, the study of cell level phenomena such as cell losses or cell delay will require that the simulator has reached the steady state in order to be certain of statistically significant results. However, if call level phenomena, such as call set-up are of interest then clearly the transient state will be more relevant. In either case, it is often difficult to judge the exact point at which the transient state ends and the steady state begins.

3.3 Simulation Languages and Tools

So far in this thesis simulators have only be described in the most abstract terms. In practice a simulator will be implemented as a system consisting of the hardware and software necessary to model the real problem. As such, the experimenter has a range of options relating to the choice of computer hardware, the programming language, and any high level tools to aid the design of the simulator.

Traditionally, ATM simulators have been implemented purely as software running on a single or multiprocessor computer. This has the advantage that it is often easy to produce a software model of a real system and it is relatively easy to reconfigure such software for a variety of different

experiments. However, such an approach suffers when the purpose of the simulation is to study rare events, such as cell losses. Since software based simulators typically run at speeds much less than real time (the simulation clock advances at a much slower rate than the real 'wall clock' time), very long simulation runs are often necessary.

This thesis focuses on the traditional architecture of computer software running on a single or multiprocessor platform. The choice of platform is dependent on the requirements on the speed of the simulator, as well as other factors such as the availability of an appropriate platform. General purpose single processor machines such as the SUN SPARCStation or IBM PC have the advantage that they are relatively cheap, widely available and have a huge existing library of software development tools. Parallel multiprocessor machines have the advantage, on the other hand, that if the simulator is carefully designed they can exhibit improved speed performance over single processor sequential machines for a given CPU clock speed. This issue is considered in depth later in this thesis.

The choice of the language in which the simulator is written can also have a significant impact on both its performance and on the development time. The simulators described later in this thesis are all written in general purpose programming languages. For example, MICROSIM and LINKSIM are written in PASCAL while the ICM simulator, MADS, is written in ANSI C. Such languages have the advantage that they often allow hand optimisation of programs, so enabling fast simulators to be written. Furthermore, their highly structured nature means that real systems can be modelled in software with little difficulty.

Despite the widespread use of conventional programming languages, several specialised languages and tools exist that can be used to generate the simulation model. One example is the SIMULA language in which real systems are modelled as a number of interacting processes. Whilst this helps in the modelling process, such language compilers can generate

relatively inefficient executable code that can be substantially slower than an equivalent simulator written in a conventional language. Object oriented languages, such as C++, have also been used to write simulators. These have the advantage that object modelling techniques can be applied to produce intuitive models of real systems. Issues relating to the object oriented design of simulators are considered in more depth in Chapter 6 of this thesis.

Recently, a number of specialised commercial simulation tools have emerged that allow the user to define the network using a graphical interface. For example, the OPNET[1] [OPN96] development environment allows the user to define the network in terms of nodes, process models and state diagrams. This description is then used to generate the appropriate C code which is finally compiled to produce a self contained simulator. Other examples of such commercial packages include MODSIM, COMNET and BONES. Whilst such approaches can give rise to considerable savings in the development time for a simulator, as in the case of specialised simulation languages the code generated is often inefficient and hence run times can be relatively long when compared with hand-crafted simulators written in a conventional programming language. Furthermore, whilst these packages are usually supplied with comprehensive libraries of models of more established network technologies, such as Ethernet, the provision of ATM models is currently very limited. Therefore, the simulator developer must often resort to writing much of their own code in order to satisfy their own requirements for simulating ATM networks. This can reduce the development time advantage of such commercial packages. This thesis is concerned with methods for reducing the run time of ATM network simulators and not development time. Whilst an in-depth review of such commercial packages is considered to be beyond the scope of the thesis, because of the increasing popularity of such packages it is valuable to briefly describe the

[1] OPNET is a trademark of MIL3 inc.

features of one of the more comprehensive systems that is currently being used in a number of projects, for example DTI/EPSRC project ATM Resource Management (ARMAN), for modelling ATM networks.

OPNET stands for Optimised Network Engineering Tools. It is a workstation based application for the modelling and simulation of communication systems. OPNET allows the user to specify the system in terms of incrementally decreasing levels of abstraction from the sub-net level down to the individual process level. This is done using a graphical user interface (GUI). These specifications are then compiled to produce a simulator executable that can be run, either under the direct control of the OPNET GUI or independently. Debugging and measurement functions can be embedded within the simulator executable.

An overview of the simulator development process using OPNET is given in Figure 3. The user specifies the network to be modelled using a top-down approach in terms of three principle layers: In the Network Editor, the Network Layer of the model is specified as a number of nodes connected by links. Each node is specified using the Node Editor and consists of interconnected process and queue modules. The behaviour of process modules is defined using one or more state transition diagrams that also encapsulate user-defined Proto-C code [OPN96]. Note that at each of these levels the user can either define their own models or can make use of pre-defined models from the OPNET library. In version 2.5B of OPNET, the ATM models in the OPNET library are extremely limited. The user can also specify the format of intra-model and inter-model messages (packets or interrupts) using the Parameter Editor and also the measurement statistics that are recorded in output scalar or output vector files for later analysis.

Once the network model has been fully defined it is compiled and linked with the OPNET simulation kernel to produce a simulation executable. This executable can be either run as a stand alone simulation or run

under the control of OPNET. Output vector and output scalar files generated during the simulation run can be analysed using the Analysis Editor.

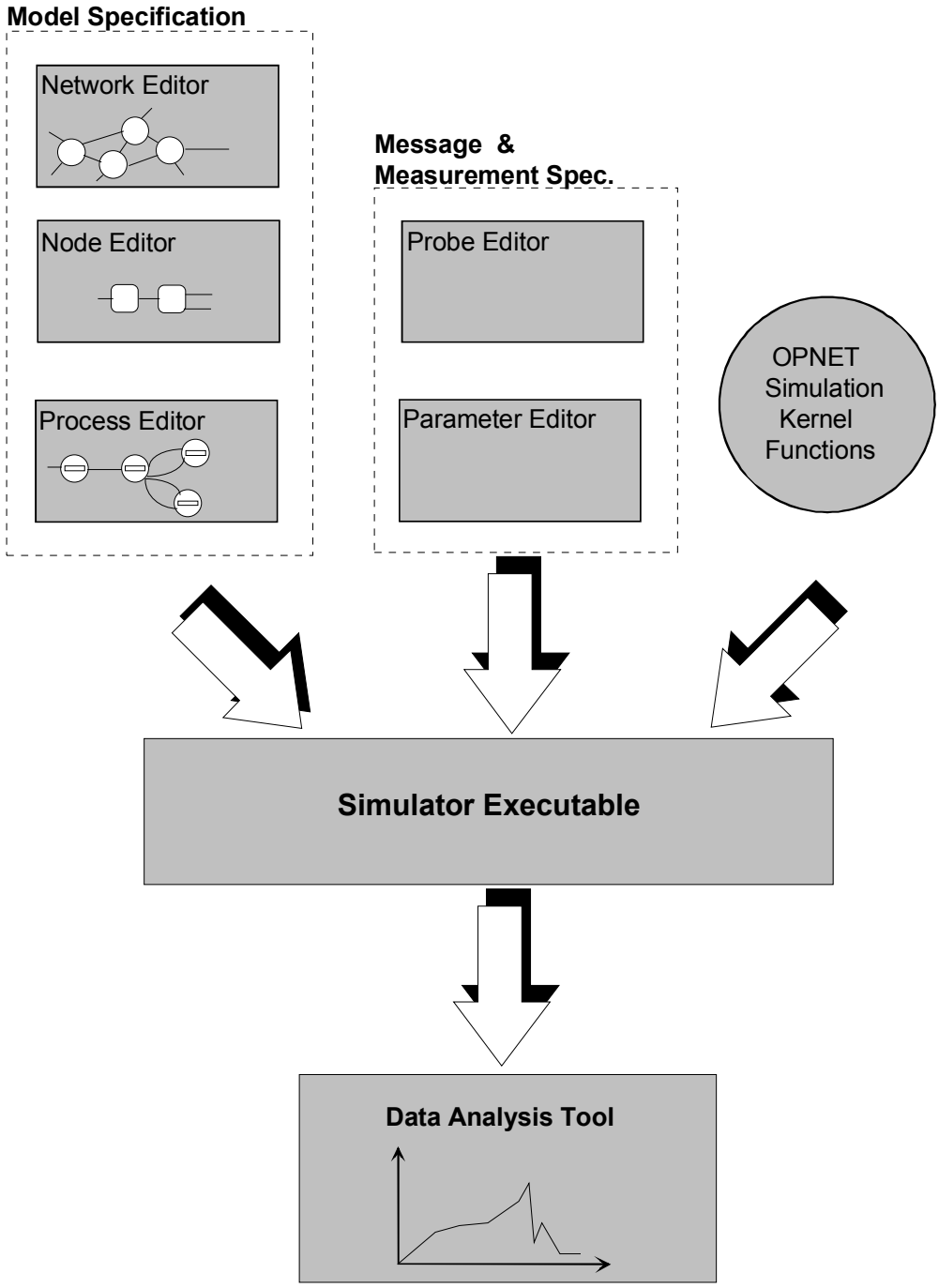

Figure 3 Simulation Development Process with OPNET

3.4 Cell Level Simulation of ATM Networks

Cell level simulation is the classical method for simulating ATM networks and network elements. It is an extremely accurate method since the propagation of every individual cell through the network is modelled. However, cell level simulation can be very slow because of the shear volume of cells that must be simulated in order to obtain statistically significant results.

3.4.1 Principles

ATM networks are generally modelled as an interconnection of a variety of elements, including links, queues or First-In-First-Out (FIFO) buffers, delay elements, servers and traffic generators. Queues can be used to represent buffers, for example in switches, while servers represent the cell multiplexing and routeing elements of switches. These basic elements are augmented by others that model, for example, the parts of the network responsible for signalling or traffic control. Figure 4 shows how these basic elements can be connected to model a simple shared buffer multiplexer.

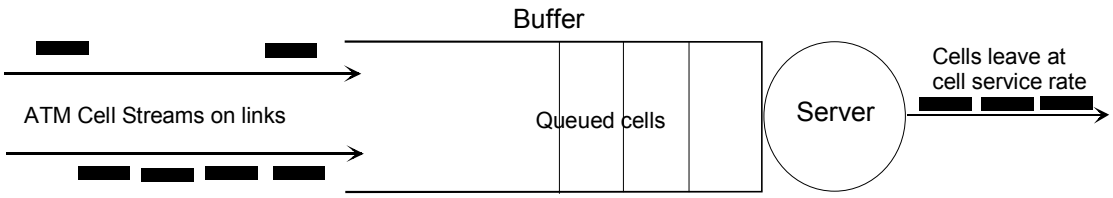

Figure 4 Simple Model of ATM Multiplexer

Each of the cells shown in Figure 4 is represented by an event. Cells enter a buffer via links, and leave when they are served by the server. This occurs at a constant rate, known as the *cell service rate*. Note that there are two situations in which cells will be queued. The first of these is if the total input rate to the buffer exceeds the cell service rate, known as *burst scale* queueing. The second of these occurs when two or more cells arrive

at the buffer simultaneously. Clearly only one can be served at a time and hence the other cells are queued while this one is served. This known as *cell scale* queueing. Cell level simulation is able to model both of these effects and so can model queues to a great level of detail. Figure 5 illustrates the cell and burst scale components of queueing. From this it can be seen that if only the burst component is modelled (as in cell rate simulation), then the measured cell loss probability is less.

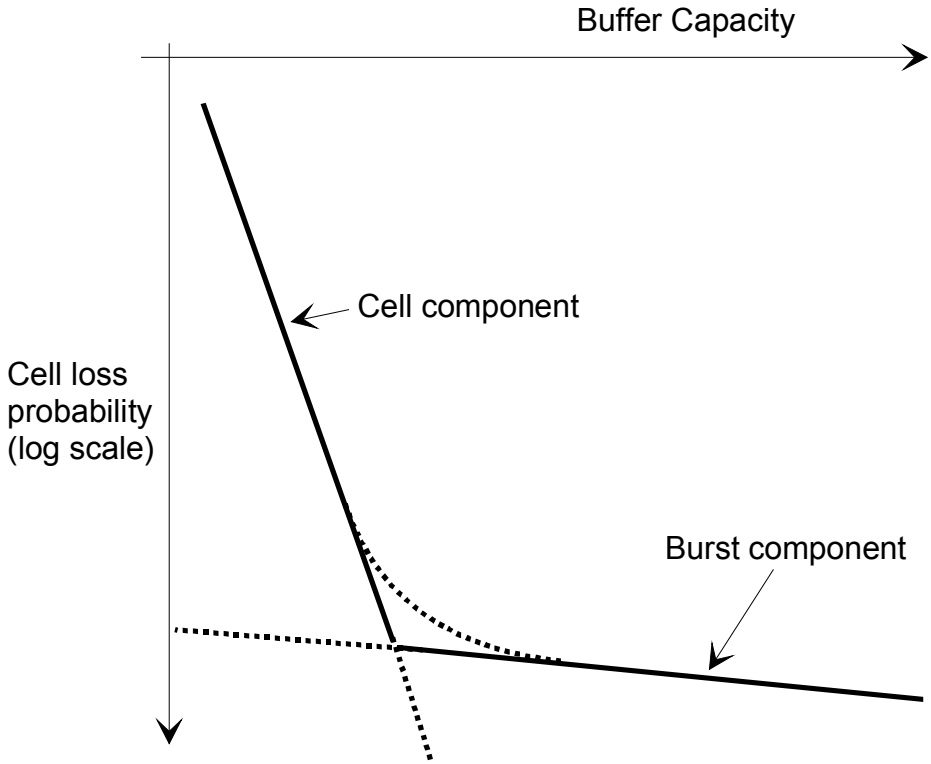

Figure 5 Cell and burst scale components of queueing

Network users are represented by traffic sources which may generate the traffic for one or more simultaneous connections associated with some user activity. In order to model a given user, the traffic which that user would inject into the network must first be characterised in terms of parameters such as mean inter-call time, mean cell rate and peak cell rate. Random number generators with appropriate probability distributions can then be

used in the simulator to generate the event times to model the traffic. A detailed description of source modelling is beyond the scope of this thesis.

3.4.2 MICROSIM: An Example Cell Level ATM Network Simulator

MICROSIM is a simulator developed at Queen Mary and Westfield College [Schor94]. It is written in PASCAL and designed to run on single processor sequential computers such as the SUN SPARCStation or IBM PC. It is able to simulate ATM and fast packet switch networks and provides for a very flexible network configuration. MICROSIM is configured using simple text files that enable a variety of model elements ranging from delay elements (that can represent link delays) and queues to a number of source models to be interconnected to model network elements or complete networks. MICROSIM enables a number of parameters to be measured and these are saved in a text file. This results file includes the following data per network element in the simulation: maximum buffer wait, mean buffer wait, cell throughput, cells blocked (i.e. cells lost), buffer length at the end of the simulation, cell delay distribution, and state probability distribution (i.e. probability distribution of buffer states, based upon arriving cells).

3.5 Summary

This chapter has reviewed the motivation for, and principles behind the simulation of ATM networks. Simulation is required both for the development of network hardware and management software, as well has for network planning and provisioning. Simulators provide a convenient and secure alternative to experimentation for these purposes on real networks. The principles of discrete event simulation were introduced, followed by a review of simulation languages and existing tools available to aid network simulator development. Finally, cell level modelling of

ATM networks was briefly described together with an existing example cell level simulator, MICROSIM.

4. Accelerated Simulation Techniques

Cell loss ratio and cell delay are important parameters of the network performance. The value suggested for the maximum cell loss ratio for adequate network performance is 10^{-8} [CCITT89]. Therefore, in order to simulate the loss of one cell at least 100 million cell arrivals at a queue must be simulated; for statistically significant results it is necessary to simulate at least 2 orders of magnitude more cell arrivals. In cell level simulation each cell arrival and departure from a queue is represented by an event and so a correspondingly large number of events must be processed. This results in very long simulation times, often amounting to many hours of 'real' time just to simulate a few minutes of 'simulated' time.

Accelerated simulation techniques attempt to reduce simulation times, not only to enable results to be obtained more quickly, but also to increase the potential for dynamic interaction between simulators and experimental network management systems [Swift94][Bocci95.2], or even real ATM networks.

Many approaches have been proposed for the accelerated simulation of ATM networks, including implementation techniques, modelling techniques, and statistical techniques. These are reviewed by Pitts [Pitts93]. This thesis concentrates on two approaches: cell rate simulation (a modelling technique that increases the simulation speed by decreasing the number of events that have to be processed), and parallel simulation (an implementation technique that achieves speedup by exploiting the inherent parallelism in a network model).

4.1 Performance Measures for Simulators

When assessing the effectiveness of a particular accelerated simulation technique it is necessary to define some metric that accurately describes the speed of the simulator. The speed of a simulator can be defined as the rate of doing some useful processing work; in ATM simulators this is commonly measured in terms of a cell processing rate.

Speedup is a measure of the increase in performance of a computer system that is achieved by making some enhancement to it. In simulation, the concept of speedup can be used to compare the relative speeds of two simulators used to simulate identical networks. Consider two simulators: simulator *b* uses some accelerated simulation technique to increase it's speed over simulator *a*. The speedup of simulator *b* over simulator *a* is defined by Amdahl's law [Amd88] as:

$$Speedup = \frac{Execution\_time\_of\_simulator\_a}{Execution\_time\_of\_simulator\_b}$$

Alternatively, if the speed of the two simulators is measured in terms of their respective cell processing rate:

$$Speedup = \frac{Cell\_proc\_rate\_of\_simulator\_b}{Cell\_proc\_rate\_of\_simulator\_a}$$

4.2 Cell Rate Simulation

4.2.1 Overview

In traditional cell level simulation of ATM each cell arrival at, or departure from, a network element is represented by an event. One way of increasing the speed of the simulation is to reduce the number of events that have to be processed. *Cell rate* simulation (also known as *burst level*

simulation) does this by modelling changes in the rate of flow of cells, or bursts of cells, within the network. This technique has been studied by Pitts [Pitts93] and has been implemented in a number of practical ATM network simulators, including LINKSIM and MACROSIM [Pitts90] from Queen Mary and Westfield College, and the parallel architecture simulators from CEC RACE projects MIME [MIME91] and ICM [Swift94].

4.2.2 Modelling the Traffic

The basic unit of traffic within cell rate modelling is a *burst* of cells. This is defined as *a cell rate lasting for a particular time period during which the inter-cell arrival time does not vary* [Pitts90]. An event represents a change from one cell rate to another cell rate, as illustrated in Figure 6.

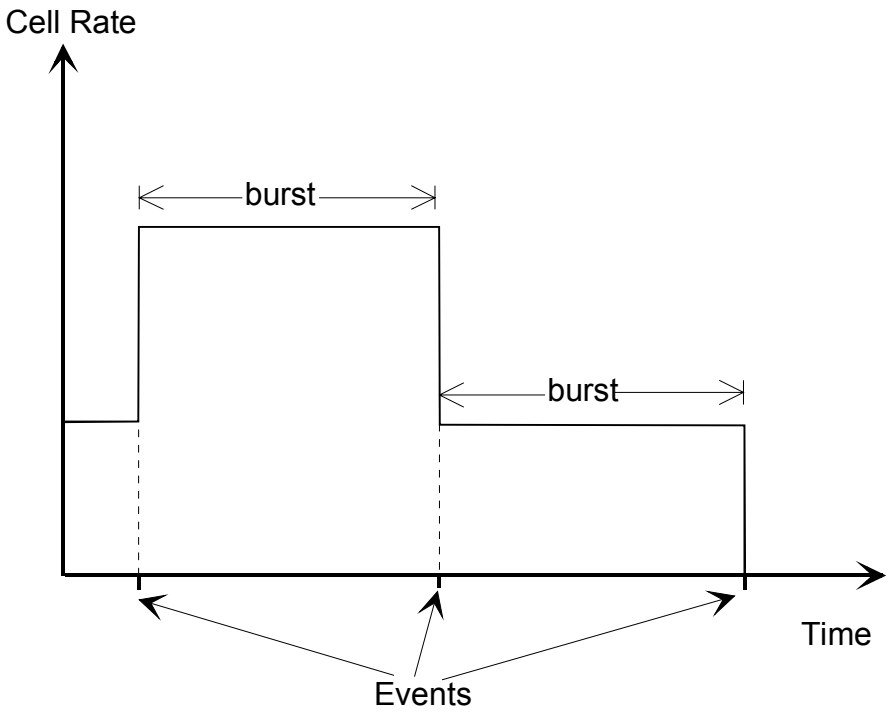

Figure 6 Basic unit of traffic

Each traffic source is characterised by a group of states, each state representing a particular cell rate produced by that source. The source changes from one state to another at a time determined by a probability

distribution. On each state change an event is generated by the source model indicating that the cell rate has changed from the old rate to some new rate. The choice of probability distribution and the transition (with particular probabilities) from one state to another is such that the source produces traffic that models the traffic from a real source.

4.2.3 Operation of the Queue

The queue model is one of the basic elements of any simulation of an ATM network. In cell rate simulation this is represented using a fluid flow model and is illustrated in Figure 7. Two parameters describe the queue: the maximum number of cells that it can hold, or *buffer capacity,* and the constant rate at which cells leave the queue, or *cell service rate*. The state of the queue at any given time is determined by the balance between the total input rate (i.e. the combined traffic from all of the virtual channels), and the service rate.

If the input rate is equal to the service rate then the queue size will remain stationary. If the input rate is less than the service rate then the queue will shrink. However, if the total input rate exceeds the service rate, then the queue will grow until such time as either it becomes full and cells are lost, or the input rate drops to less than or equal to the service rate. These relationships are summarised by the following balance equation which accounts for all of the cells within the queue:

$$input\_rate = service\_rate + queueing\_rate + loss\_rate$$

Because the queue is served on a first-in-first-out basis (FIFO), changes in the cell rate at the input to the queue will take a finite time to propagate through to the output of the queue if there are cells queueing. This time will depend on the current size of the queue and the cell service rate.

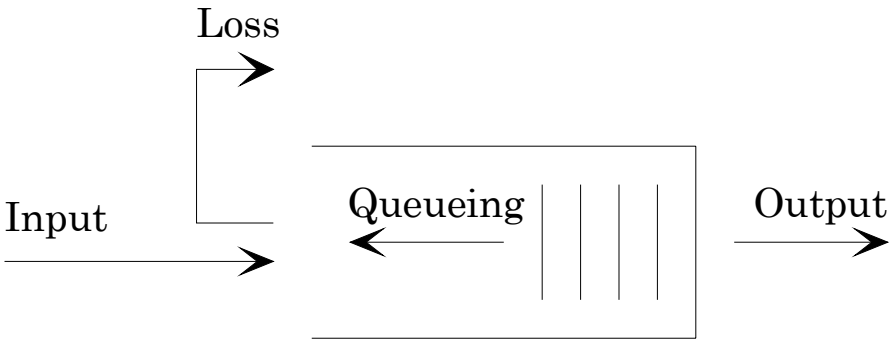

Figure 7 The Queue Model

It is important to note that, because the discrete nature of ATM cells is not modelled, cell rate modelling can model burst scale queueing but it is not able to model the effects of cell scale queueing. Furthermore, it is also unable to model the simultaneous arrival of individual cells at the queue.

4.2.3.1 Analysis of Queue Behaviour

Consider the queue shown in Figure 7. Nine possible states are defined describing its current state [Pitts93] and these are shown in Figure 8.

	Size of the queue		
	empty	mid	full
input > service rate	1	4	7
input = service rate	2	5	8
input < service rate	3	6	9

Figure 8 The Nine Possible States of a Queue

Whilst these states fully describe the current state of the queue, they are not adequate to fully model the queue because the delay in the propagation of events through the queue is not represented. In order to

produce a more complete model, the state of the queue at the previous event must also be considered.

The following analysis by Pitts & Sun [Pitts90] provides a more complete description of the behaviour of the queue.

4.2.3.1.1 Notation

Let $I(i,e)$ = input cell rate
$O(i,e)$ = output cell rate
$Q(i,e)$ = cell queueing rate
$L(i,e)$ = cell loss rate
$C(i,e)$ = number of cells queued
C_{max} = buffer capacity
O_{max} = cell service rate

where i {1,...,n} indicates the i^{th} VC and e {1,...,n} indicates the e^{th} event at the queue.

Let $I_{tot}(e) = \sum_{i=1}^{n} I(i,e)$ = total input cell rate for all VCs at event e

$O_{tot}(e) = \sum_{i=1}^{n} O(i,e)$ = total output cell rate for all VCs at event e

$Q_{tot}(e) = \sum_{i=1}^{n} Q(i,e)$ = total queueing rate for all VCs at event e

$L_{tot}(e) = \sum_{i=1}^{n} L(i,e)$ = total cell loss rate for all VCs at event e

$C_{tot}(e) = \sum_{i=1}^{n} C(i,e)$ = total no. of cells queues for all VCs at event e

Using this notation, the balance equation for the queue is:

For each VC:

$$I(i,e) = O(i,e) + Q(i,e) + L(i,e)$$

and for the queue as a whole:

$$I_{tot}(e) = O_{tot}(e) + Q_{tot}(e) + L_{tot}(e)$$

4.2.3.1.2 The Empty Queue

If $I_{tot}(e) \leq O_{max}$ and $C_{tot}(e) = 0$, then for any events that do not change this condition:

$$O(i,e) = I(i,e) \quad \text{and} \quad Q(i,e) = L(i,e) = 0$$

4.2.3.1.3 The Non-Empty Queue

Consider an event $e=j$ that arrives at the queue when $C_{tot}(j)=0$ and which causes changes in the input rates of one or more VCs such that $I_{tot}(j)$ becomes greater than O_{max}. Each VC's output and queueing rates will be directly proportional to the input rate. Therefore the output rate and queueing rate of the i^{th} VC will be:

$$O(i,j) = \frac{I(i,j).O_{\max}}{I_{tot}(j)}$$

$$Q(i,j) = \frac{I(i,j).(I_{tot}(j) - O_{\max})}{I_{tot}(j)}$$

The next event will either be a state change because the queue has filled up, or a change in the input rate of one or more VCs. If this occurs at time $t(j+1)$:

$$C(i,j+1) = Q(i,j).(t(j+1) - t(j))$$

$$C_{tot}(j+1) = Q_{tot}(j).(t(j+1) - t(j))$$

In the general case at time t(e), the equation for the total number of cells queued becomes:

$$C_{tot}(e) = Q_{tot}(e-1).(t(e) - t(e-1)) + C_{tot}(e-1)$$

If $Q_{tot}(e-1)>0$, then the queue becomes full, when $C_{tot}(e) = C_{max}$. If, however, $Q_{tot}(e-1)<0$, then the queue will become empty when $C_{tot}(e)=0$. Therefore the time at which the queue becomes full or empty can be determined by substituting the appropriate value for $C_{tot}(e)$ in the above equation.

Note that when $I_{tot}(e) < O_{max}$ and the queue is decreasing in size the output rate of the queue, $O_{tot}(e)$, will be equal to the server rate, O_{max}.

Events on the input of the queue will not immediately be apparent on the output if there are cells queued. Therefore, the queue will give rise to a delay in the propagation of events. The delay due to this queue will have a maximum of:

$$D_{\max} = \frac{C_{\max}}{O_{\max}}$$

For an event e=j+1 and provided that $C_{tot}(j+1) < C_{max}$, only the queueing rate will change at the instant of the event. So, for VCs with changes in cell rate the new queueing rate will be:

$$Q(i, j+1) = Q(i, j) + I(i, j+1) - I(i, j)$$

This change in input rate will manifest itself on the output at some later time by an event $e=k$. At this output event, the output rates of all the VCs must correspond to the new balance of input rates at $e=j+1$. Therefore, the new output rate for a given VC at $e=k$ due to an input event $e=j+1$ is:

$$O(i, j+1{:}k) = I(i, j+1).\frac{O_{\max}}{I_{tot}(j+1)}$$

This result is important, particularly if we are to consider concurrency in cell rate simulation, because it indicates that if an event on one input VC causes a change in the total input rate of the queue, then output events will be generated on all the VCs. One implication of this is that, in a congested simulator, events on a single VC can give rise to related events on other VCs even though there is no direct logical connection between them. Therefore single events can give rise to a whole wave of knock-on events which will propagate out across the network, from the original source queue along all VCs passing through that queue.

The time of output event $e=k$ is:

$$t(k) = t(j+1) + \frac{C_{tot}(j+1)}{O_{\max}}$$

Now consider the situation where the queue becomes full at event, *e=j+1*, i.e. $C_{tot}(j+1)=C_{max}$, and there is a change in input rate such that $I_{tot}(j+1)>O_{max}$. The queuing and loss rates are:

$$Q(i, j+1{:}k) = O(i, j+1{:}k) - O(i, j+1)$$

$$L(i, j+1) = I(i, j+1) - O(i, j+1{:}k)$$

When the queue is full, the cell rate flowing into the queue from a given VC is equal to the output rate, *O(i, j+1:k).* Any excess of *I(i,j+1)* over *O(i, j+1:k)* is lost. Note that each VC still has a queueing rate even though the total queueing rate, Qtot(j+1) is zero.

At e=k the effects of the input event at e=j+1 become apparent at the output of the queue. The output rate changes to O(i, j+1) and the queueing rate for the ith VC changes by an amount equal to the change in its output rate:

$$O(i,k) = O(i, j+1{:}k)$$

$$Q(i,k) = Q(i,k-1) + (O(i,k-1) - O(i, j+1{:}k))$$

4.2.3.1.4 Measurements of Cell Loss and Delay

The cell loss ratio (CLR) is defined as the proportion of cells input to a queue over a given time interval that are lost:

$$CLR = \frac{\sum_{e=1}^{n} (L_{tot}(e-1).(t(e) - t(e-1)))}{\sum_{e=1}^{n} (I_{tot}(e-1).(t(e) - t(e-1)))}$$

For a complete VC, this equation can be simplified to find the end-to-end cell loss ratio:

$$CLR = 1 - \frac{R_{tot}(i)}{S_{tot}(i)}$$

Where $S_{tot}(i)$ is the total number of cells entering the i^{th} VC and $R_{tot}(i)$ is the total number received at the destination.

Now, $(I_{tot}(e-1) - L_{tot}(e-1)).(t(e) - t(e-1))$ gives the number of cells arriving at the queue and being either queued or served, but not lost, between times *t(e-1)* and *t(e)*. The average delay experienced by these cells is $(C_{tot}(e) + C_{tot}(e-1) + 2) / (2.O_{\max})$. Therefore the average delay experienced by all cells in the queue (including waiting time and service time) is:

$$\Delta_{ave} = \frac{\sum_{e=1}^{n} (C_{tot}(e) + C_{tot}(e-1) + 2).(I_{tot}(e-1) - L_{tot}(e-1)).(t(e) - t(e-1))}{2.O_{\max} . \sum_{e=1}^{n} (I_{tot}(e-1) - L_{tot}(e-1)).(t(e) - t(e-1))}$$

4.2.4 Example Cell Rate Simulators

A number of cell rate simulators have been implemented. These include the LINKSIM and MACROSIM sequential simulators from Queen Mary & Westfield College and the simulators from the CEC RACE projects MIME and ICM.

4.2.4.1 Linksim

LINKSIM is a simulator that fully implements the cell rate model described above [Pitts93]. It is deliberately limited in scope in that it only models a single link queue, as opposed to a complete ATM network, because it was originally written as a verification and validation tool for the cell rate model rather than as a general network simulator. In LINKSIM, the matrix of possible queue states shown in Figure 8 is extended to include all the possible previous states of the queue. This results in a matrix of 81 transitions between states, of which only 39 are actually valid. In cell rate simulations, the simultaneous cell rate changes

that occur at the output of a non-empty queue must be processed together in order to avoid the generation of multiple resultant events with the same time-stamp (these are clearly a waste of processor time). LINKSIM is designed to efficiently manage such events by using a novel event list topology: Two extra dimensions are added to the normal sequential event list structure; these list events in terms of the place that they are scheduled to occur and the type of the event.

4.2.4.2 Macrosim

MACROSIM [Pitts90] is a cell rate ATM network simulator that implements a simplification of the above model. It represents ATM networks as the interconnection of network elements, which can be either links or nodes, and network terminations. Both the burst and call levels are modelled by this simulator and each network termination can have multiple calls in progress at any one time.

4.2.4.3 RACE Simulators

CEC RACE projects MIME and ICM have both produced cell rate simulators that use a simplified version of the above model. The MIME simulator runs on a network of Transputers [MIME91] while the ICM simulator (MADS - Multipurpose Aid for Distributed Simulation) is designed to be portable between a sequential and a parallel architecture. MADS is presented in Chapter 5 of this thesis as an example of a parallel cell rate simulator and is described in detail in [ICM92.2] and [Swift94].

4.2.5 Performance of Cell Rate Simulation

The performance of a simulation scheme can be assessed in terms of two main factors: the speed of the simulator and the accuracy of measurements taken. One way of assessing the speed of the simulation is to measure the rate of cell processing and to compare this with another benchmark simulator running the same network topology. The accuracy

can be assessed by comparing results (for example those for cell loss ratio) with those predicted by analytical methods and also by another simulator of a known accuracy.

Pitts et al have validated cell rate simulation in terms of both accuracy and speed. In [Pitts92.1] cell loss measurements in a queue obtained using LINKSIM were compared with analytical results using equivalent source models. This showed excellent correlation between the two approaches. Comparison of the cell rate results from LINKSIM with those obtained from the cell level simulator MICROSIM [Pitts91][Pitts92.2] also displayed a good correlation although the measured cell loss is slightly less with cell rate modelling than with cell level modelling. This is because the simultaneous arrival of cells at the queue is not modelled and so the probability of achieving buffer overflow is reduced.

When comparing the speeds of the two simulators, cell rate modelling was found to give a significant speedup over cell level modelling. In the experiments described in [Pitts91] the speedup varied between 1.1 and 13.3, depending on the burst length of the source. An increase in speedup seems to correspond most strongly with an increase in burst length; this is as expected as the modelling tends to cell level as the burst length decreases.

4.3 Parallel Simulation

4.3.1 Overview

Parallel simulation (also termed distributed or concurrent simulation) is an accelerated simulation technique that has received much attention in recent years. Many authors have also reviewed the literature on this subject in great detail including Hind [Hind94], Phillips [Phillips92] and

Richter & Walrand [Rich89], and so only a brief overview is given in this thesis.

The aim of parallel simulation is to provide speedup by exploiting the inherent parallelism in a system model. This is done by distributing the elements of the simulation model across a multi-processor computer. This has a particular attraction for the simulation of telecommunications networks because they are naturally distributed systems in which one would intuitively expect much concurrent activity to exist.

4.3.2 Decomposition

Decomposition is the process by which parallelism in the simulation model is identified and exploited in order to maximise the utilisation of resources within a multi-processor computer.

For parallelisation to be effective, the decomposition needs to satisfy a number of criteria in order to ensure optimal performance of the multi-processor system. These criteria include:

a) *Coarse Granularity and Minimal Communication.* Because communication between processors is usually slow compared with the processing of data, unnecessary inter-processor communication should be minimised. Aiming to keep processors busy performing local activities improves the proportion of time spent fulfilling some useful operation. The use of decoupling buffers can help reduce processor time spent on inter-processor communication.

b) *Balanced Workload.* The workload of the system should be distributed so as to minimise the idle time of the processors. The decomposition should also be tailored to the available resources such that, for example, functions involving a large amount of floating point mathematics should be placed on processors with integral floating point co-processors.

Note that each decomposed element of the simulation is generally assigned to a *process,* with one or more processes be placed on each processor. In general, for maximum parallelism it is assumed that there is one process per processor and hence the terms are used interchangeably in this thesis.

Various ways of decomposing a simulation problem have been identified, broadly classified by Phillips [Phillips91] into *Functional Parallelism* and *Spatial Parallelism*. Richter & Walrand [Rich89] describe five distinct methods: *Parallelising Compilers*, *Distributed Experiments*, *Distributed Language Functions & Distributed Events* (both forms of *Functional Parallelism*), and *Distributed Model Components (Spatial Parallelism).* A sixth method, *Time Decomposition,* is reviewed by Hind [Hind94].

4.3.2.1 Parallelising Compilers

This approach involves simply applying an appropriate parallelising compiler to a sequential simulation program. Parallelising compilers try to find sections of code that can be run concurrently on separate processors. Hence they generate a parallel version of existing sequential code. This approach has the advantage that much of the code for existing sequential simulators can be reused, and that the details of the parallelisation process are hidden from the user. Therefore both the learning curve for a developer migrating from a sequential computer architecture to a parallel architecture and the development time for parallel simulator itself can be minimised. However, the Parallelising Compiler approach ignores the structure of the problem itself and these compilers are often unable to fully exploit the available parallelism. Hind [Hind92] has applied a parallelising compiler to the simulation of a circuit switched telecommunications network and found that the speedup that could be achieved was disappointing.

4.3.2.2 Distributed Experiments

Long simulation runs are necessary in order to obtain statistically significant results. However, significant results can also be obtained by running shorter independent replications of the simulation on separate processors and averaging the results at the end. This approach can be extremely effective because the only co-ordination required between the processors is when the results are averaged. However, the topology and scale of the network simulation is limited because in many multi-processor computers each processor does not have sufficient local memory to support a complete simulation, particularly of a large network. Furthermore, this approach in unlikely to be of use if the simulation is required for TMN studies because of the requirement for dynamic interaction between the simulator and the network management system.

4.3.2.3 Spatial Decomposition (Distributed Model Components)

This involves decomposing the network model into loosely coupled components, or domains, containing a number of components; the components are assigned to distinct processors for concurrent execution. In the simulator designed by RACE project 2059 ICM, the network model is decomposed such that each node is mapped to a single processor; Phillips [Phillips91] maps a complete domain to each processor

Spatial decomposition has received much attention over recent years. It is the most intuitive form of decomposition since it would appear to be relatively easy to model an inherently distributed system on a distributed computer. Each component must maintain its own local clock measuring the elapsed simulated time and must have a list of scheduled future events that will occur at this location. A change in some state of the system is usually associated with an event, and a local event may affect the state of a remote system component. Therefore communication between system components is required such that the simulation of particular components progresses only when it is valid to do so. This has given rise to many problems in maintaining event causality relationships

in spatially decomposed simulations, and it is in overcoming these problems that much work has concentrated.

4.3.2.4 Functional Decomposition Methods

These involve taking a simulation scheme, identifying the functions that operate on data and data structures and separating them into a number of component processes. Loosely coupled components that are largely independent of each other are isolated so they can work concurrently if placed on distinct processors.

Two main forms of functional decomposition have been identified: *Distributed Language Functions* and *Distributed Events.*

4.3.2.4.1 Distributed Language Functions.

Here the support functions of the simulator are assigned to individual processors. Simulators typically include many support functions that can be distributed in this way, including random variable generation, statistics processing, and I/O and file manipulation. This form of parallelism has the advantage that it avoids deadlock problems, and that it is transparent to the user. Comfort [Comf82][Comf83][Comf88] has shown that some speedup can be achieved using this approach. For example, he has used Transputers to simulate a queueing system, obtaining a maximum efficiency of 60% on a two processor computer (one of the processors handled the random number generation). In Comfort's approach, significant speedup is obtained when only a few processors are used but follows a pattern of diminishing returns when further processors are added. Clearly in any simulation language there is a limit to the number of functions that can be effectively placed on other processors.

4.3.2.4.2 Distributed Events

The distributed events approach involves maintaining the centralised event list of traditional sequential simulators. When a processor becomes available it is responsible for processing the event at the head of the list.

This approach is more appropriate to shared memory multi-processor systems because of the large communication overhead between the processor managing the event list and the other processors in the simulator. Phillips [Phillips91] implemented a cell level ATM network simulator on a network of Transputers using this approach and found that some speedup was achievable due to the additional functional parallelism that is exploited. However, as the number of domains is increased the speedup is limited by the build up in work load of the processor responsible for managing the event list.

4.3.2.5 Time Decomposition

Time decomposition represents a more recent approach to the identification of parallelism in a simulation model. Whilst the most obvious form of decomposition relies on concurrent activity in spatially separated objects, this form of decomposition attempts to exploit parallelism in the time domain. A brief survey of recent work in this field is given by Hind in [Hind94].

Chandy et al [Chand89.2] introduce a concept known as *space-time*. In this, the behaviour of a system is represented as a two dimensional graph. A simulation represents a specific rectangle in the space/time plane. They propose an algorithm that induces parallelism by partitioning this region into regions of space/time. Reiher et al [Reih91] have extended this concept to encompass time parallelism that partitions the simulation into phases along the time axis. This can lead to significant speedup because of better load balancing than in spatially decomposed simulators.

In [Amm92], Ammar and Deng describe an approach that combines time warp simulation techniques (see Section 4.3.5) with time scale decomposition. They demonstrate that a simulation system may be decomposed into *fast event* sub-models and *slow event* sub-models. Fast event sub-models are those in which events occur on a relatively fast timescale, whereas slow event sub-models process events on a relatively

slow timescale. Therefore, in fast event sub-models the local simulated time advances by a smaller amount for each event than in slow event sub-models. The fast event sub-models can be simulated concurrently because interactions between these sub-models are relatively weak in comparison with the interactions between the processes that are contained within them. However, because the interactions between sub-models are ignored, errors are introduced into the simulation.

Of particular relevance to the subject of this thesis is a study by Nikolaidis et al [Nikolai93]. This describes the time-parallel cell rate simulation of an ATM multiplexer. Their approach is based on the exploitation of time parallelism in the cell rate change arrival process at the multiplexer. Each processor within a multi-processor computer is responsible for simulating a particular time interval. At the start of the simulation, the simulator generates a discrete time Markov chain that represents all of the possible states of a set of ON/OFF cell-rate traffic sources at the input to a queue. This enables a number of time division points to be identified at times of guaranteed queue overflow and also at times of guaranteed queue 'underflow' i.e. when the queue length is zero. The system states at either side of these are considered to be independent. Therefore, regions of time between these time division points can be simulated concurrently. Nikolaidis et al demonstrate that the granularity of the decomposition is limited by the number of time division points, which is often much greater than the total number of processors available! An implementation of this method on a 16 processor computer simulates cell arrivals at a rate of "up to five orders of magnitude greater than an efficient conventional sequential simulation".

Despite the fact that there are a number of studies into time parallelism described in the literature, as yet there are few practical implementations of simulators that exploit time parallelism. Because of the great variety of ways in which simulations evolve in time, these schemes are highly dependent on the specific application. Therefore, the scope for general

purpose simulation schemes based on this form of decomposition is extremely limited. Indeed, Fujimoto et al [Fujim95] state that:

> *"Time-parallel simulation is more of a methodology for developing massively parallel algorithms for specific simulation problems than a general approach for executing arbitrary discrete-event simulation models on parallel computers. Time parallel algorithms are currently not as robust as space-parallel approaches because they rely on specific properties of a system being modelled."*

Furthermore, because the simulated time in each processor differs substantially, there is little scope for coupling a time-parallel simulator to real time management or control systems. The use of time-parallel simulators is therefore restricted to determining specific performance metrics of particular elements of a network.

4.3.3 Consistency in Concurrent Simulation Schemes

In a distributed simulation some mechanism is required to ensure that event causality relationships are never violated and hence the simulation remains consistent with the real system being modelled. It is possible that, due to load variations between different processors, the simulated time will not advance at a constant rate across the whole network. For example, consider the model illustrated in Figure 9.

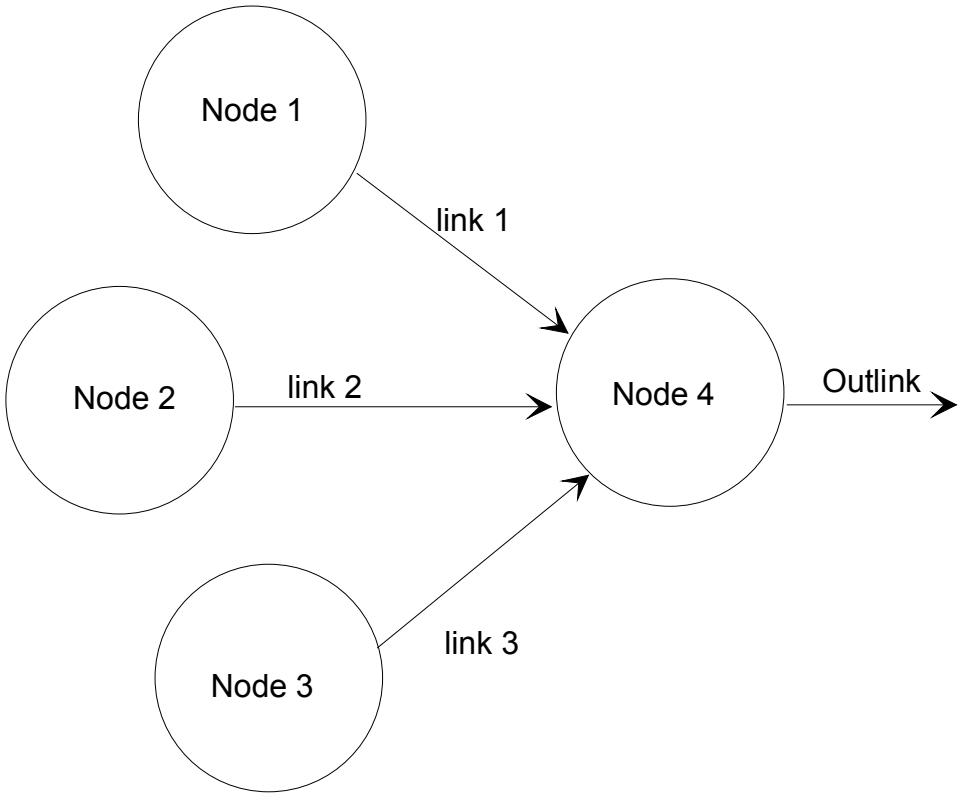

Figure 9 Example of the need for Consistency

Events are sent along links 1, 2 and 3 and are processed at node 4, giving rise to events on the outgoing link of node 4. Consider an event on link 1 with time-stamp t_1. In order to process this event node 4 must know that no events will arrive on links 2 or 3 with a time-stamp of less that t_1. If an event arrives with time-stamp $t \leq t_1$ after this event has been processed then the consistency of the simulation will have been broken.

Deadlock is a situation that can occur when one or more processes cannot continue without violating consistency and so enter a continuous wait state.

4.3.4 Lookahead

Lookahead is a key characteristic of a simulated system. It is defined as the ability of a process to predict its future behaviour [Lin90]. In a parallel

telecommunications network simulation this equates to the ability of a model to process all events currently scheduled up to a given time in the future without violating consistency. Consider once again the above example. The concept of lookahead means that node 4 is able to process events on link 1 from the current time to some time in the future knowing that they will not be invalidated by historical events on the other links. A further example of lookahead is in a queue with a minimum service time of δt. When it transmits a cell at time t it knows that it will not transmit another before time $t+\delta t$. Therefore, a connected process could process all of its scheduled events autonomously up until this upper time bound, knowing that no historical events from the queue process would later invalidate this.

In a cell level simulation of an ATM network, where cell level queueing is modelled, the minimum lookahead must be the smallest service time of any component within the network model. This is called the network-wide lookahead value. Events in spatially separate parts of the network must be independent if their times differ by less than this value. Hence they can be processed concurrently. However, the network wide lookahead is a very small value and many parallel simulation schemes aim to maximise the available lookahead. Nicol [Nicol88] describes a scheme using a *future list* of pre-sampled service times to give enhanced lookahead at a queue. This was found to be most effective under heavily loaded conditions.

In a cell rate simulation the discrete nature of the arrival and departure of cells from queues is not modelled and hence the identification of a network-wide lookahead is not so straightforward. This is one limitation on the available parallelism in a cell rate simulation model.

Two main forms of lookahead have been identified: *explicit* and *implicit* lookahead. *Explicit lookahead* [Lin89] is when the lookahead is known before the start of the simulation and is invariant. *Implicit lookahead* is

variable but can never-the-less offer performance gains even when there is no explicit lookahead.

Lookahead ratio is defined as the mean message time-stamp increment divided by the lookahead. It is this value, and not the absolute value of lookahead, that determines the scope for concurrency in a simulation. For maximum concurrency this value must be as small as possible. This is because maximum concurrency is obtained when as many events as possible have time-stamps that are within the current lookahead of a process.

4.3.5 Conservative and Optimistic Synchronisation

The purpose of synchronisation within a parallel simulator is to maintain consistency. Two fundamental policies exist: *conservative* and *optimistic* synchronisation, based on the way in which a decision is made as to whether or not to process a scheduled event.

A general definition of conservatism is given by Nicol [Nicol90] as *methods which prevent any processor from simulating beyond a point at which another processor might affect it.*

Conservative synchronisation means that a process is able to process events on its inputs knowing that they will not be invalidated by any future historical events that may arrive at those inputs. This requires that, in the case of Chandy-Misra type spatially decomposed simulators, messages from one process to another process are transmitted in chronological order according to their time-stamps and that a process must receive a message on each of its inputs before it can proceed. This could cause deadlock since a process could wait on an empty input for a message that is not due until after the times of all the other messages pending on its inputs. Misra describes a scheme using *null messages* to avoid this form of deadlock [Misra86]. In this scheme, null messages are

sent between processes to announce the absence of simulation related messages.

Optimistic schemes allow a process to process messages as far forward in time as it wants without concern for any historical messages that may arrive from other processes at some future time and invalidate this activity. This means that a process in a spatially decomposed simulator, for example, is able to compute the result of message arrivals even though some its inputs may be empty. In order to avoid inconsistent behaviour, optimistic simulators must be able to *rollback* [Gaf88] in time to correct any erroneous actions whenever a message is received that invalidates some past processing, in order to correct any erroneous actions. *Antimessages* must also be sent from this process to other connected processes in order to correct for any output messages generated as a result of this inconsistent behaviour. Optimistic simulators must keep a record of their state such that rollback to a previous state is possible.

A practical implementation of optimistic synchronisation is the *timewarp mechanism* [Jeff85]. This is based on the concept of *virtual time*, which is synonymous with simulated time. In timewarp an error in consistency is detected if a message is received that has a time-stamp earlier than the time of the last message, and hence earlier than the time of the process' local clock. These messages are known as *stragglers*. The process will then undo all of the events that have been processed incorrectly by rolling back to the time of the straggler. When this occurs, the process state returns to the last correct old state and antimessages are sent to cancel the effects of any messages sent from this process. These antimessages will cause rollback in other processes if the events that these processes were sent have already been processed. This continues across the simulation until the effects of the causality error have been removed.

The type of events generated within a simulator impacts upon the synchronisation mechanism. *Conditional* events are those that are time-

variant [Chandy89.1]; the time at which this activity may happen is not explicitly defined and is dependent on other events. *Unconditional* events are time-invariant and will be processed at a predetermined time irrespective of other events. Conservative schemes will never process conditional events.

4.3.6 Synchronous and Asynchronous Synchronisation

Synchronisation schemes can also be categorised according to the way in which the advancement of simulated time across the simulation is controlled.

In synchronous approaches a global record of the elapsed simulated time is maintained. This clock can either be centralised, or distributed. The global clock can be advanced according to a number of schemes. For example, to the next event time for all processes, or in a series of discrete ticks as in the timestepping approach (see Chapter 5) or Lubachevsky's Bounded Lag approach [Lub88]. A review of synchronous approaches is given in [Hind94].

In asynchronous approaches, no global record of simulated time is maintained and different processes may, at any given instant, have differing elapsed simulation times. Conservative asynchronous simulators are possible through the exploitation of lookahead. Asynchronous simulation has received the greatest attention in the literature because of its potentially high performance. This is because processes spend less time waiting for other processes than in synchronous synchronisation.

4.3.7 Practical Parallel Simulators

Many of the schemes described above have been applied in implementations of parallel ATM simulators. In the ideal situation, the speedup attained is proportional to the number of processors used.

However, in practice this is rarely the case due to such factors as overheads in synchronisation and sub-optimal load balancing.

Both Hind [Hind94] and Phillips [Phillips91][Phillips92] have built parallel cell level ATM simulators using networks of transputers. Other practical implementations have already been described above. In summary, the results of Phillips are typical. He implemented centralised event list, distributed event list, and distributed event buffering schemes and found that the most effective technique was distributed event buffering using the Chandy-Misra null messages deadlock avoidance scheme. A detailed assessment of these simulators is given in [Phillips92].

4.3.8 Parallel Computer Architectures

Although the parallel decomposition paradigm and time synchronisation scheme will be partially influenced by the traffic modelling technique and the characteristics of the network model, the precise details of the computing platform on which the simulator will run are also an important consideration. Multiprocessor computers are available with a wide range of architectures. These start from simple multiprocessor workstations whose operating systems do little more than map independent software processes onto different central processing units that share a common set of system resources (e.g. dual-Pentium[2] PCs running Microsoft Windows NT[3]). A next step is the general purpose 'truly parallel' machine, often based on CPUs that are specifically designed for a parallel environment and that control significant amounts of their own private memory and other system resources (e.g. systems based on the INMOS Transputer[4] and INTEL i860). The most specialised systems employing digital signal processors for applications such as image processing and pattern recognition.

---

[2] Pentium is a trademark of Intel Corp.

[3] Microsoft & Windows NT are trademarks of Microsoft Corp.

Flynn [Flynn66] has classified computer architectures by considering parallelism in the instruction and data streams. This has resulted in the following four categories:

1. Single Instruction Stream, Single Data Stream (*SISD*)
2. Single Instruction Stream, Multiple Data Streams (*SIMD*)
3. Multiple Instruction Streams, Single Data Stream (*MISD*)
4. Multiple Instructions Streams, Multiple Data Streams (*MIMD*)

Note that this classification is very rigid and in practice many computers will fit in more than one category.

The first category, Single Instruction Single Data Stream, encompasses most uni-processor machines. Here there is no parallelism (or, at least, no attempt to exploit any potential parallelism) in the instruction or data streams.

Categories 2, 3 and 4 attempt to exploit parallelism in the instruction and data streams. The most common of these are SIMD and MIMD computers. MISD computers are rarer; Hennessy & Patterson [Henn90] suggest that these are limited to super-scalar, VLIW (*very long instruction word*), decoupled and systolic architectures.

In SIMD machines many functional units are invoked by a single instruction pointed to by a single program counter. This has the advantage that all parallel execution units are synchronised. SIMD architectures reduce the cost of the processor's control unit per execution unit since this is effectively 'spread' over many execution units. They also reduce the size of the program memory since only one copy of the code is simultaneously executed (MIMD machines often require one copy to be held per processor). SIMD computers are similar in many respects to

---

[4] Transputer is a trademark of INMOS Ltd

vector processors. In practice they will have a mixture of both SISD and SIMD instructions, each SIMD instruction being broadcast to all of the execution units, each of which has its own set of registers. SIMD is most effective in processing code that contains many arrays and loops where there is massive data parallelism.

The most common form of multiprocessor computer architecture is the MIMD machine. These machines employ many CPUs in an attempt to achieve speedup over uni-processor computers and follow the philosophy that a powerful computer can be built simply by connecting together many existing smaller ones. The speedup thus accomplished is simply a function of the number of processors. In practice such an ideal situation is difficult to achieve. However, MIMD computers do have some distinct advantages over those with an SISD architecture, including higher absolute performance and higher reliability/availability through redundancy [Henn90].

Two principle classes of MIMD machine exist, the classification depending upon how the processors share information. In a s*hared-memory* computer the processors communicate by passing variables using a block of memory that can be accessed by all of the processors. Whilst this may appear to be very fast method of processor intercommunication, the ultimate speed will be limited by the bandwidth of the bus connecting the processors to the shared memory and by the number of processors that can simultaneously access the memory. These problems can be reduced by using local caches on each processor. However, cache coherency then becomes an issue. A typical shared memory scheme is shown in Figure 10.

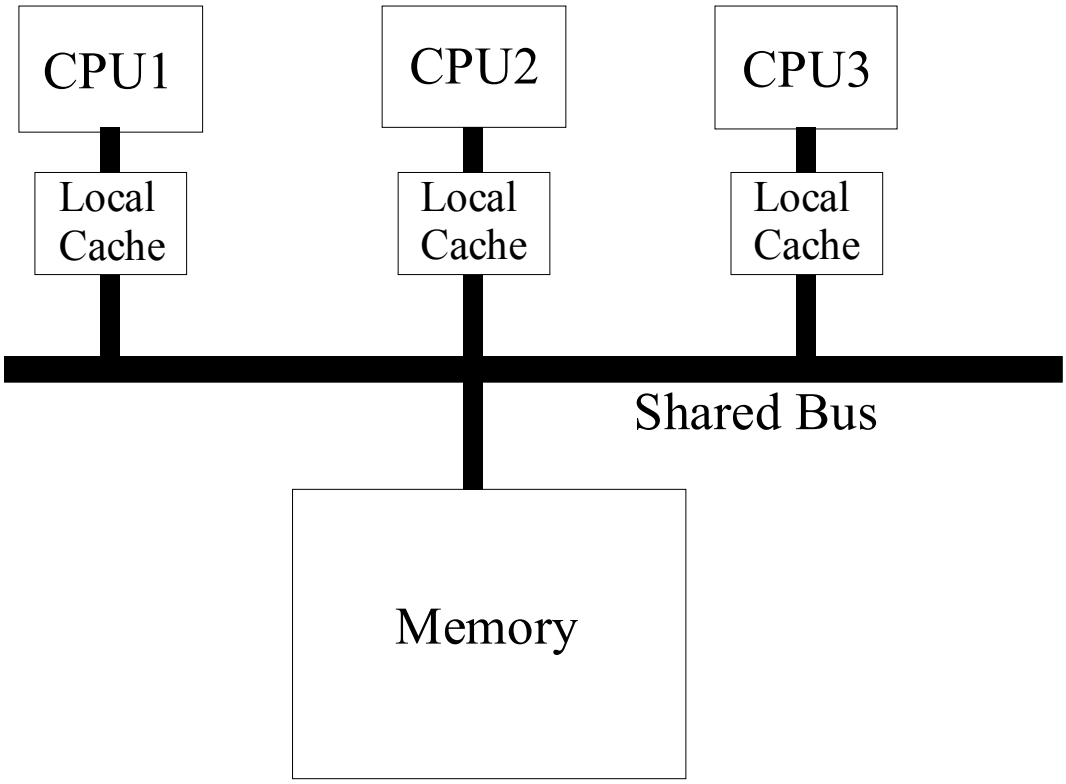

Figure 10 Shared Memory Architecture

The alternative method for sharing data between processors is to pass discrete messages between the processors. Here there is no global shared memory; memory is distributed amongst the processors. The performance of such a computer is limited by the efficiency of the message passing and routeing scheme, which often decreases as the number of processors is increased. Messages can be passed between processors using a network of busses, or, in the most extreme example, the processors can reside in separate computers and communicate via a local area network. A distributed memory architecture is illustrated in Figure 11.

Parallel computer architectures are reviewed in greater depth in [Fost95].

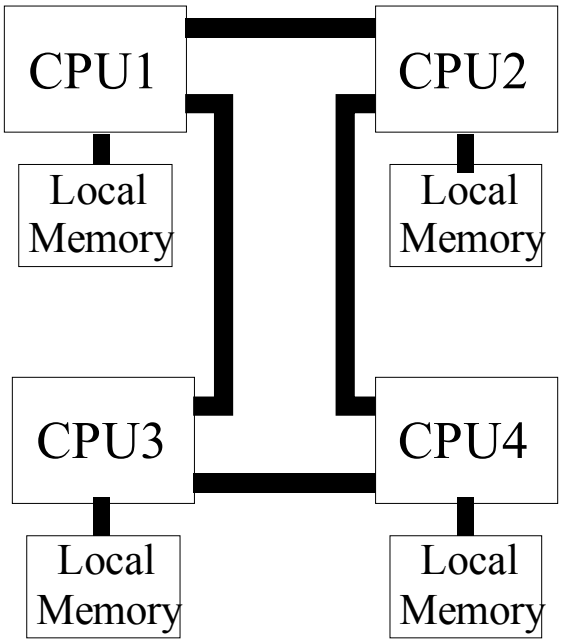

Figure 11 Distributed Memory Multiprocessor Architecture

4.4 Summary

This chapter has outlined two methods for accelerating the simulation of ATM networks. Cell rate simulation attempts to provide speedup by reducing the number of events that have to be processed relative to cell level simulation. This has proven to be an extremely effective technique. Parallel simulation attempts to provide speedup by exploiting the inherent parallelism in a network simulation. In order for this to be effective, a number of basic criteria must be satisfied: these include the fact that the communication between processors must be minimised and processing concentrated on local tasks; the workload must be evenly balanced across the network of processors. Many different decomposition and synchronisation schemes are described in the literature which all attempt to exploit the inherent parallelism in the network model. Whilst parallel simulation has shown some promise in terms of offering speedup over sequential simulators, many schemes have been implemented with

only limited success since there is often a conflict between the aforementioned criteria. Where these problems are overcome, significant speedup can be achieved.

In the next chapter, the application of a synchronous parallel simulation scheme to the cell rate modelling of ATM networks is described.

5. Timestepping - A Synchronous Parallel Synchronisation Scheme

5.1 Outline of Timestepping Simulator

Timestepping is a simple scheme that attempts to overcome the problem of maintaining time synchronisation across a parallel simulator while minimising inter-processor communication overhead by providing a common clock to all the processors. This is achieved by only allowing the local time in each node model to advance by some fixed period, or *timestep*, before all local times across the simulation are synchronised to some global time.

The timestepping approach is similar to Lubachevsky's *Bounded Lag* technique [Lub88]. Lubachevsky defines a *bounded lag restriction* such that, if two events e_1 and e_2 , that occur at times $\tau(e_1)$ and $\tau(e_2)$, are processed concurrently then:

$$\left|\tau(e_1) - \tau(e_2)\right| \leq B$$

Where B is the *bounded lag restriction*, and is a known constant such that $0 \leq B < +\infty$. The timestepping scheme considered here represents a novel application of this concept to cell rate ATM simulation.

Timestepping is the synchronisation scheme that has been implemented in the cell rate simulator developed by the RACE project R2059 Integrated Communications Management (ICM). Although the ICM Simulator [Swift94] has initially been implemented on a single processor computer, it was designed to be relatively easy to port to a parallel machine. Therefore, the simulator has been designed such that the modules modelling the network nodes can be run on a single processor or

distributed across a multi-processor architecture. Note that timestepping has also been implemented in the SPROG[5] ATM simulation tool developed by the author and described later in this thesis

Cell rate modelling is the chosen method for modelling the traffic because, in sequential simulators, it has been shown that significant speedup can be achieved when compared with cell level modelling [Pitts91].

A spatial decomposition paradigm is used. In this, different nodes in the network are modelled by separate processors. Each node model consists of a *traffic generator* and a *node code* module. Events (such as cell rate changes and signalling messages) are transferred between models by kernel software (this modifies the arrival time of the events according to any link delays), and are placed in a time ordered queue (*stream queue*) for processing at the destination model. Note that a *timer queue* is also provided for the self-queuing of locally generated events. This is illustrated in Figure 12.

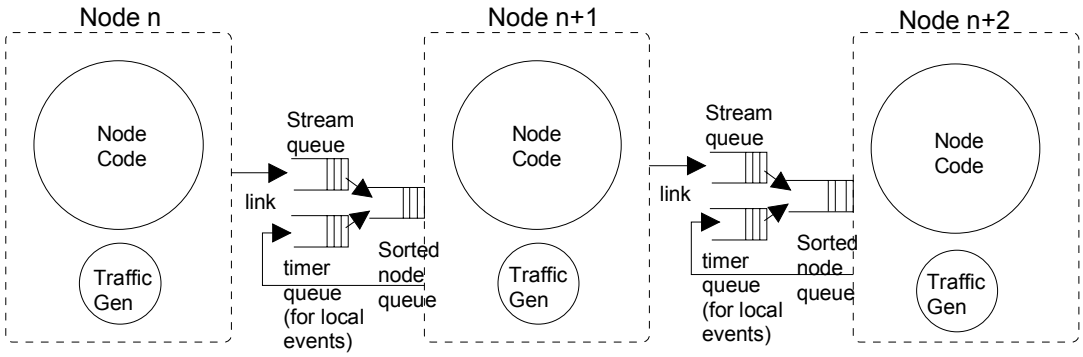

Figure 12 Simplified node architecture

The traffic generator for each node is called at the beginning of each timestep. This generates all of the traffic events of any users present at that node for that timestep plus the first event to occur after this timestep, hence ensuring that initial traffic generation is not affected by any errors

[5] Simulation PROGram

introduced by timestepping. Once the traffic generators for all of the node models have been run, the kernel will sort in time and priority order all of the events for the current timestep from all the stream queues and the timer queue at each node. These events are placed in the sorted node queue at each node. The main node code of each model will then be called for each event in that model's sorted node queue. This sequence of operations is illustrated in Table 2 [Swift94].

```
FOR each timestep:

     evaluate timestep size
     FOR each node:
        - run traffic generator
        - sort in time and priority order events that
          arrive at this node
     END_FOR_LOOP
     FOR each node:
        retrieve local_time
        WHILE local_time < end_step_time AND events to
        process
        - if event_time > local_time
          local_time = event_time
        - process event
          This processing may generate other events that
          are sent to other nodes. They cannot arrive at
          other nodes before the start of the next
          timestep.
     END_FOR_LOOP
END_FOR_LOOP
```

Table 2 Sequence of Operations for Timestepping Simulator

Because the sorted node queues are built from the timer and stream queues at the start of each timestep, only events received in previous timesteps will be processed during any given timestep by a given model. This means that any events sent from one model to another cannot be processed by the destination model until the start of the next timestep at

the earliest. Hence timestepping can introduce errors in the effective arrival times of events at models. These errors can have serious implications for the accuracy of the simulator.

5.1.1 Grouping of Simultaneous Events

In cell rate simulation, cell rate changes on a single VC on the input to a non-empty queue give rise to simultaneous cell rate changes on all VCs on the output of the queue. These cell rate changes must be processed together when they are delivered to the input of any subsequent queues in order to ensure that spurious output rate changes from that queue are suppressed and to ensure that statistics generated are correct. An in-depth analysis of the reasons for, and the effects of this on the performance of cell rate simulators is presented later in this thesis.

The ICM implementation of timestepping deals with simultaneous cell rate changes using a technique known as *grouping*. If events inserted in the stream queue of a given node are found to have the same scheduled arrival time, then they will be labelled as being at the start, the end, or within a group. This enables node models to identify whether the current cell rate change event forms a part of a group. Only when all of the cell rate changes associated with a given group have been delivered to the node model is the overall effect of these events on the output of the model calculated.

5.2 Performance Implications of Timestepping

Consider a simple ON/OFF traffic source feeding bursts of greater than one timestep in length along a single finite delay link to a switch, SW1. This switch has one output port that is connected to a second switch, SW2, by a single finite delay link. The link delay is less than the timestep and the switches have zero buffer size. Suppose that a single event

representing the leading or trailing cell rate change of a burst is sent by the traffic source. Since this event will be generated by the traffic generator in the source it will arrive at SW1 at the correct time. However, output events generated at SW1 during this timestep cannot be processed in SW2 before the start of the next timestep. Therefore, events sent from SW1 during the current timestep will be delayed until the start of the next timestep. This situation is illustrated in Figure 13.

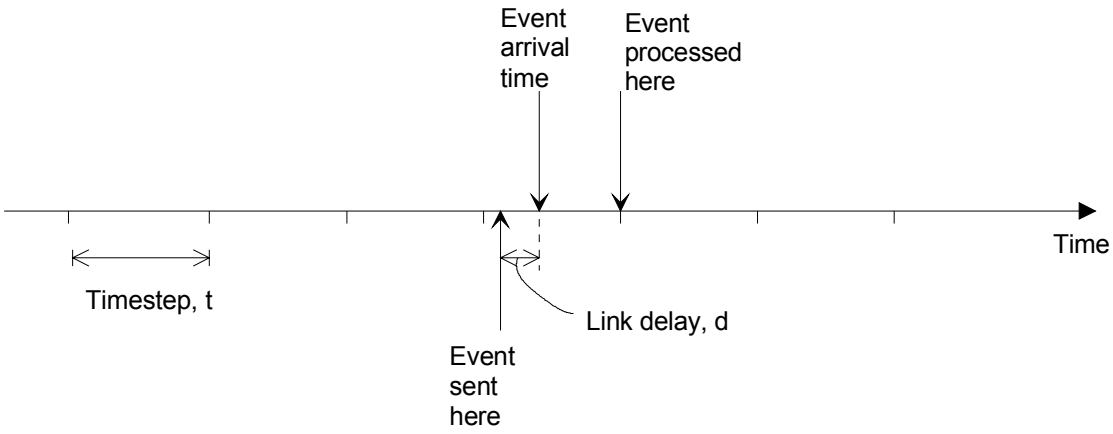

Figure 13 Event Delay Due to Granularity of Timestep

If the output of switch SW2 is connected to another node model, then output events generated at SW2 due to events on the input of the switch cannot arrive at this next node until the start of the next timestep. Hence, for a timestep greater than the link delay, event arrivals will become synchronised with the boundaries of timesteps. Distortion of the traffic due to the synchronisation of events with timesteps is shown in Figure 14.

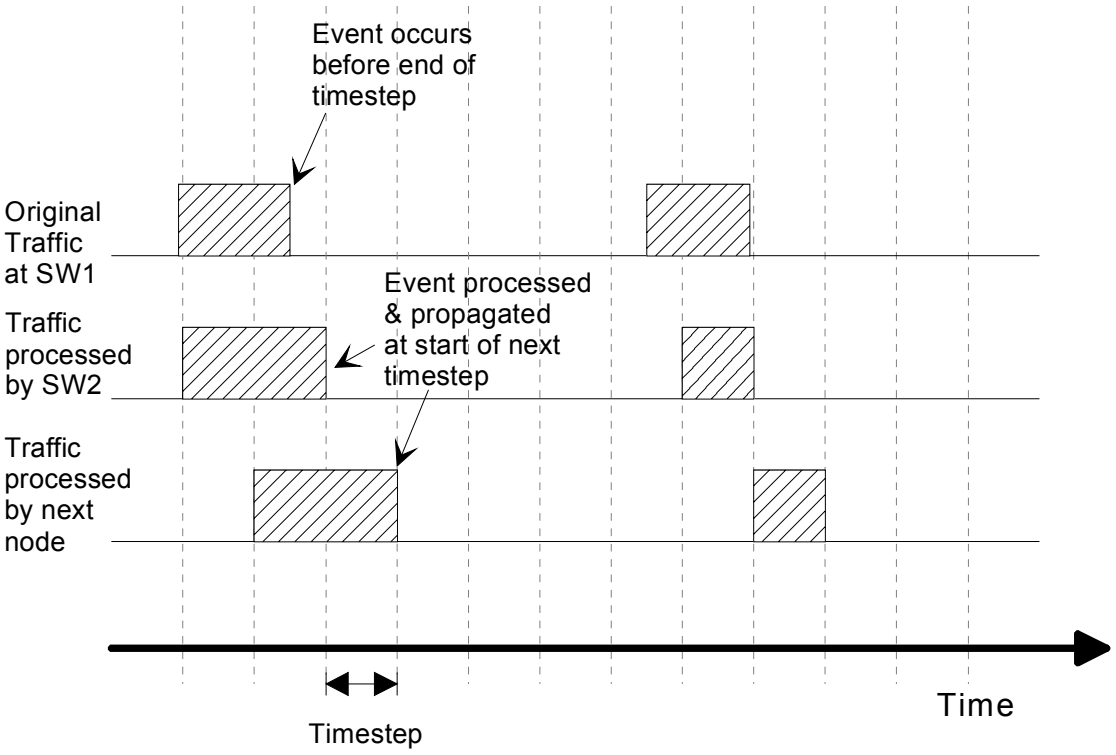

Figure 14 Event Synchronisation with Timestep Boundaries

It can be seen from the above figure that, for situations where burst length is greater than one timestep, the burst length will be adjusted to some integer number of timesteps. Whether it is rounded up or down will depend on the precise timing of the front and tail events of the burst in relation to the timestep boundary.

Now consider a situation where the timestep is larger than the burst length. If the front and tail events of the burst lie within the same timestep then both edges of the burst will be processed at the beginning of the next timestep by the receiving node simultaneously (as shown in Figure 15). Therefore the burst will be reduced to zero length and will be lost.

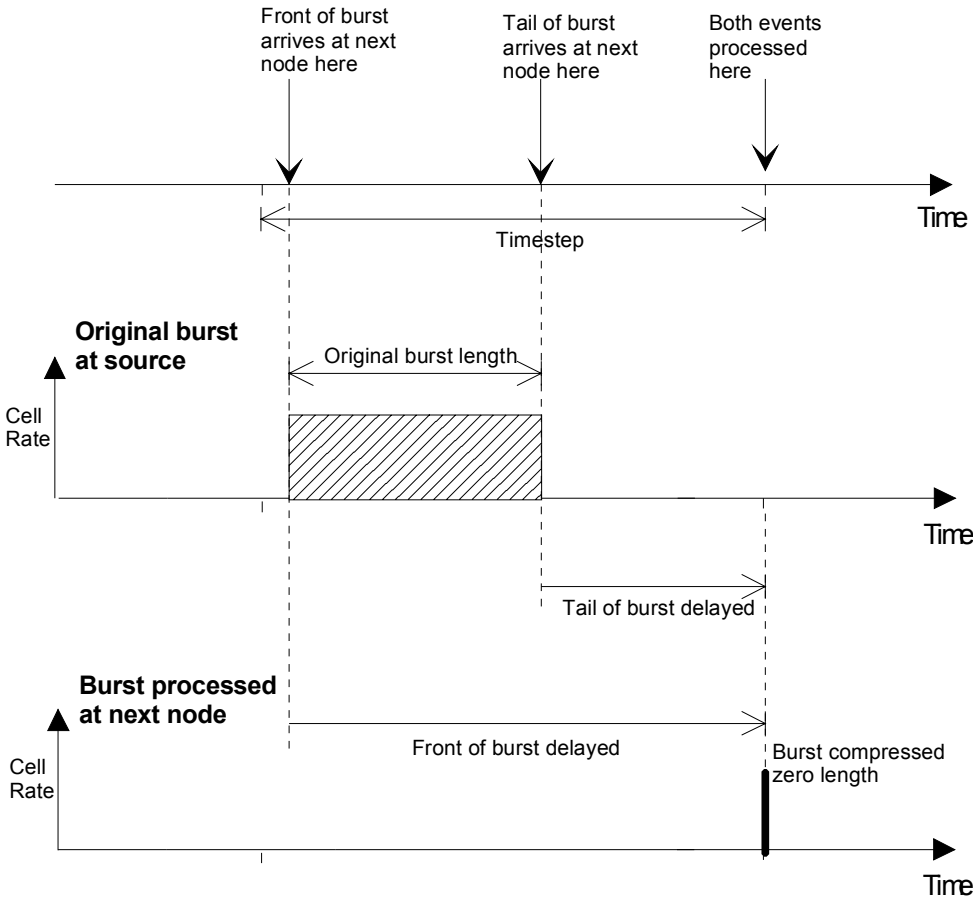

Figure 15 Effect of Timestep on Burst Length for Bursts Shorter than 1 Timestep

If the burst happens to straddle a timestep boundary then the processing times of the front and tail events of the burst will be delayed until the start of the next timestep, effectively stretching the burst to one timestep in length, as shown in Figure 16.

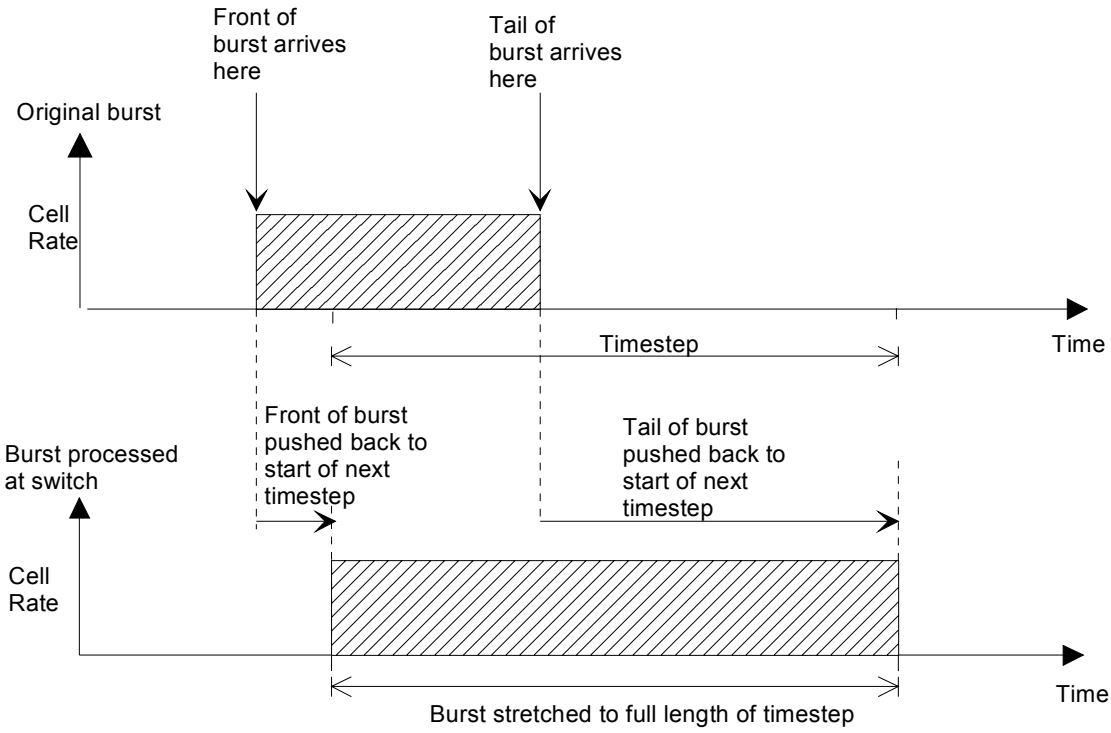

Figure 16 Burst Stretching Due to Timestep Effect

Hence, for situations where the link delay is small compared with the timestep, the following distortions of the traffic will occur:

- Events are synchronised with timestep boundaries.
- If burst length > timestep, burst length is adjusted to an integer number of timesteps ('burst length quantisation').
- If burst length < timestep and the burst starts and finishes within the same timestep, then the burst is lost ('burst swallowing').
- If burst length < timestep and the burst straddles a timestep boundary, then the burst length is increased to that of the timestep ('burst stretching').

In the above discussion it is assumed that the link delay is less than one timestep. However, if the link delay is greater than the timestep then cell rate changes will arrive within the next timestep (as opposed to the

current timestep) and hence will be sorted correctly and processed at the appropriate time by the receiving node. Therefore, the events will not be synchronised with the timestep boundaries and no burst stretching or swallowing will occur.

5.3 Experimental Study of Timestepping Performance

For the purposes of this study a timestepping ATM network simulator running on a single processor Sun workstation was designed [Bocci94]. The simulator was based on an early version of the RACE ICM simulator. The network configuration used is shown in Figure 17.

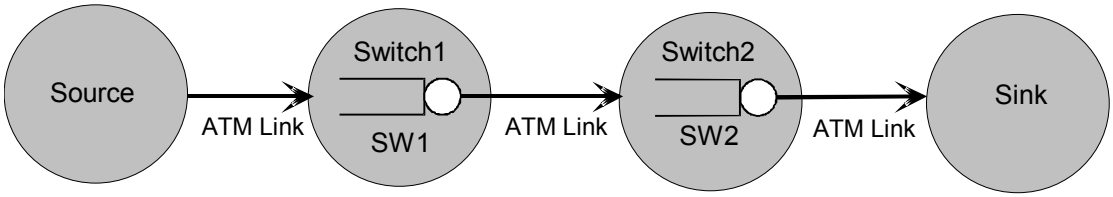

Figure 17 Network Configuration Used for Timestep Studies

Five virtual channel connections (VCCs) were set up from the source along the ATM links through the switches to the sink. The switch models had zero buffer size and therefore any cell rate input to the switch in excess of its server rate was lost. Despite the fact that the full cell rate queue was not modelled, using a basic zero-buffer size model allowed the study of the essential effects of cell rate modelling while minimising the complexity of the switch model. The traffic was generated in the source using a Wichman-Hill random number generator with exponential statistics. The traffic was chosen to be similar to a Virtual Private Network consisting of a number of Ethernet Local Area Networks connected by an ATM link. The switch server rates were 150,000cells/s and 75,000cells/s for Switch 1 and Switch 2 respectively. The source was a simple ON/OFF type with the

following parameters: peak rate=26,042cells/s, mean ON time = 0.13s, mean OFF time = 1.17s. All link delays were initially set to 0.0001s.

5.3.1 Relationship Between Timestep and Simulation Speed

Figure 18 shows the effect on cell processing rate of increasing the timestep value.

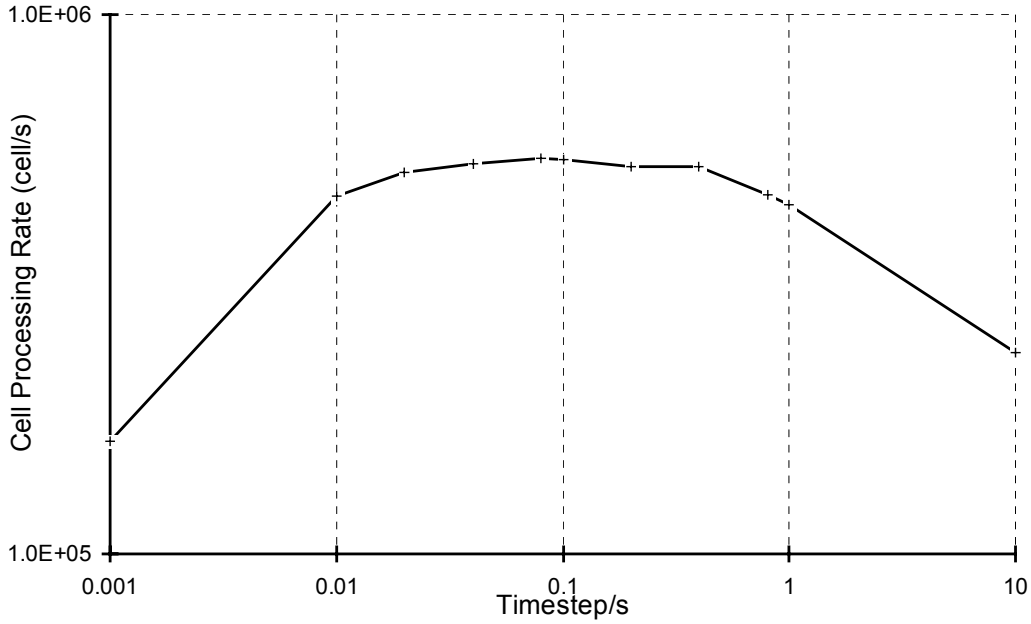

Figure 18 Effect on Cell Processing Rate of Increasing the Timestep Value

Between a timestep of 0.001s and 0.1s the expected speedup is observed; this is due to the reduction in the overhead of timesteps in which no useful work is done by the models (i.e. the overhead involved in calling the traffic generator and node queue sorting functions in the simulator is reduced). However, increasing the timestep value above the mean ON time causes the cell processing rate to drop: when events are placed in the stream queue of a given node model they are inserted in the correct time order but the sorting algorithm in this particular simulator starts from the head of the queue rather than the tail. This is slow if the queue becomes large.

When the timestep is set to a large value very large stream queues can develop before they are shifted into the sorted node queue and hence the process of sending an event becomes increasingly computationally intensive.

5.3.2 Relationship Between Timestep and Traffic Characteristics

Figure 19 and Figure 20 are cumulative relative frequency graphs showing the relative frequencies of ON bursts of different lengths measured at SW1 and SW2 for various values of timestep and a link delay of 0.0001s. Events arriving at SW1 will always be processed at the correct time because they are sourced directly from the traffic generator in the source model. The traffic at SW1 is therefore taken as the reference. However, events sent from SW1 to SW2 will have their processing times at SW2 altered by the timestep. Where the timestep is small relative to the mean burst length (0.01s compared with a mean ON time of 0.13s) there is little distortion of the traffic because, although event times will still be adjusted to be synchronous with timestep boundaries, the amount by which they are adjusted will be very small relative to the length of the burst. Furthermore, the small timestep size means that relatively few bursts will be completely enveloped by a single timestep and hence swallowed. As the timestep value is increased steps begin to appear in the cumulative relative frequency graph. These steps coincide with timestep boundaries and correspond to the events being synchronised with the timestep boundaries. As the timestep increases both the distance between steps and the height of the steps increases, showing that burst lengths are being concentrated at a few values corresponding to integer numbers of timesteps. It is also significant to note that as the value of the timestep increases relative to the mean ON time, the frequency of bursts of length 0 increases. This is because, for a fixed mean source ON time, there is an increasing chance of bursts being swallowed.

In summary, we can see that as the timestep is increased from small values relative to the mean burst length to significantly larger values two effects will cause distortion of the traffic: events will become synchronised with timestep boundaries resulting in burst stretching, and bursts will be compressed to zero length. These experimental results support the theoretical discussion in Section 5.2.

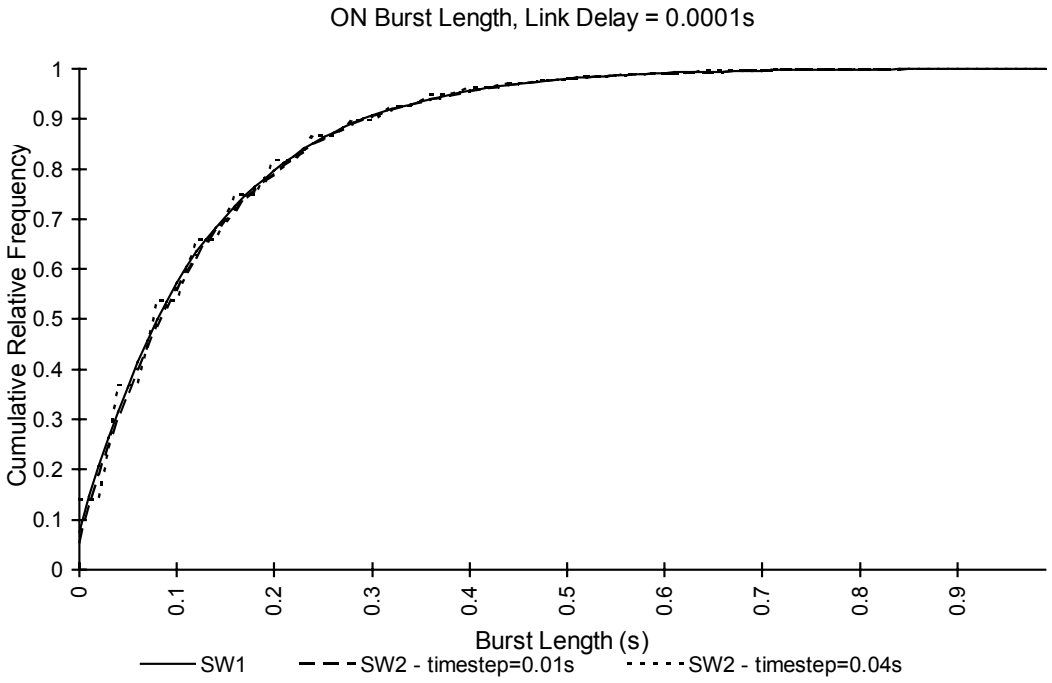

Figure 19 Effect of Timestep on Burst Length Distribution for Small Link Delay

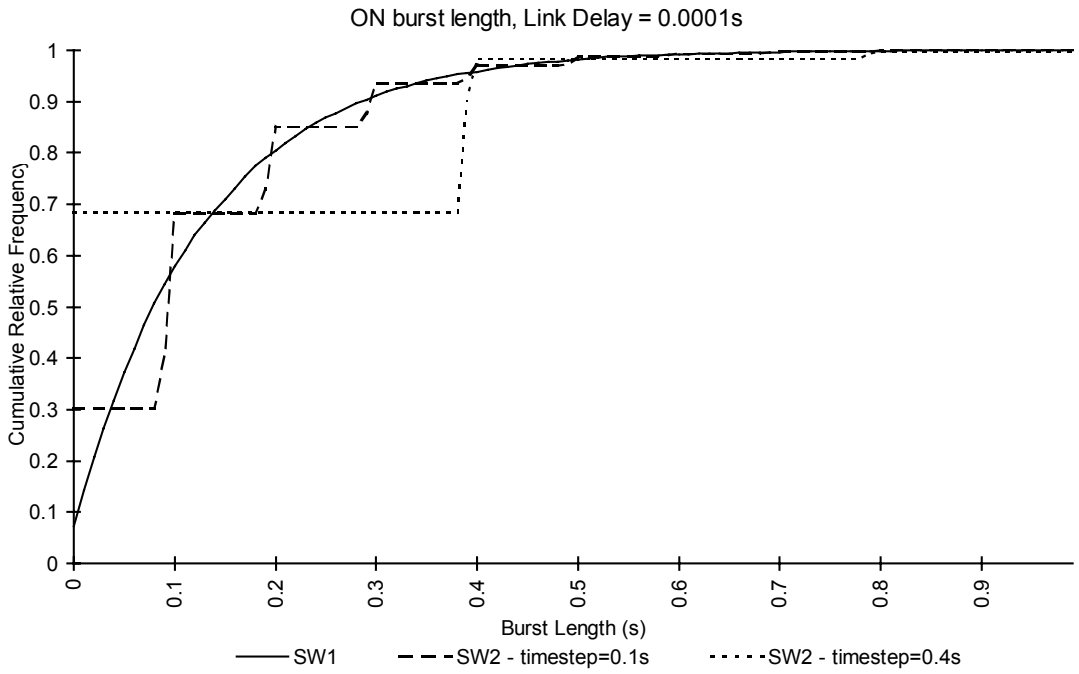

Figure 20 Effect of Timestep on Burst Length Distribution for Small Link Delay

5.3.3 Relationship Between Timestep and Link Delay

In order to assess the relationship between timestep and link delay, the delay between SW1 and SW2 was increased to 0.1s. The relative frequencies of the ON burst lengths arriving at SW1 and SW2 were measured and compared with the above results for a small link delay. The cumulative relative frequencies for the burst lengths for two values of timestep are shown in Figure 21. Consider first the comparison between measurements with a 0.1s delay and a 0.0001s delay and a timestep of 0.1s. The results show that setting the timestep to a value that is comparable to, or smaller than the link delay eliminates the traffic distortions described above. This is because all of the arrival times of events sent from SW1 are delayed past the start of the next timestep such that they will be processed at the correct time by SW2. However, if the timestep is increased above the link delay then the traffic distortions noted above start to appear. This is because the link delay is insufficiently

large to delay all events sent during the current timestep until the next timestep. However, events sent from SW1 within one link delay period of the end of the timestep will be delayed by the link delay until the next timestep and so will be processed at the correct time by SW2. This results in the less distinct steps in, for example, the 0.4s timestep graph with 0.1s delay than the 0.4s timestep graph with 0.0001s delay. Furthermore, this effect also reduces the relative frequency of zero length bursts, from 0.69 at 0.0001s delay to 0.5 at 0.1s delay in this case.

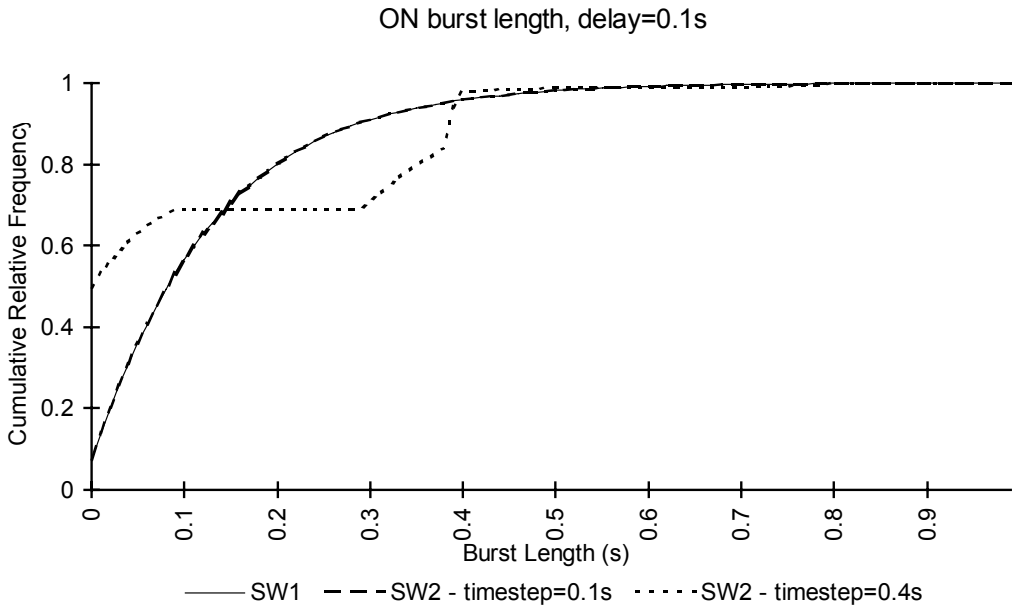

Figure 21 Effect of Timestep on Burst Length Distribution for Large Link Delay

5.3.4 Effect of Burst Length Quantisation on Cell Loss Measurements

Whilst the above simple switch models with no queueing are adequate for measuring burst lengths, they are not suitable for assessing any errors in the cell loss ratio that are introduced by timestepping. This is because the mean cell rate remains constant in spite of burst length quantisation. To demonstrate this a queueing model is required.

In order to demonstrate the effect that the burst length quantisation described above has on measurements of cell losses in a queue, an experiment [Bocci95.1] was set up using the cell rate simulator LINKSIM [Pitts93]. This models a single link queue through which the traffic from a number of sources can be multiplexed. In this experiment ON-OFF sources with an exponential characteristic were used. This characteristic was modified by quantising the burst lengths of each source into integer numbers of timesteps. The experimental topology is shown in Figure 22. The number of sources was varied from 1 to 50 in order to investigate the effects of combining a number of traffic streams while the link delay was assumed to be 0.0001s.

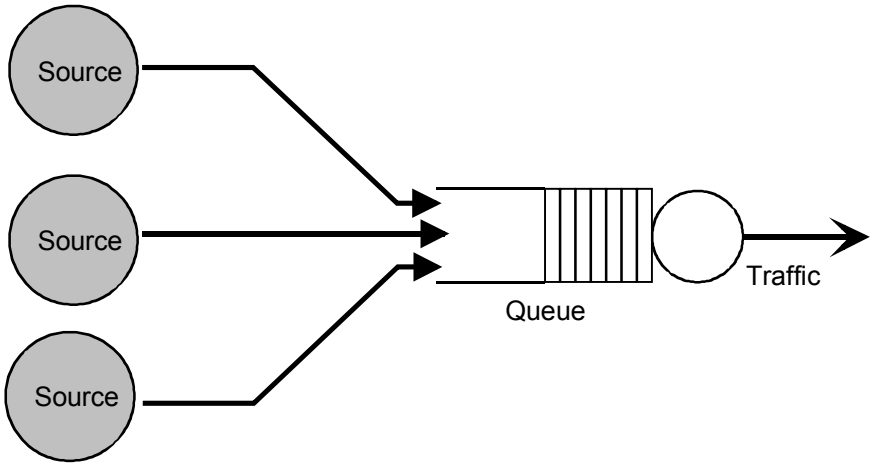

Figure 22 Cell Loss Experiment Using Linksim

Figure 23 shows the percentage change in cell loss as the effective timestep, or burst length quantisation, is varied from 0.0001s (the assumed link delay where there are no errors) to 0.1s. This shows that the errors in burst arrival times caused by timestepping are significant and that they increase with timestep size. The errors also increase as the number of sources multiplexed through the queue is increased.

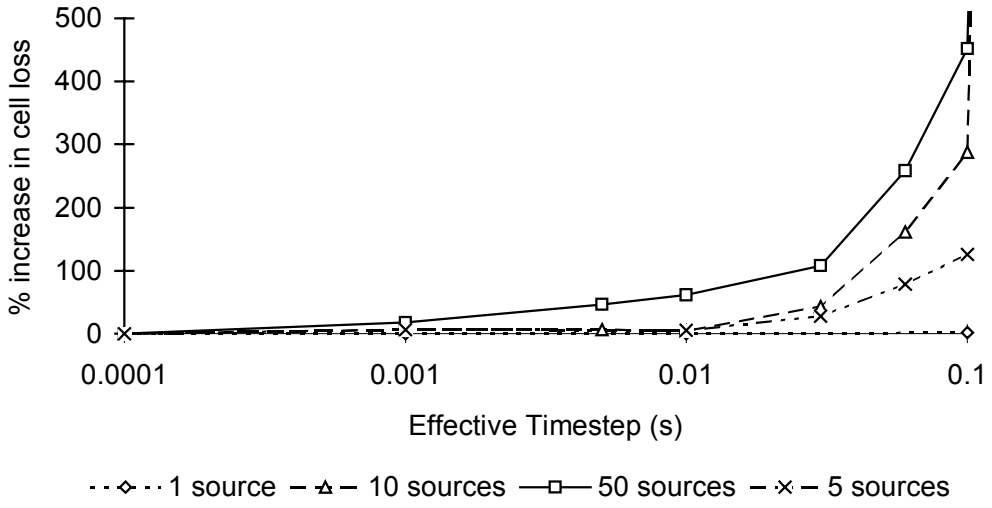

Figure 23 Variation in Cell Loss Measurements with Effective Timestep Size

5.3.5 Optimising Timestepping - a Two-Level Timestep Switching Scheme

The above experimental results and discussion suggest that there is a trade off between accuracy of the simulation results and the speed of the simulation. Clearly it is advantageous to set the timestep as large as possible so as to optimise the simulation speed. However, setting the timestep above the link delay results in significant errors in the way that events are propagated from one node to the next. These errors can be eliminated if the timestep is set to less than the link delay so that all events will arrive at the next node at least one timestep after being sent. However, since typical physical link delays are of the order of 5-500μs, setting the timestep value below the link delay will result in very long simulation times. Furthermore, since the timestep will typically be much less than the mean burst length, there will be many timesteps in which no useful work is done by many of the models in the simulator.

In order to try to maximise the speed of the simulator while maintaining a sufficient level of accuracy, the ICM simulator kernel provides the facility to switch between two levels of timestep. The default timestep size is the

larger of the two values, while the simulation is switched to the smaller of the two timesteps on request from a model. It then remains in small timestep mode for a fixed number of small steps; the value of this number is determined by an initialisation parameter (*n_small_steps*) before automatically dropping back to large timestep mode. In the ICM simulator, the models request a timestep switch by simply returning a flag to the kernel. This minimises the communication overhead between the models and the centralised time control element of the kernel (an important consideration when running the simulator on a multiprocessor computing platform). From the experimental results described previously it is clear that the timestep should be set to less than the link delay while events are propagating through the network. However, it can be set to a significantly larger value when there are no events propagating through the network in order to minimise the number of timesteps in which no useful work is done and hence maximise the speed of the simulator.

Consider the generation of events at source models. The source model should request small timestep mode on the large timestep boundary immediately preceding the generation of an event. The *n_small_steps* parameter should therefore be set according to:

$$n\_small\_steps = \frac{large\_timestep + next\_node\_processing\_time}{small\_timestep}$$

such that the simulator will always be in small timestep mode when the event is generated and will remain in small timestep mode for at least as long as it takes to deliver the event to the next node, and be processed there. This scheme is illustrated in Figure 24.

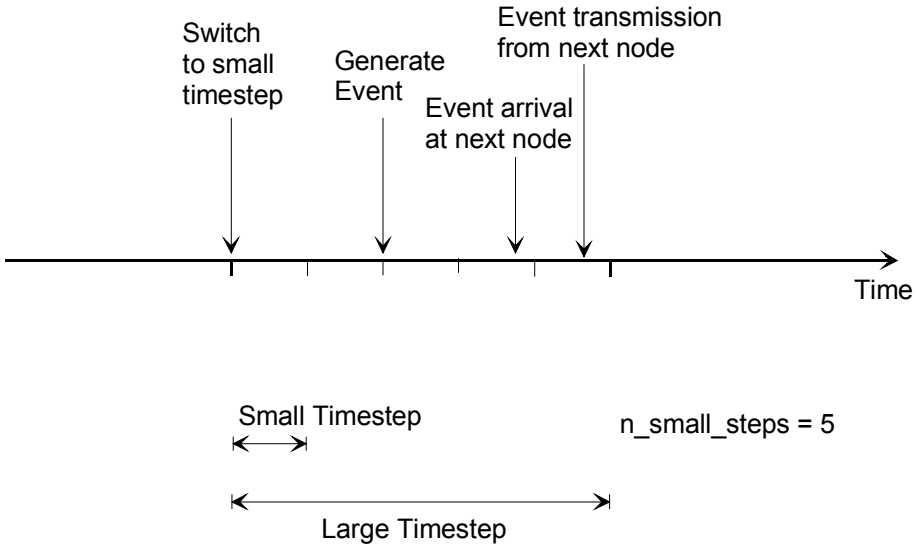

Figure 24 Timestep Switching Scheme

Now consider the situation at the next node, which will typically be a switch model. This node should also request small timestep mode when it has finished processing the event and sends it on an output link. This process continues as the event propagates through the network so that the simulator will remain in small timestep mode. When there are no more events to process the simulator will automatically drop back to large timestep mode.

With this scheme, the communication between the models and the kernel is minimised because only a single flag is transferred. However, the simulator will always remain in small timestep mode for at least *n_small_steps* after each request for small timestep mode by node models. Therefore the simulator will not drop back to large timestep mode for *n_small_steps* after the last event was processed. These small timesteps represent wasted processor time.

5.3.6 Performance With Timestep Switching

In order to assess the speedup potential of timestep switching, the above scheme was implemented in the simulator described in Section 5.1 using

the network topology of Figure 25 [Bocci95.1]. The small timestep value was set to the link delay of 0.0001s while the large timestep was varied between this value and 0.1s. The cell processing rate at switch 1 was measured with the traffic from 1, 5, 10 and 50 sources being fed through the switches. The traffic characteristics of each source were as before.

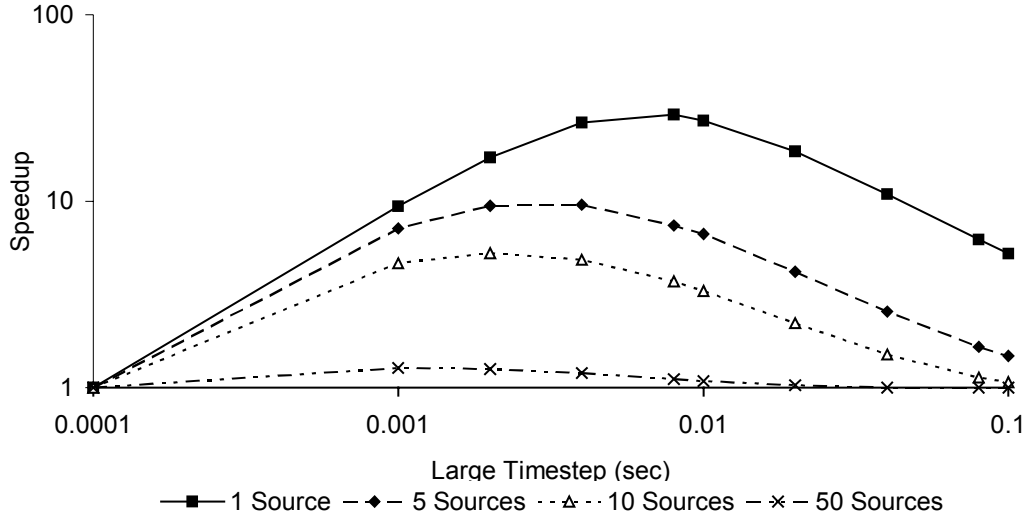

Figure 25 Speedup with Timestep Switching

Figure 25 shows the speedup attained by increasing the large timestep above the small timestep. For the 1, 5 and 10 source cases there is a clear maximum in the curves. This shows that some speedup is obtained by using timestep switching, while maintaining the accurate event propagation of the no switching simulator with the timestep set to the link delay. Above this maximum value progressively less speedup is achievable because of the excess of small timesteps in which no events are processed at the end of each event's propagation across the network. Furthermore, as the number of sources is increased less speedup is also observed. This is because the probability of small timestep mode requests is increased and so the simulator spends an increasing proportion of it's time in small timestep mode. Indeed, the results from the 50 source case suggest that

for the large numbers of sources that would be expected in a realistically large network model, the speedup from timestep switching is insignificant.

5.4 Summary

Timestepping is a synchronous system for time synchronisation within a parallel network simulator. Accurate cell loss results can be obtained with timestepping by setting the timestep to less than the minimum link delay in the network. However, with the timestep set at this value there are many timesteps in which no useful work is done at many of the node models. In very small networks, simple timestep switching schemes can reduce the number of these idle timesteps and hence increase the rate of the simulation. However, as the size of the network is increased by even a moderate scale, the timestep switching scheme becomes progressively less effective. Therefore, it must be concluded that simple timestepping is not an efficient method for synchronisation within a parallel cell rate simulator when modelling small networks. This is because the sparse nature of events in a cell rate simulation means that it is impossible to optimise the lookahead ratio and hence maximise the exploitation of parallelism in the network model.

The key aspect of the results presented in this section is that they apply to small networks. In small networks the probability of the occurrence of an event in any given timestep is very low and hence the majority of processor time is spent in idle timesteps. However, in practice a small network could be simulated within a relatively short time period on a traditional sequential cell rate simulator. There would be little justification for attempting to build a complex and potentially expensive parallel simulator for dealing with such trivial problems. In practice the networks in which the performance of traditional sequential cell rate simulators is inadequate incorporate tens of thousands of users and many tens of nodes. In cell rate simulations of such commercial scale networks,

there will be a much greater probability of the occurrence of an event during any given timestep. The proportion of processor time wasted in idle timesteps will therefore be greatly reduced when compared with small network simulations. In order to fully assess the potential of timestepping as a time synchronisation scheme for parallel cell rate simulation of ATM networks, its performance must be compared with that of traditional sequential schemes in realistically large networks. Chapter 6 of this thesis describes the design and application of a cell rate ATM simulation program (SPROG) capable of allowing a fair comparison of timestepping with other schemes when simulating large networks. However, the argument presented here does not degrade the contribution that the research described above represents. The study of timestepping using the ICM simulator has demonstrated the effect of the scheme on the accuracy of traffic measurements that may be obtained using the simulator, and has provided an insight into issues relating to the parallelisation of the cell rate model. In particular, it has demonstrated the importance of applying simulation techniques that are appropriate to the problem under study.

6. SPROG Object Oriented Simulator

6.1 Motivation for the Development of SPROG

In Chapter 5 of this thesis the simulator developed by the RACE II project ICM was used as the basis for studying the performance of the timestepping mechanism. However, whilst this simulator is a highly capable modelling tool, it was developed with the aim of providing a test environment for a TMN system and not for the evaluation of different simulation techniques. The ICM simulator has proven to be a useful tool for demonstrating the effects of timestepping on the simulated traffic in a limited range of experiments using small network scenarios, but it is not suitable for comparing timestepping with other synchronisation schemes over a much wider range of network scenarios. In particular, it is desirable to be able to assess the performance of timestepping in comparison with other synchronisation schemes and in very large (and hence more realistic) networks.

The inflexibility in the ICM simulator is a result of the fact that it was designed from the outset as a timestepping simulator. It would be a major task to re-engineer the simulator around a different synchronisation scheme. Furthermore, in order for it to be possible to make a fair comparison of the performance of a timestepping version of a simulator with one using some other scheme, the amount of executed code that is common to the two versions should be maximised. Indeed, ideally the only differences between the simulators should be the code directly relating to event list management and time synchronisation. This would be difficult to guarantee if an existing timestepping simulator were simply modified to incorporate some other event list management scheme.

The requirement for a highly flexible simulator for studying the performance of a variety of time synchronisation schemes in a wide range of ATM network scenarios was the main motivation for the development, by the author, of the SPROG simulator (Simulation PROGram). The main requirements on the design of the simulator were as follows:

- The simulator must be scaleable from small to very large networks.
- It must be possible to modify the simulator for different time synchronisation schemes with a minimum of effort.
- The amount of common code shared between simulators with different time synchronisation schemes should be maximised.

In order to achieve a high level of flexibility and code re-use the simulator was based on an object oriented design and implemented in the C++ programming language.

This chapter first outlines the principles of object oriented software design before describing the design of the SPROG simulator, both in terms of its functional architecture and object class definitions. The application of the simulator to a study comparing the performance of timestepping in the simulator with that of a traditional linear centralised event list management scheme is then described.

6.2 Object Oriented Software Design

In this section, a brief overview of object oriented software design and its application to simulator development is given. A detailed description of object oriented programming is beyond the scope of this thesis and the reader is referred to the many tutorial books on the subject, for example [Wiener88].

6.2.1 Overview

Object oriented programming is a relatively new method for designing and implementing software systems. It differs from traditional functional programming[6] in that it provides a much closer link between the design and the implementation phases of software development. In comparison with procedure-based programming, object oriented techniques aim to improve programmer productivity by increasing software extensibility and reusability and to control the complexity and cost of software maintenance [Wiener88]. This means that existing code can be reused, with additional features added if required, in new software with a minimum of effort expended in 'porting' the old code to the new software system.

The central concept in object oriented programming is the *abstract data type (ADT).* This is a model that encompasses a *type* and a *set of operations* that characterise the behaviour of that type. In C++, the behaviour of an abstract data type is described by a *class definition.* The class definition describes the data structure of the type and the interface to all of the operations that can be performed on the type. The scope of the data and operations of an ADT are also described by the class definition. These can be either *private,* in which case they are only visible within the scope of the class, or *public* in which case they are visible outside the class. A third category of scope that appears in C++ is known as *protected.*

The architecture of an object oriented system is built around a set of classes that describe the behaviour of all of the underlying data in the system. Objects are individual instances of a given class and model the data and operations of a specific type. Objects interact by sending messages to one another that cause operations to be performed on the data contained within an object. Hence it can be seen that object oriented software design firstly considers the data of a system and secondly the manipulation of that data. This is in contrast with traditional procedure-

[6] as implemented using languages such as Pascal and Fortran

based program design that tends to consider the operations first and the data second.

6.2.2 Encapsulation

Wiener [Wiener88] defines Encapsulation as the *process by which individual objects are defined.* This entails:

- a clear definition of the scope of all the object's internal software
- a clearly defined interface that describes how objects interact with other objects
- a protected internal implementation that is invisible from outside the object.

Objects encapsulate all of the data and operations that characterise a particular type. In C++, functions, or *methods*, are used to manipulate the data, or *attributes*. These methods provide the clearly defined interface through which objects interact. The scope of both the methods and the objects attributes can be defined to be *private* (internal to the object), *protected* (visible only to objects of classes derived from this one), and *public* (visible to other objects of different classes). This scoping mechanism can be used to protect the internal implementation of an object from outside view because all interaction with that object must occur through the clearly defined interfaces of the object's methods. Therefore, provided the interfaces remain the same, the details of the internal implementation of the object can be changed with little or no redesign of the surrounding software system. This feature of object oriented programming provides a major enhancement over procedural programming in terms of the ease of software maintenance.

6.2.3 Inheritance and Polymorphism

Software extensibility and re-usability are supported by the concepts of inheritance and polymorphism. In object oriented programming, a hierarchy of classes exists in which some classes, known as *sub-classes* or *derived* classes, are subordinate to a parent class. These classes are derived from the class above them in the hierarchy. The classes at the top of the hierarchy are known as *base* classes.

When a sub-class is derived from some other class it will encapsulate not only its own set of methods and attributes but also those of the parent class. This is the principle of *inheritance*. One important result of this is that it is possible to easily extend the functionality of a class by simply deriving a sub-class from it that provides the additional methods and attributes without having to write a completely new class. This is illustrated in the example inheritance tree of Figure 26.

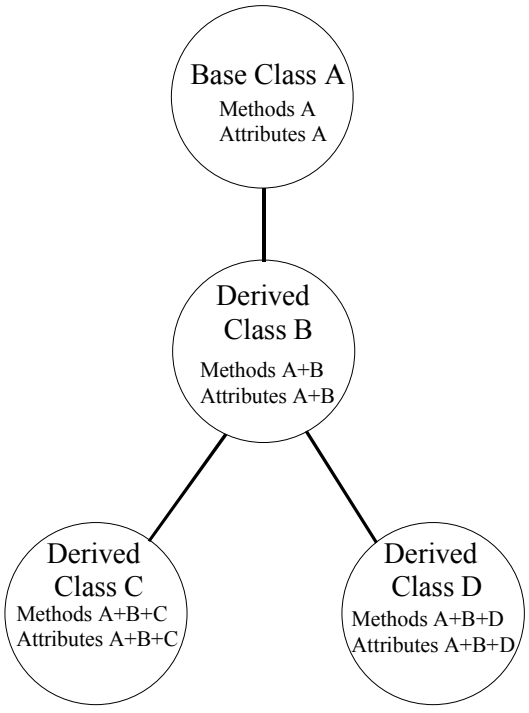

Figure 26 Example Inheritance Tree

Inheritance means that if, for example, class B is a sub-class of class A then there will be some portion of the interface to class B that is common with that of class A. Therefore, in principle an object of class B can be used wherever an object of its parent class A can be used because they will both be able to respond to some common message format. This principle is known as *polymorphism*. Polymorphism is a key feature of object orientation because it also allows methods in a base class to be redefined (*overloaded*) in sub-classes with the same interface but different functionality. In C++, methods in a base class that are to be redefined in sub-classes are known as *virtual functions*.

6.2.4 Application of Object-Oriented Techniques to Network Simulation

The basic objective of discrete event simulation of telecommunications networks is to model the operations and interactions of the various hardware and software elements within the network. It is convenient to model elements as 'black boxes' with a standard interface comprising a restricted set of inputs and outputs together with a model of their internal behaviour.

Object oriented techniques provide a very natural mapping of this conceptual network model into software. Complex networks can be modelled by initially defining a generic set of base classes at the most abstract level that represent the principle features of the system. Sub-classes that model increased levels of detail in specific elements, such as traffic sources, links or switching elements are then derived from these base classes. Polymorphism is an essential facilitator for this process as it enables both the interchange of models within the simulator and the easy addition of new functionality to existing models. Encapsulation means that the problems of modelling particular network elements will remain local to the simulation model, and that strict control can be maintained over which internal details of a given model can be made available outside

its scope. This modularity aids in software maintenance as well as reliability.

A further motivation for using object oriented techniques in simulator design is that the international bodies of the telecommunications industry are increasingly specifying interfaces and protocols using formal top-down specification techniques. An example of this is the Specification and Description Language, SDL [Z.100][Belina89] that is in widespread use in ITU-T recommendations. Object oriented languages enable such descriptions to be readily mapped into software simulations.

Little & McCue describe an object oriented simulation package implemented using the C++ language [Lit94]. They endeavoured to emulate the object oriented features of the SIMULA simulation language and found that C++ provided a number of advantages over the use of a SIMULA based development tool. In particular:

- C++ compilers typically generate code that is several times more efficient than similar SIMULA code resulting in faster simulations.
- C++ provides more extensive object oriented features than SIMULA. For example, C++ provides the keywords *public, protected and private* in order to control the scope of class methods and attributes, but in SIMULA everything is public.

In summary, object oriented programming languages provide a natural mapping of 'real world' systems into software in a manner that enables highly flexible and extensible simulators to be rapidly designed and coded.

6.3 Architecture of SPROG

The logical architecture of SPROG is shown in Figure 27. The simulator architecture consists of 2 major blocks: The *simulation server*, and the

models. The simulation server provides a set of *simulation services* that provide the basis for building a discrete event simulator. These services are largely independent of the precise network technology being modelled. The models are responsible for the detailed simulation of the elements of the telecommunications network (known in SPROG as *places*).

The simulation services provide a wide range of generic simulation support functionality and include the following:

- Pass messages between the models.
- Ensure the synchronised time progression of the models.
- Event list management.
- Configuration of the simulator.

The server supports a standard interface though which all models communicate with it. This interface presents an object oriented view of the server to the models.

As well as providing a set of simulation services, the server includes a number of prototypes for the various entities in the simulation. These include base class definitions for *places* (representing network nodes such as traffic sources and switches), *events* (representing, for example, cell rate changes or signalling messages), and inter-place *links*. These are described in more detail in the next section.

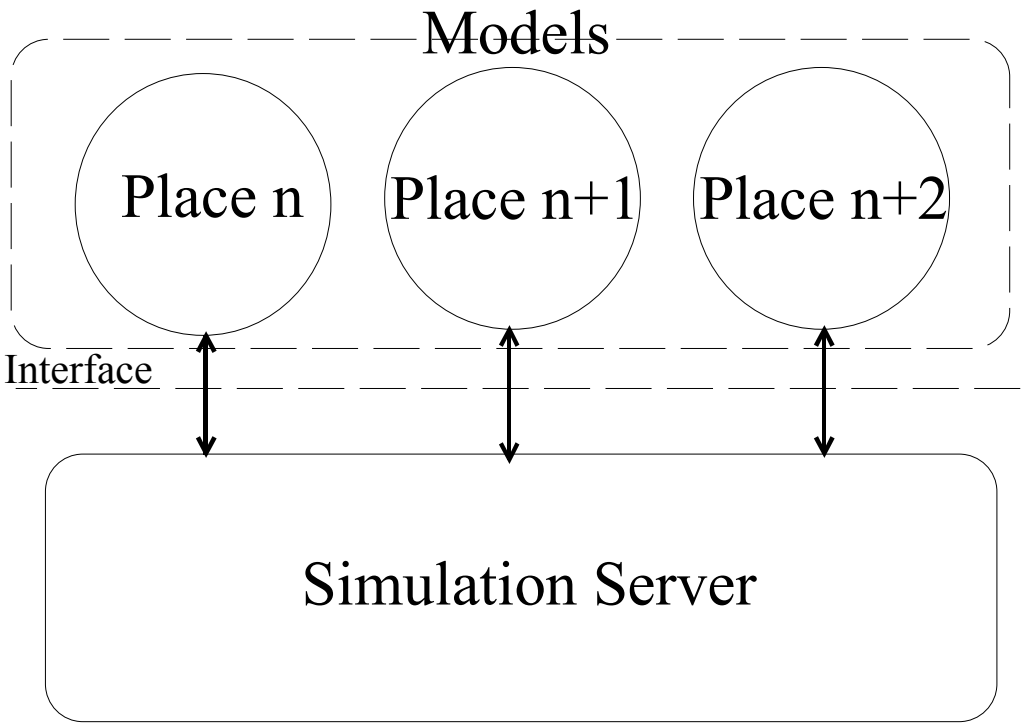

Figure 27 Logical Architecture of the SPROG Simulator

6.3.1 Object Class Definitions

Figure 28 shows the inheritance tree for the SPROG simulator. Two main groups of classes are defined: a set of base classes that constitute the *simulation services library*, and a set of *user defined sub-classes*. In general, the user defined sub-classes are derived from the base classes in the simulation services library in order to implement the full detail of the simulation. However, users can also define their own base classes where necessary.

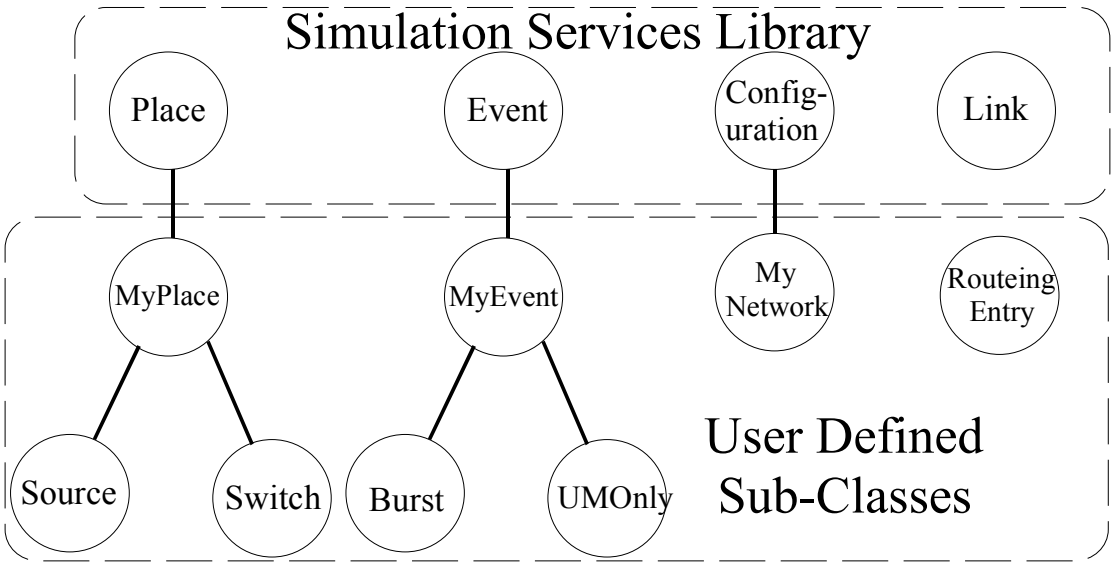

Figure 28 Object Class Definitions for SPROG

6.3.1.1 Simulation Services Library

The simulation services library implements the majority of the functionality of the simulation server. This library contains a set of base classes that incorporate the essential data and functionality required by any discrete event driven network simulator. The properties of these classes can be inherited by object classes derived from them in order to implement almost any entity in the network together with any further functionality required for the operation of the simulator.

The classes of the SPROG simulation services library are summarised as follows:

6.3.1.1.1 Class Place and Class Link

`Class Place` and `Class Link` are the base classes that enable the basic physical topology of the network to be modelled. `Class Place` contains the functionality and data for modelling a generic network node that has associated with it one or more output links. These output links are modelled using objects of `Class Link`. The arrangement of objects of `Class Place` and `Class Link` in order to form a network is shown in

Figure 29, while the principal public and protected member functions and attributes are shown in Table 3. These form the interface to the Place and Link simulation services.

`Class Place` contains a number of functions for controlling simulation time and the configuration of any individual place, including manipulation of the initialisation file for this place (each place is assumed to be initialised through its own individual file). Additionally, `Class Place` contains a virtual function for processing events arriving at this node. This can be overloaded by classes derived from `Class Place` in order to model the behaviour of any network node while maintaining a common interface to all network nodes. Attributes of `Class Place` include local and global simulation time, an array of pointers to all places in the network, the global ID of this place, pointers to the objects modelling the output links from this place, and the name of this place's initialisation file.

The attributes of `Class Link` include the destination place of the link (links are assumed to be unidirectional), the port number at which the link connects to that place, and the link delay. The methods are simply associated with getting and setting these attributes.

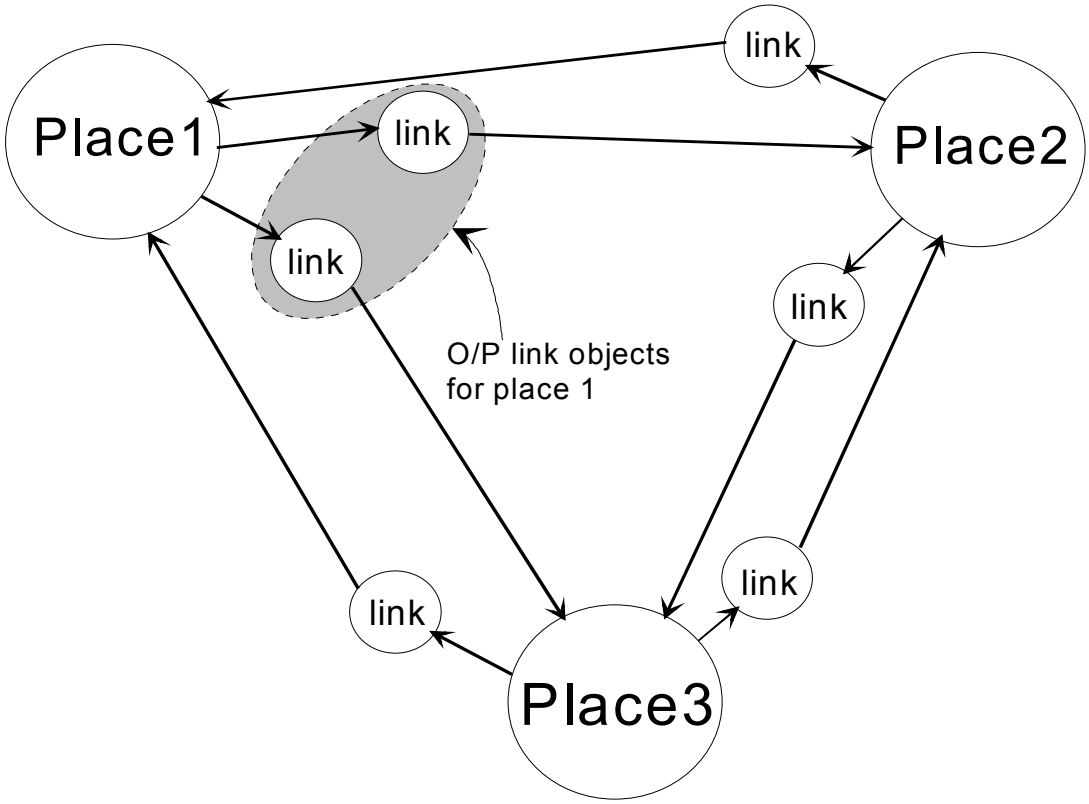

Figure 29 Arrangement of Places and Links in SPROG

```
//class place is base class for all network nodes
class Place
{
public:

     //static functions for creating a network of places
     static void SetGlobalTime( double time); //set the global simulation time
     static double GetGlobalTime();           //returns the global simulation time
     static void CreatePlaceList(int NoPlaces);              //create list of places
     static void SetPlacePointer(Place *ptr, int p)  //set placelist pointer
     static Place *GetPlacePointer(int p)            //return pointer to place p
     static Place **GetPlaceList()            //return pointer to placelist
     int GetThisPlaceID()                     //return global ID of this place
     inline void SetThisPlaceID(int id)       //set global ID of this place

     //functions for initialising this place
     inline void SetNOutlinks(int n);         //set number of output links
     inline int GetNOutlinks();               //get number of output links
     inline Link *GetLinkPointer(int no);     //return pointer to outlink n
     inline void SetIniFileName( char *name );//set ini filename for this place
     inline char *GetIniFileName();           //returns init filename for this place
     virtual void ProcessIniFile();           //process init file for this place

     //run time place simulation functions
     virtual void ProcessEvent( class Event *ptr, int inc_port, double t) //process
                         next event for this place

     //general functions
     virtual void PrintPlaceConfig(); //print configuration of this place

protected:
     void SetUpLinks(int n);          //create output link objects
     void SetLocalTime(double t)      //set the local time
     inline double GetLocalTime()     //get the local time
     static double GlobalTime;        //record of global time across whole simulation
};

//class link represents the output links from a place
class Link
{
public:
     void SetLinkWhereTo(int to);     //set destination place for this link
     int GetLinkWhereTo();            //returns destination place for this link
     void SetLinkDelay(double d);     //set link delay
     double GetLinkDelay();           //returns the link delay
     void SetLinkDestPort(int n);     //set incoming port number on destination place
     int GetLinkDestPort();           //returns port number on destination place
};
```

Table 3 Interface to Place and Link Simulation Services

6.3.1.1.2 Class Event

This is the base class for all events in the simulation. the principle methods are for sending events between places, for event list management (a simple linear event list scheme is implemented as a virtual function so that it can be changed by overloading to some other scheme in a derived class), and for creating and managing lists of free events; objects representing events that are no longer required by the simulator are placed on a free list for later reuse rather than being deleted. This reduces the amount of dynamic memory allocation thus reducing the number of

'holes' in the memory allocated to the simulator. Attributes include event time, incoming port at the place to which event is to be delivered, destination place, a pointer to the list of free event lists and a pointer to the event list head. Table 4 shows the principle functions of the interface to the event simulation service.

```
//class event is base class for all event types in simulation.
class Event
{
public:
     //free list management functions
     static void CreateFreeEventLists(int No);//create array of pointers to free events
                                              //also initialises head pointer for
                                              // central event list
     void FreeEvent(int type);                //place used event on free event list
     static Event *GetEvent(int type);        //get event from free list

     //event list management functions
     virtual void SendEvent(int output_link, class Place *ptr,
               double time, int priority);   //send event on outlink from this place
     static Event *GetNextScheduledEvent();   //get the next event from event list
     static double GetNextScheduledEventTime();      //get time of next event
     inline static Event *GetEventListHead();        //get pointer to event lists
     static void Event::DumpEventList(Event *ptr);   //dump event list to screen

     //event attribute manipulation functions
     double GetEventTime();           //returns scheduled arrival time of this event
     int GetSourceID();               //returns ID of place that sourced this event
     int GetDestinationPlace();       //returns ID of destination place for this event
     int GetIncommingPort();          //returns incoming port on destination place
     Event *GetpNextEvent();   //get pointer to next event in list
     void SetpNextEvent(Event *ptr);  //set pointer to next event in list

     //virtual prototypes
     void DisplayEvent(){ }           //provides a standard interface for all types
                                      //of event

protected:
     void SetPriority(int p);                //set the event priority
     void SetSourceID(int ID);               //set source ID
     void SetDestinationPlace(int dest);     //set destination place
     void SetEventTime(double t);            //set the event time
     void SetIncommingPort(int inc_port);    //set incomming port on destination place

};
```

Table 4 Interface to Event Simulation Services

6.3.1.1.3 Class Configuration

`Class Configuration` is the base class that describes the current configuration of the network. It contains functions for processing the global network configuration files (the *run time control* and *network configuration* files), together with some basic file handling and file input/output functions. Attributes include the names of the run time control and network configuration files, the global end of simulation time,

and the total number of places in the network. Note that in general, for any given simulation, only one object instance of `Class Configuration` will exist. Table 5 illustrates the principle functions of the interface to `Class Configuration`.

```
class Configuration
{
public:

     virtual void ProcessRunTimeControl();    //process the run time configuration file
     virtual void ProcessNetworkConfig();     //process the network configuration file
     double Get_EndSimTime;                   //return the end of simulation time
     int Get_NumPlaces();                     //return number of places in the network
     char *Get_Word(ifstream *fp)             //get text string from input filestream
     int Get_Int(ifstream *fp);               //get an integer from input filestream
     double Get_Double(ifstream *fp);         //get a double from the input filestream
     void Write_Log(const char *, const char *);     //write a string to output logfile
     inline char *GetLogFileName();           //return name of log file

protected:
     void SetRunTimeControlName( char *rcf);  //set run time configuration filename
     void SetNetworkConfigName( char *netcon);//set network configuration filename
     char *Get_RCFName();                     //get runtime control file name
     char *Get_CNFName();                     //get configuration file name
     void Set_EndSimTime(double time);        //set end of simulation time
     void Set_NumPlaces(int n);               //set number of places in the network
     void SetLogFileName( const char *lfn);   //set filename of output log file

};
```

Table 5 Interface to Class Configuration

6.3.1.2 User Defined Sub-Classes of SPROG

Based on the library of base classes, the simulator developer can derive a set of object classes that are more specific to their simulation needs. In the case of the study of timestepping performance, objects were required for modelling cell rate ATM traffic sources and switches, as well as objects to represent cell rate change events. A class was also required to represent locally scheduled events at the traffic source (for example, the time of the next cell rate change on a VCC), together with a class to represent extra details of the configuration of the ATM network over and above that which could be described by the simulation services library.

An important consideration in the development of the SPROG simulation services library was that it should be possible to use it as the basis for simulators of a range of network technologies using a wide variety of

simulation techniques. Therefore the library was implemented with only those features that were considered to be sufficiently generic. However, it was designed with maximum flexibility in mind. During the development of the derived classes for the timestepping study it was found that a number of attributes and methods were required that, whilst being common to all of the objects derived from a given base class, were not sufficiently generic to be included in the simulation services library. An example of this is the necessity in a timestepping simulator to have a local timer queue at every place, whether that place be of type traffic source or switch. Therefore, where necessary, common classes were derived from the appropriate base classes in the simulation services library. This was found to be necessary for object classes of type Place and type Event, giving rise to `Class MyPlaces` and `Class MyEvents`.

The classes derived for the purposes of building a simulator for the timestepping study are summarised as follows:

6.3.1.2.1 Class MyPlace

In the timestepping version of the simulator this class encapsulates the timer, stream, and sorted node queues for a place. The attributes are the pointers to these queues (which are implemented as linked lists of event objects), while the methods are associated with manipulating these pointers. The programming techniques used to enable SPROG to be used for the timestepping study are described below.

6.3.1.2.2 Class Source

Objects of `Class Source` model groups of ATM traffic sources within the network. Each source contains an ON-OFF cell rate traffic source for an unrestricted number of permanent VCCs. The timing of cell rate change events on the VCCs is determined by a Poisson arrival process.

6.3.1.2.3 Class Switch

Objects of `Class Switch` implement a cell rate model of a simple zero size buffer burst level queue/server. In this model, cell scale queueing is

not modelled. Outgoing bursts are routed on to the appropriate output link of the switch based on their VCI using a static routeing table that is set up at initialisation time. Switches are able to calculate cell loss and cell throughput statistics.

6.3.1.2.4 Class RouteingEntry

A data object representing a single entry in a switch routeing table.

6.3.1.2.5 Class MyEvent

In the timestepping version of the simulator, this class encapsulates the methods and data for managing the timestepping event lists. In both timestepping and centralised event list versions of the simulator, this class has attributes describing the type of the event (i.e. Burst or UMOnly).

6.3.1.2.6 Class Burst

This class is derived from `Class MyEvent` and represents a cell rate change. Attributes include the size of the rate change (in cell/s), VCI, and a burst serial number (BurstID).

6.3.1.2.7 Class UMOnly

`Class UMOnly` is used by objects of `Class Source` to schedule cell rate changes on a given VCC.

6.3.1.2.8 Class MyNework

Contains methods and data for the configuration of a particular network.

6.3.2 Event List Management

The SPROG simulation library provides a basic event list and time synchronisation scheme for sequential simulation. This is based around a linear event list. In this scheme, a single time ordered list of all of the currently scheduled events in the whole simulation is maintained. New events are inserted in the correct position in the list using a simple

searching algorithm that starts at the head of the list and moves down until the first event of greater scheduled time is found. The events are stored in the list in increasing time order, and therefore the next scheduled event is always at the head of the list. Note that, while the insertion order of simultaneous events is preserved, this event list management scheme is unable to deliver multiple simultaneous events that occur at a model together. Therefore, although this scheme still enables correct cell rate modelling, spurious events are generated during periods of cell queueing. This can have a significant impact on the performance of the event list. An event list scheme that properly accounts for multiple simultaneous cell rate changes is described in Chapter 7.

The event list is implemented in SPROG using a linked list of event objects as shown in Figure 30.

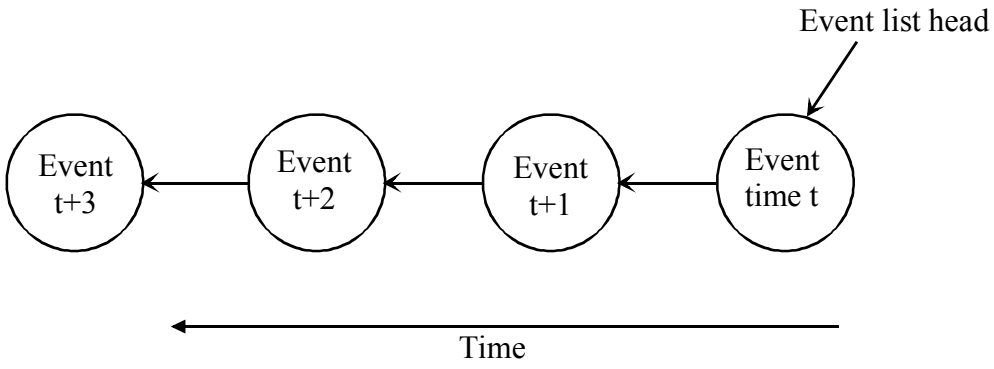

Figure 30 Simple Linear Event List

The object oriented design of the simulator enables this form of event list management to be easily interchanged with alternative schemes by simply overloading the appropriate functions and providing and additional attributes in sub-classes of `Class Place` and `Class Event`. For timestepping, stream, sorted node and timer queues were added to `Class Place` through the derivation of `Class MyPlace`. The additional functionality required to manage this scheme was achieved by overloading the event management methods in `Class Event`. The code for each

particular event list management scheme was enclosed within conditionally compiled blocks enabling a different scheme to be selected at compile time.

6.4 Application of SPROG to the Comparison of Synchronisation Schemes

In order to assess the potential of timestepping as a practical time synchronisation scheme for ATM cell rate simulators that are portable between a sequential and a parallel environment, the speed performance of the timestepping version of SPROG was compared with a version that was compiled with the simple linear event list of the simulation services library.

Despite the fact the timestepping is intended as a synchronisation scheme for parallel simulation, it is considered reasonable to assess certain aspects of its speed performance on a sequential computer. In such conservative synchronisation schemes running on parallel platforms, the speed of the whole simulator will always be limited by the speed of the slowest processor. This is because all of the processors must have finished processing all of the events for the current timestep before any of the processors can begin to process events scheduled for the following timestep. Therefore, in order to suggest that a timestepping parallel cell rate simulator could give some speedup over a conventional cell rate simulator, it must be demonstrated that the slowest processor in a timestepping parallel simulator will never simulate a portion of the total network (the size of this portion will depend on the level of granularity of the spatial decomposition) at a slower rate than a sequential simulator using a conventional synchronisation scheme would simulate the complete network. It follows that, to achieve the ideal situation of a linear speedup with increasing numbers of processors for a given total fixed network size, each processor must simulate at N times the rate of a sequential

simulation of the complete network (where N is the total number of processors). Note that the assumption is made that delays due to the transfer of both synchronisation information and cell rate change events between processors are small compared with the processing time of those events at each processor. This is because cell rate modelling requires greater amounts of floating point arithmetic than cell level modelling.

The assessment of parallel simulation techniques using sequential simulators has the major advantage that potential parallelism can be identified without the need to for any implementation on a real multiprocessor system, hence avoiding the need for complex and time consuming development work. Indeed, authors in other fields of simulation have used sequential simulators to assess the potential of a parallel simulation scheme, for example Rawling et al [Raw92].

6.4.1 Experimental Comparison of Timestepping and Linear Event List

The experimental network configurations are shown in Figure 31 and Figure 32. Four experiments were carried out on both a version of SPROG that used a timestepping synchronisation scheme and also a version of SPROG with a simple linear event list. These experiments had the following objectives:

- To assess the performance of timestepping in lightly loaded networks
- To assess the performance of timestepping in heavily loaded networks
- To assess the effect of network scaling on the performance of a timestepping simulator

Two network topologies were chosen:

- Three node network. This consisted of one traffic source model, one simple switch, and one further switch configured as a traffic sink.
- Eleven node network. This consisted of one traffic source model, nine simple switches and a switch configured as a traffic sink

These networks are shown in Figure 31 and Figure 32.

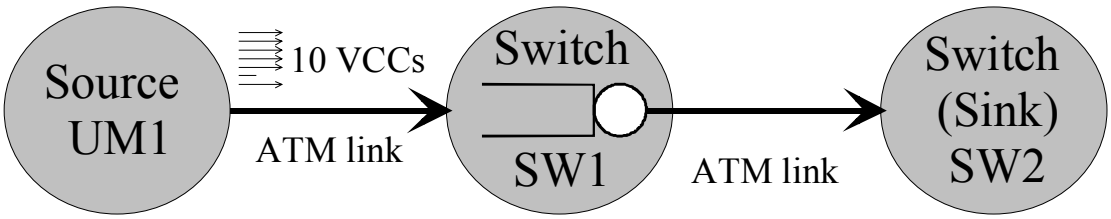

Figure 31 Three Node Experimental Network Topology

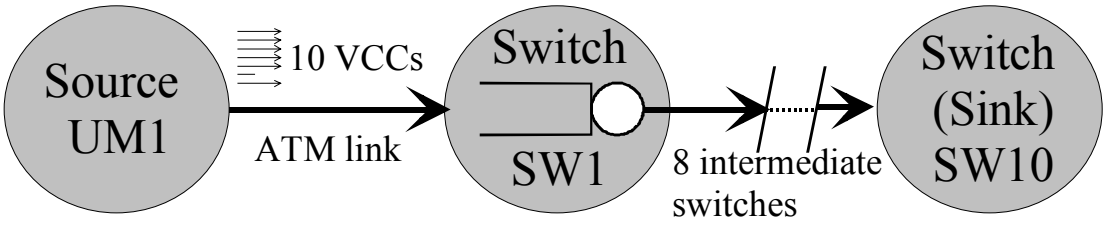

Figure 32 Eleven Node Experimental Network Topology

Note that while the network topologies are clearly not representative of those of real telecommunications networks, they were deliberately chosen to simplify the analysis of the burst propagation thought the network and the effects of scaling. All link delays were set to 0.001s. The timestep size was set to the link delay in the timestepping SPROG such that there would be no event propagation errors and corresponding burst length quantisation. This represents a 'worst case' scenario for the impact of timestepping on simulator speed. If some error in the results can be tolerated, then the timestep size can be increased, as demonstrated in

Chapter 5 of this thesis. Ten permanent virtual channel connections were set up between the source and the sink through the switches. The traffic sources were of ON-OFF type with a peak rate of 100 cells/s and a mean ON and mean OFF time that was varied between 10s and 0.01s in order to force a known mean number of events to occur per timestep. The ON and OFF times were exponentially distributed about the mean. The buffers in the switch models were of sufficient capacity to accommodate cell scale queueing, but if any burst scale queueing occured it immediately resulted in cell loss. Note that, whilst the queueing of cells in the buffers in this experiment is not explicitly modelled, the term queueing (in preference to loss) is used here to emphasise the fact that it is the change in the state of the buffer from no queueing to queueing (and vice versa) that affects the number of cell rate changes and hence the distribution of events on the output of the buffer. Therefore, queueing represents any situation where the total input rate of the buffer is greater than its server rate. Note also that no account was taken in either implementation of the simple linear event list scheme or the timestepping scheme of the occurrence of simultaneous events at the output of buffers during burst scale queueing. The impact of these on the performance of simulators is investigated in Chapter 7.

For each network topology, two groups of experiments were performed

- No burst scale queueing
- Burst scale queueing in SW1 or SW1 and SW2
    - Low utilisation resulting in a small amount of burst scale queueing in SW1
    - High utilisation resulting in heavy burst scale queueing in SW1 and SW2 in the eleven node network, and heavy burst scale queueing in SW1 of the three node network.

The computing platform on which these experiments were run was as follows:

- Hardware:

  IBM AT clone with 100MHz Intel 80486DX4 CPU and 16Mbyte RAM

- Operating System:

  Microsoft Windows 95/MS DOS 7. SPROG always run with PC in DOS mode

- C++ Compiler:

  Microsoft Visual C++ v1.51. SPROG compiled as a DOS application with full compiler optimisations selected.

6.4.2 Comparative Performance of Timestepping with no Queueing

In this experiment all of the switch server rates were set to 1000 cells/s. The peak cell rate of each VCC was 100 cells/s and hence no burst scale queueing occurred in the network. This experimental set-up was used for both the three node and eleven node networks.

Figure 33 and Figure 34 show the effect on the cell processing rate, measured at SW1, of decreasing the mean ON and mean OFF times in the three node and eleven node timestepping and linear event list simulators. In both cases, the cell processing rate decreases in an approximately linear manner in the linear event list simulator.

There are two major factors that determine the cell processing rate of the simulator:

- time taken to process each event in the models
- time taken to place each event in the correct time ordered position in the event list

When there are few events propagating through the simulation the first factor will dominate. However, increasing the numbers of events by either reducing the ON/OFF times of the traffic or increasing the numbers of nodes will lead to an increasingly large central event list so causing the second factor to dominate.

In the timestepping simulator, there is very little decrease in cell processing rate with a mean ON (and OFF) time of 10s and 0.1s. This is because when there few events to process in the simulator there is a significant overhead of timesteps in which no useful work is done by the models. However, as the mean ON time is reduced below 0.1s, the cell processing rate begins to drop . This corresponds to 0.1 total source events per timestep. This indicates that the majority of CPU time is spent processing events in the models rather than empty timesteps. It is clearly desirable to maximise the proportion of CPU time spent in useful event processing work in the models.

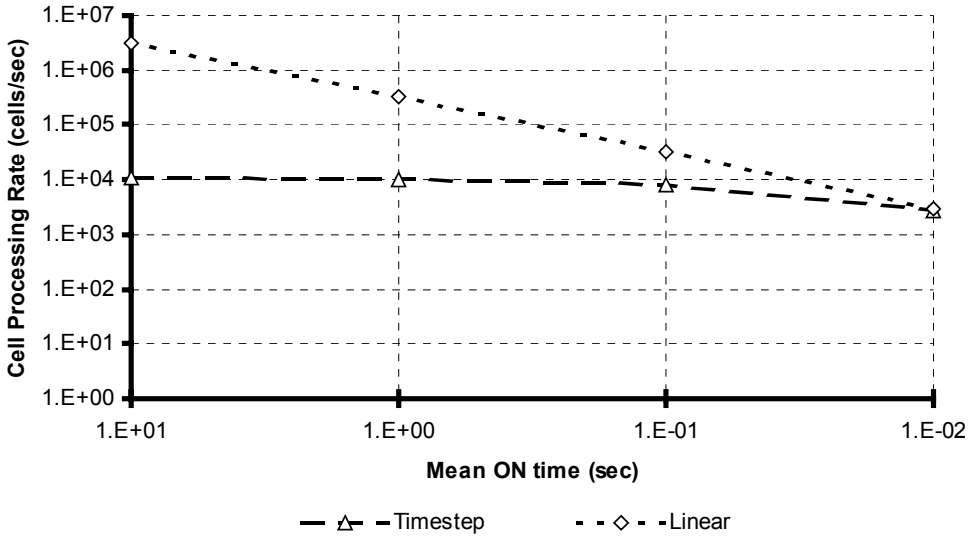

Figure 33 Variation of Cell Processing Rate for 3 Node no Queueing Network

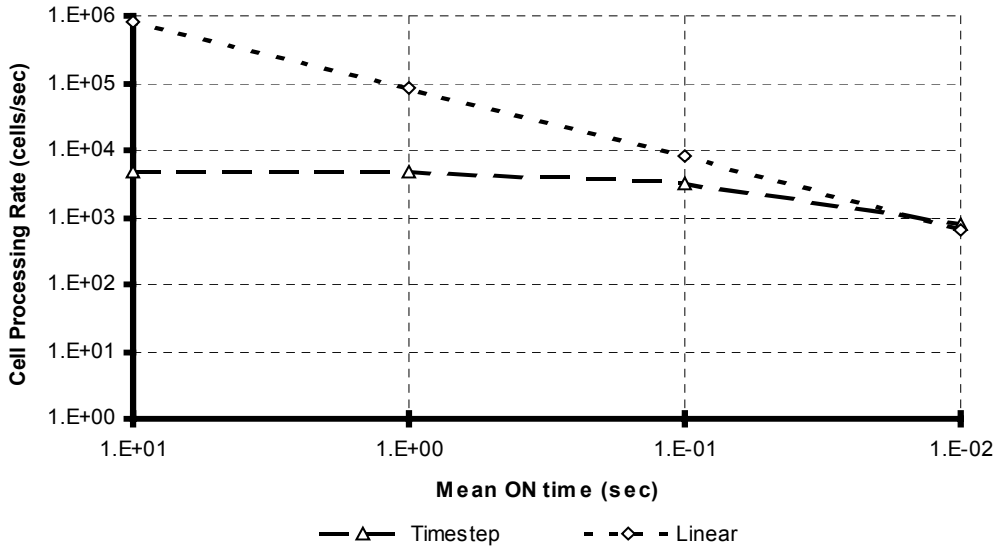

Figure 34 Variation of Cell Processing Rate for 11 Node no Queueing Network

The significant feature to note from these results is that, although the timestepping simulator is much slower than the linear event list simulator with small numbers of source traffic events per timestep, it can actually be faster with large numbers of source events (demonstrated

further in later experiments). This is because, in the timestepping simulator the event list is spatially decomposed. If the number of nodes in the simulation is increased from three to eleven, the total number of events propagating through the network will increased by a proportionate amount. In the timestepping simulator, the event lists will remain approximately the same size because the increased number of events will be distributed approximately evenly amongst more lists (the sorted node queues of each node model). However, since all of the events in the linear event list simulator are stored in one list, it would be expected that the list would be much longer. Hence, the time required to insert each event in the event list increases.

6.4.3 Comparative Performance of Timestepping with Queueing

In this experiment, two sets of simulations were performed for each of the network topologies. For each of these simulations, whilst the peak cell rate of each VCC remained constant, the mean ON and mean OFF times were varied from 10s to 0.01s.

6.4.3.1 Low Utilisation

The low utilisation networks represent networks that are not heavily loaded, but never-the-less experience some burst scale queueing. For both the three node and eleven node networks, the service rate of SW1 was set to 800 cell/s. All other switch server rates were set to 1000 cell/s.

Figure 35 and Figure 36 show how the cell processing rate varied as the mean ON and mean OFF times of the traffic source were reduced. These results differ very little from the no queuing case. This is because, at low utilisations, whilst it is possible for queueing to occur, this does not happen with sufficient frequency to generate substantially increased numbers of events in the simulation.

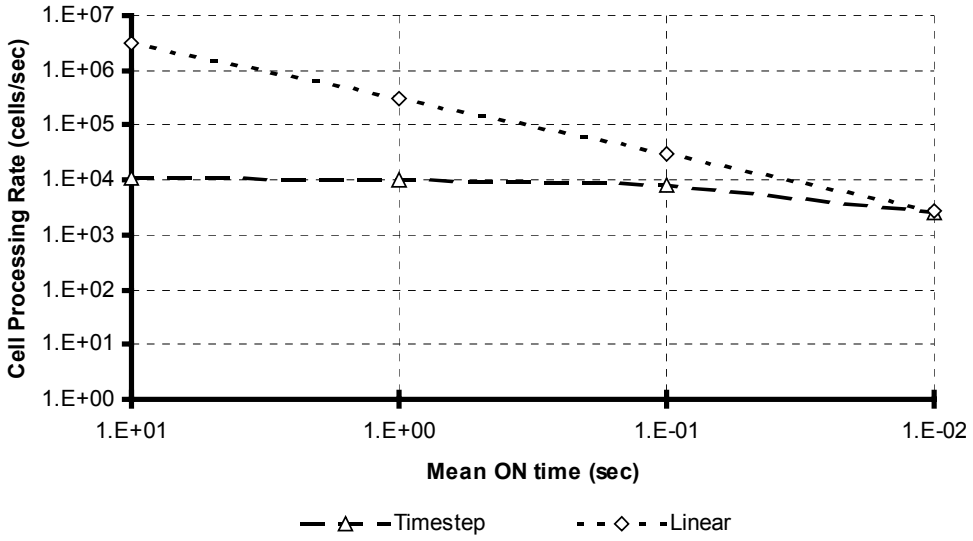

Figure 35 Variation of Cell Processing Rate for Low Utilisation 3 Node Queueing Network

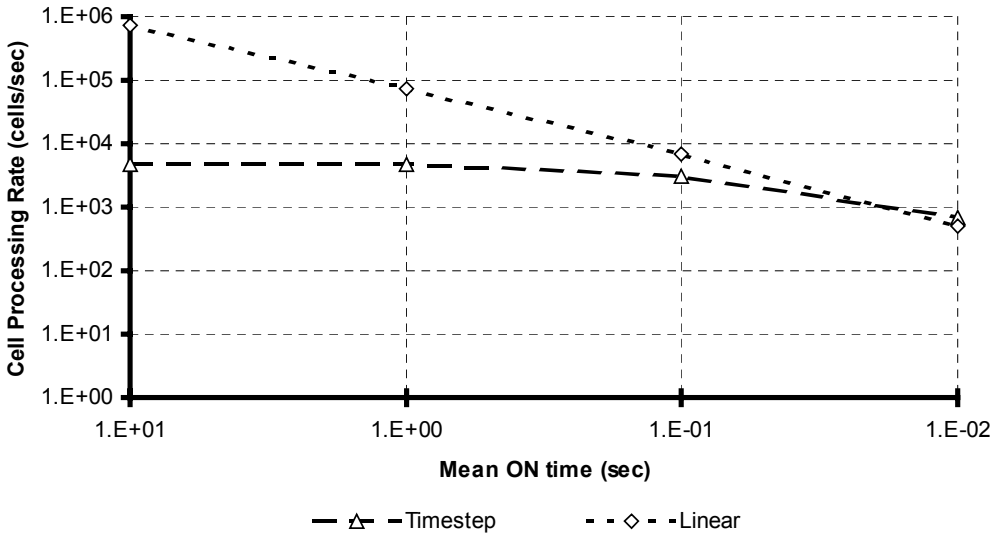

Figure 36 Variation of Cell Processing Rate for Low Utilisation 11 Node Queueing Network

6.4.3.2 High Utilisation

The high utilisation experiments represent periods of very heavy loading in which burst scale queueing is very likely. For both the three node and eleven node networks the service rate of SW1 was set to 500 cell/s. In the eleven node network the service rate of SW2 was set to 400 cell/s. All other switch server rates were set to 1000 cell/s.

Figure 37 and Figure 38 show how the cell processing rate at SW1 changed with varying values of man ON and mean OFF time of the traffic source for the highly utilised network.

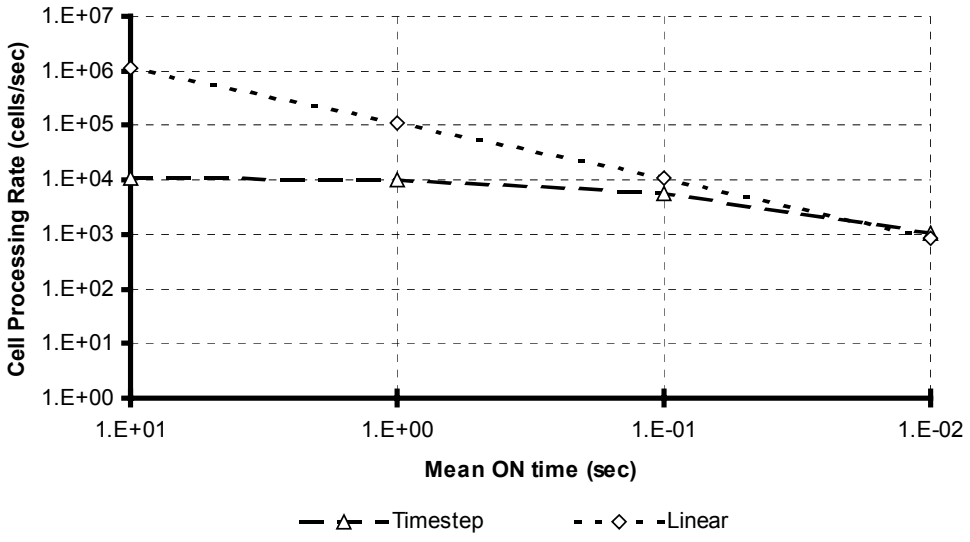

Figure 37 Variation of Cell Processing Rate for High Utilisation 3 Node Queueing Network

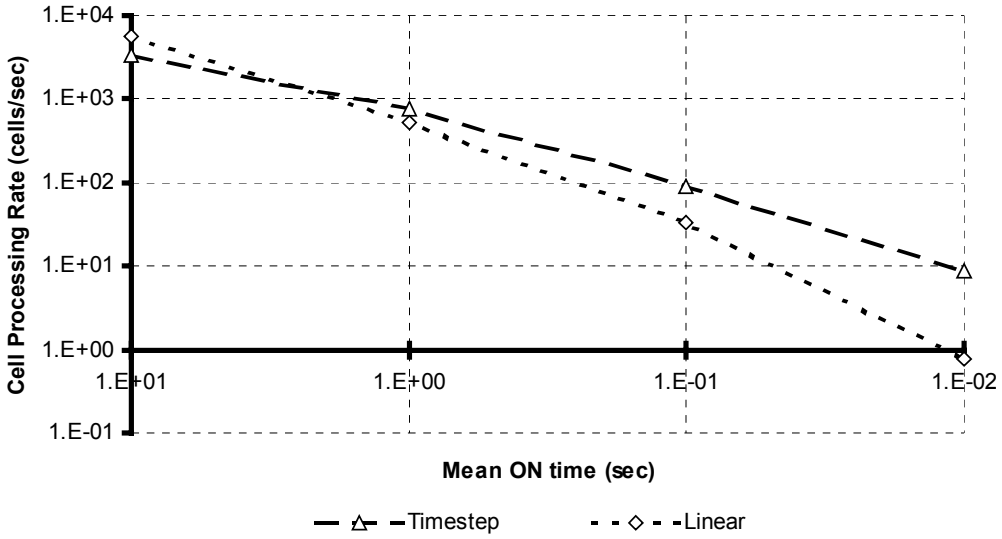

Figure 38 Variation of Cell Processing Rate for High Utilisation 11 Node Queueing Network

The significant feature of these results is that, in the high utilisation network simulation running on the linear event list version of SPROG, the cell processing rate decreases more rapidly than in either the low utilisation or in the no queueing networks. This is particularly true for the eleven node network. Furthermore, the curves for the timestepping and linear event list simulator cross at a point with a lower number of source events per timestep than in the low utilisation and no queueing cases. This is because burst scale queueing increases the number of events propagating through the simulator. From the analysis of queue behaviour presented in Section 4.1.3 of this thesis it can be shown that an event on one VCC passing though a buffer where queueing either begins due to this event, or is currently occurring, will cause a corresponding output event on every other VCC passing through that buffer. This substantially increases the number of events. As with the no queueing case, when there are sufficiently large numbers of event to process, the timestepping simulator is faster than the linear event list version because the spatial decomposition of the event list reduces the time taken to insert each event.

In the above experiments, the grouping of simultaneous events in the timestepping simulator was not implemented. However, the speed of the timestepping simulator could be increased in the high utilisation case by implementing grouping. This is because grouping would eliminate spurious simultaneous events propagating through the simulation. Furthermore, if some error in cell loss results is considered acceptable, then the timestep can be increased to some value that is greater than the link delay.

In order to demonstrate the effect of grouping on the speed of a timestepping simulator, a grouping scheme similar to that used in the ICM simulator (Chapter 5) was implemented in SPROG. In this scheme, an event inserted into the stream queue of a given place is assigned a *group ID* if it is found to have the same scheduled arrival time as an existing event in the queue. Note that, in the ICM implementation it is possible to specify a *group tolerance* whereby all events scheduled within a specified tolerance of a given time are considered to be simultaneous and are grouped together. However, here only events with the same scheduled arrival time are grouped.

The group ID specifies whether an event is the first, last, or within a group of simultaneous events. When the first event in a group is delivered to the switch model in SPROG it is stored and control is immediately returned to the simulation server without that event actually being processed. This continues until the last event in a group is delivered. The switch is then able to process all of these events together and hence to assess their overall effect on the state of the queue.

Although the grouping mechanism in the timestepping scheme eliminates the build up of multiple simultaneous events in the simulation, a given place must still be invoked for every event in a group, and events on different virtual channels must be passed back to the server individually even if they are on the same outgoing link and are simultaneous. An

optimised grouping mechanism for sequential cell rate simulators that removes these inefficiencies is described in Chapter 7 of this thesis.

Figure 39 shows the effect on the cell processing rate of the timestepping simulator of introducing a grouping mechanism for the eleven node highly utilised network. The effect of increasing the timestep size from one to ten times the link delay (and hence introducing some error into any cell loss measurements) is also shown. The source traffic was as before.

The results show that grouping does indeed increase the speed of the simulator because spurious simultaneous events are eliminated. However, because the total number of events in the simulation is reduced, there are more timesteps in which no useful processing work is done for large source mean ON times. This explains why the curve for the for the grouped timestepping case decreases less rapidly than the simple timestepping case at first. Indeed, the introduction of grouping has given rise to results that resemble those of the lower utilisation networks described above.

Increasing the size of the timestep to ten times the link delay increases the speed of the grouped timestep simulator because less processor time is now wasted 'counting' empty timesteps. Therefore a higher proportion of processor time is spent processing events, giving rise to a steeper curve at low source mean ON times.

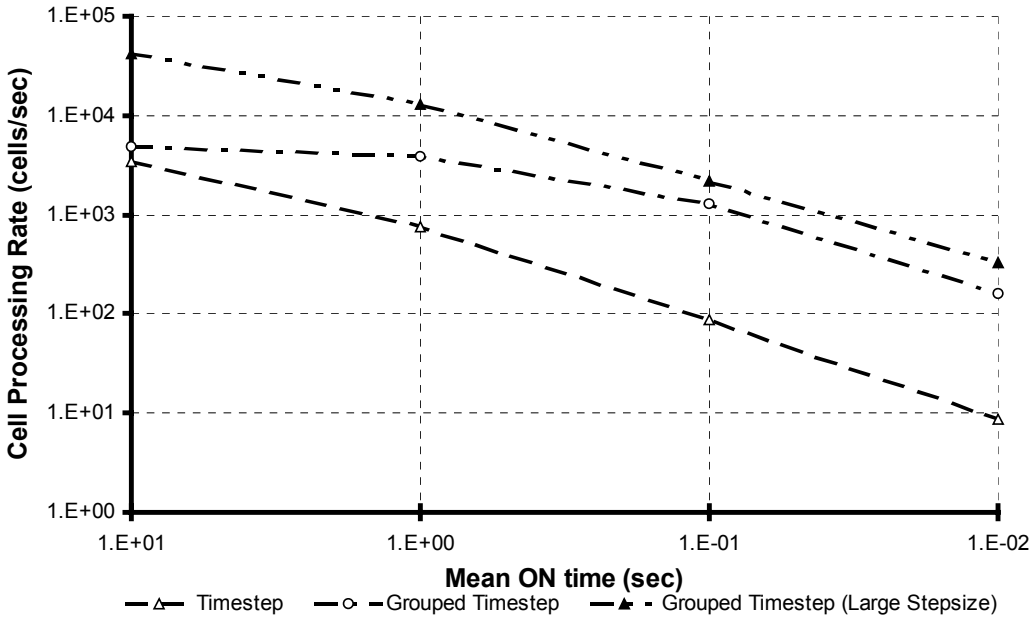

Figure 39 Comparative Cell Processing Rates of Timestepping with Grouping and Large Timestep

6.5 Summary

In this chapter, a new cell rate ATM network simulator (SPROG), developed by the author, has been presented. The development of SPROG was motivated by the need for a highly flexible research simulator in order to compare the performance of timestepping with other time synchronisation schemes fairly. This flexibility was achieved through the object oriented design of the simulation platform and models and the implementation of the simulator in the C++ programming language. The design of the simulator enabled the timestepping approach to be compared with a basic linear event list scheme in a sequential simulator. The results of the study presented in this chapter demonstrate that, whilst timestepping may be inefficient in small lightly loaded network scenarios, for large network scenarios there are sufficient events propagating through the simulation to minimise the number of timesteps in which no

useful event processing work is done. This is particularly the case for a heavily loaded network where queueing occurs in the buffers. Indeed, the results show that a sequential timestepping simulator is actually faster than a sequential simulator employing a simple linear event list for large networks. This is because, for a heavily loaded simulator with a linear event list, the insertion time of each event into the event list becomes very large. Henriksen [Henr83] describes the spectacular failure of such "linear-search-in-descending-time-order" algorithms.

Timestepping gives speedup when there are sufficiently large numbers of events because the event list is broken into smaller segments, each of which is associated with a particular node (the *sorted node queue*). Therefore, when an event is inserted into the list the search length of the sorting algorithm is much reduced. The exploitation of this effect in a purely sequential simulator where there is no overhead at low event loads of timestepping is discussed in Chapter 7.

The results presented in this chapter suggest that a coarse spatial decomposition scheme is required if timestepping is to be used as the time synchronisation scheme in a parallel cell rate ATM network simulator. Such a scheme would maximise the number of events processed per timestep and hence minimise the processing overhead of timestepping.

7. Efficient Event List Management for Cell Rate Simulation

The study described in the previous chapter of this thesis demonstrated the importance of an efficient event list management algorithm in simulator design. Indeed, despite the fact that the study was primarily intended to assess the suitability of timestepping for the parallel cell rate simulation of ATM networks, it has also highlighted the major contribution that the event list management scheme can make to the speed of a sequential simulator.

In this chapter, a review of the many event list management algorithms (in general, intended for sequential simulators) described in the literature is presented. The literature on this subject is both extensive and comprehensive and hence only a brief review of some of the important survey papers is given here. This is followed by a description of a study, using the SPROG simulator, of improved event list management schemes for cell rate ATM network simulators. The study takes some of the principles of the timestepping scheme, in particular the spatial decomposition of the event list, and applies them to cell rate ATM simulation in a sequential computing environment.

7.1 Review of Event List Management Schemes

7.1.1 General Simulation Algorithms

Many papers exist that describe a wide variety of event list algorithms. These range from variations on the linear linked list to more complex binary trees. Comfort [Comf79] details seven categories of event list

algorithm and analyses the performance of examples of each. The categories are:

1. *Linked List* - A time-ordered doubly-linked list in which events are inserted by searching from the event scheduled for the most distant future time to the next scheduled event.

2. *Indexed List* - Dummy events are inserted at known intervals that act as indexes in order to rapidly identify the correct insertion point for an event.

3. *Adaptive Linked List*- This is a variation of the linked list that is able to adapt to the distribution of event inter-arrival times.

4. *Indexed/Adaptive List* - This is a combination of (2) and (3) above.

5. *Multiple (Dynamically Changing) Adaptive Linked List* - an extension of (3) above to allow the creation of multi-layer keys to the event list.

6. *Heap Data Structure* - A restricted binary tree in which events inserted with the same time stamp will be removed in an unpredictable order.

7. *FIFO Heap Data Structure* - A variation on (6) that preserves the insertion order of events with the same time-stamp.

In common with much of the literature, Comfort uses a *hold model* to assess the performance of an algorithm rather than using a real simulation problem. A hold model involves rescheduling the event with the smallest next processing time to some later time according to some incremental processing time distribution. The poor performance of the linear linked list is demonstrated and it is shown that, in general, indexed and adaptive linked lists are the most efficient.

Henriksen [Henr83] discusses a range of event list algorithms and describes the 'spectacular' failure (when the event list is large) of simple linear linked lists that search from next scheduled event to most distant future event.

Kingston [Kings89] analyses the performance a number of binary tree search algorithms and, although he finds these to be disappointing, is unable to recommend any alternative. Other reviews of binary tree algorithms are provided by Evans [Evans86], McCormack [McCor81] and Nikolopolous [Niko93].

In general, advanced event list management algorithms are intended to provide speedup when compared with the poor performance of the simple linear linked list (as demonstrated in Section 6.4.1 of this thesis). However, as Jones points out in [Jones86], whilst the performance penalty due to choosing a poor algorithm may be great, the gain achieved by selecting between particular fine-tuned algorithms may be very small. Indeed, in such cases development effort may be better spent in optimisation of the models rather than the event list.

7.1.2 Optimised Event Lists for Cell Rate Simulation

The Linksim cell rate ATM simulator [Pitts93] contains an event list management algorithm that has been designed for optimal performance when used with cell rate modelling of the traffic. The event list algorithm in Linksim takes account of the generation of simultaneous events at a queue by storing these events together. When these events are subsequently processed by a model, they can be passed to it as a single group. This allows that model to avoid the generation of multiple spurious 'knock-on' events.

The structure of the Linksim event list is shown in Figure 40. Linksim adds two extra dimensions to the simple linear event list structure. The first of these extra dimensions contains a list of records of where the

simultaneous events occur together with a pointer to the linked list of simultaneous events that occur at a place. This constitutes the second extra dimension. Events are classified at two levels: *general* and *specific*. The 'general' classification categorises events as either those that occur at a network termination (NT), or those that occur at a network element (NE). The 'specific' category identifies the details of the actual event. Specific NT events include send and receive cell rate changes, while specific NE events include those that describe input and output cell rate changes and those representing changes in the state of the cell rate queue model of the NE.

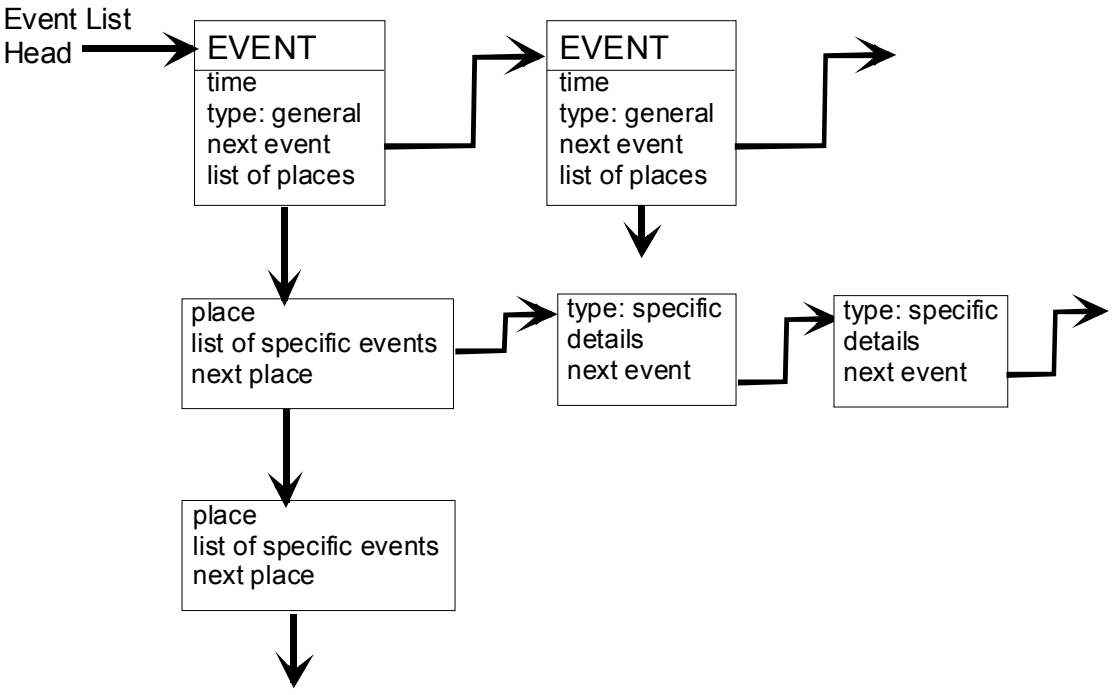

Figure 40 Linksim Event List Structure

In Linksim, when the simulation is advanced by one event, the procedure actually processes all of the events happening simultaneously at either all network terminations or all network elements. The significance of simultaneous cell rate changes in cell rate simulation was introduced in Section 5.1.1 of this thesis.

The processing of simultaneous events on the input to a queue as a group is an important factor in the design of efficient cell rate simulators. Consider the simple scenario of Figure 41. A source produces cell rate changes on ten VCCs that pass through a chain of 3 buffers, each consisting of a single queue/server pair. Consider a single cell rate change on one of the VCCs on the input to Buf1. If Buf1's queue is not empty then simultaneous cell rate changes will appear on the output of the queue on each on the ten VCCs . If Buf2 is also in queueing, then a cell rate change event will appear on every VCC emerging from its queue for each rate change at the input to the queue. Therefore one hundred cell rate change events will appear at the input to Buf3. In a worst-case scenario, it is possible that the queue in Buf3 will not be empty either, giving rise to a thousand cell rate change events on the output of the queue!

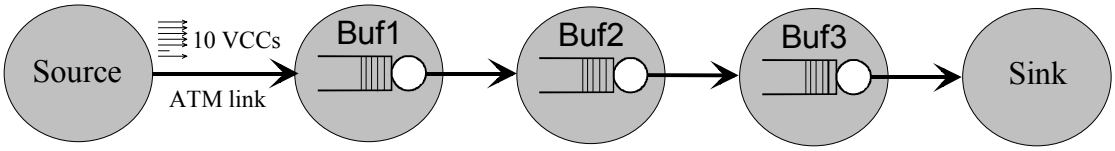

Figure 41 Example Network Scenario

If all of the simultaneous input events at Buf2 are accounted for before the effect of these on the cell rate queueing model is calculated, then only one output event per VCC need be generated.

It can be seen that, because Linksim processes simultaneous events on the input to the queue together, it is able to calculate the overall change in the input rate to the queue before it calculates any change in the output cell rate. Therefore spurious simultaneous rate change events are eliminated.

In summary, the Linksim event list structure is designed to account for both the spatial distribution, and the simultaneous occurrence of events. The implications for simulator performance of an event list that accounts

for these two factors is investigated in Section 7.2 and Section 7.3 of this thesis.

7.2 A Spatially Decomposed Event List Scheme

In Chapter 6 of this thesis it was shown that, under heavy event loads, timestepping simulators exhibit speedup over simple linear event list simulators. This is because the spatial decomposition of the event list reduces the number of events though which the sorting algorithm must search when an event is inserted. However, at low event loads there is a significant overhead of processing timesteps in which no useful work is done and timestepping simulators are therefore slower than linear event list simulators.

In order to fully exploit the effects of spatially decomposing the event list in a sequential simulator it is necessary to dispense with the timestepping mechanism. The advancement of the simulation time should be based solely on the time of the next scheduled event and not on the start time of the next timestep. For this, an event list with two dimensions, one of *space* (the place where the event occurs) and one of *time* (the time at which an event occurs) and with a *next event* time advancement strategy is required. Here, separate event lists are maintained for each place (the *sorted node queues*). The places are linked together in a doubly-linked list that is time ordered according to the time of the first event scheduled for each place. This enables the next scheduled event at each place to be identified efficiently. There is also an array of pointers that point to every place in the network, whether or not those places currently have events in their sorted node queues. This scheme is illustrated in Figure 42.

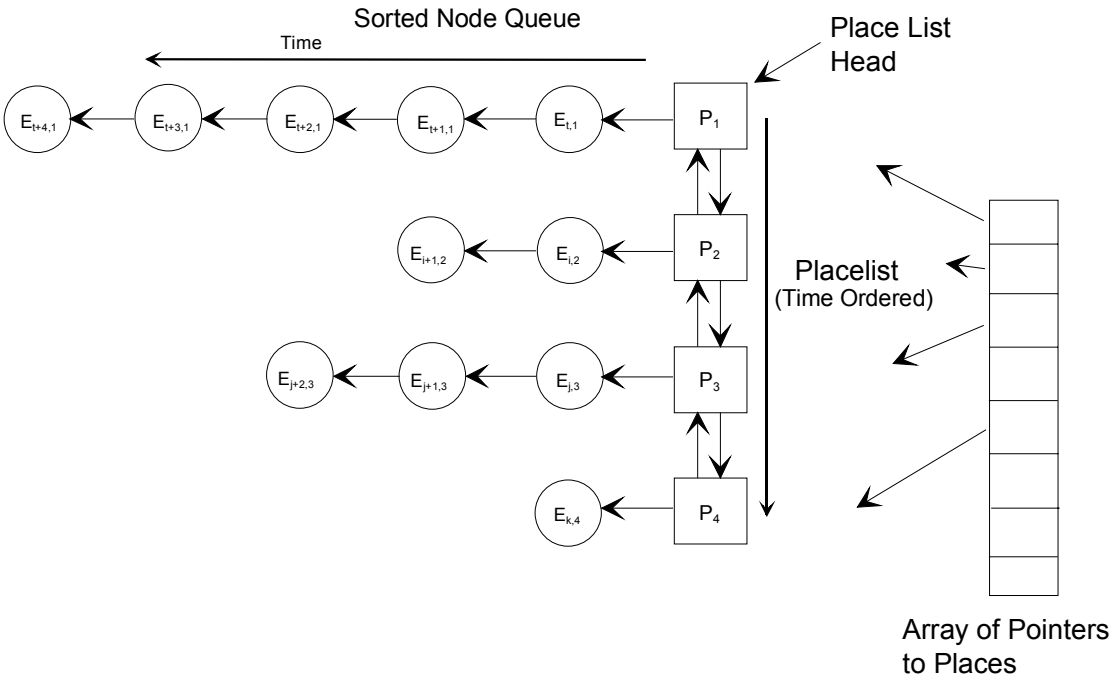

Figure 42 Structure of Space-Time Event List

The procedure for the insertion of an event in the event list is described in Table 6. Once the destination place of an event has been identified, then it can be placed in time order in the sorted node queue of that place. This operation does not involve a lengthy search of the place list since each place is indexed using the array of pointers to places. If the event was inserted at the head of the sorted node queue then it is likely that the place list will no longer be in correct time order. This place must therefore be removed from the place list and reinserted in correct time order. If, however, the sorted node queue was empty before the insertion of this event, then this place will not currently be in the place list and hence needs to be inserted in the correct time order. If this place has the earliest event at the head of its sorted node queue, then it will be placed at the head of the place list.

```
IF outlink destination == this place
      Insert in SortedNodeQueue of this place
ELSE
      Insert in SortedNodeQueue of destination place

IF SortedNodeQueue != empty && event inserted first
      Remove destination from PlaceList and reinsert in correct order
ELSE
      IF SortedNodeQueue == empty
            Insert destination in correct order in PlaceList
```

Table 6 Sequence of Operations for Event Insertion

The procedure for the extraction of the next scheduled event from the event list is described in Table 7. The place with the next scheduled event is pointed to by the place list head. Once the first event in that place's sorted node queue has been removed for processing, the place list may no longer be in correct time order. If there are still events in this place's sorted node queue then this place is removed from the place list and reinserted in correct time order. However, if the sorted node queue of this place is now empty, then this place is removed from the place list and the place list head updated accordingly. Further details of this operation are shown in the table.

```
Place with next scheduled event = PlaceListHead
      NextEvent = SortedNodeQueueHead at PlaceListHead

IF sorted node queue != empty && next place != NULL
      Remove this place from Place List
      Reinsert in correct time order in Place List

ELSE IF sorted node queue != empty && next place == NULL
            This place is next
      ELSE
            Remove this place from place list
```

Table 7 Sequence of Operations for Event Extraction

7.2.1 Performance of Spatially Decomposed Event List

In order to assess the impact on simulator performance of spatially decomposing the event list, the space-time event list management scheme described above was implemented in SPROG. New functions for sending and retrieving events were implemented by overloading the existing event list management functions provided by `Class Event`. Note that in this implementation no account was taken of the generation of multiple simultaneous events in the cell rate model during queueing. A scheme for dealing with this is described in Section 7.3.

The cell processing rate for both the three node and the eleven node networks described in Chapter 6 of this thesis, was measured for the high and low utilisation queueing and the no-queueing cases. This was compared with the results obtained previously for the linear event list and the timestepping scheme. The traffic characteristics were varied as before. The switch server rates and link delays were also as before.

Figures 43 to 48 show how the cell processing rate for the space-time event list varied with event load for the no-queueing and the high and low utilisation queueing cases. These results demonstrate that, at low event loads the cell processing rate of the space-time event list is not significantly greater than that of the simple linear event list. However, it does give substantially better performance than the timestepping scheme. This is because the overhead of processing idle timesteps has been removed in the space-time event list. However, with few events there will be less chance of there being events scheduled for more than one place at any given time. Therefore the performance gain from spatially decomposing the event list is small in comparison with the linear event list, resulting in similar cell processing rates.

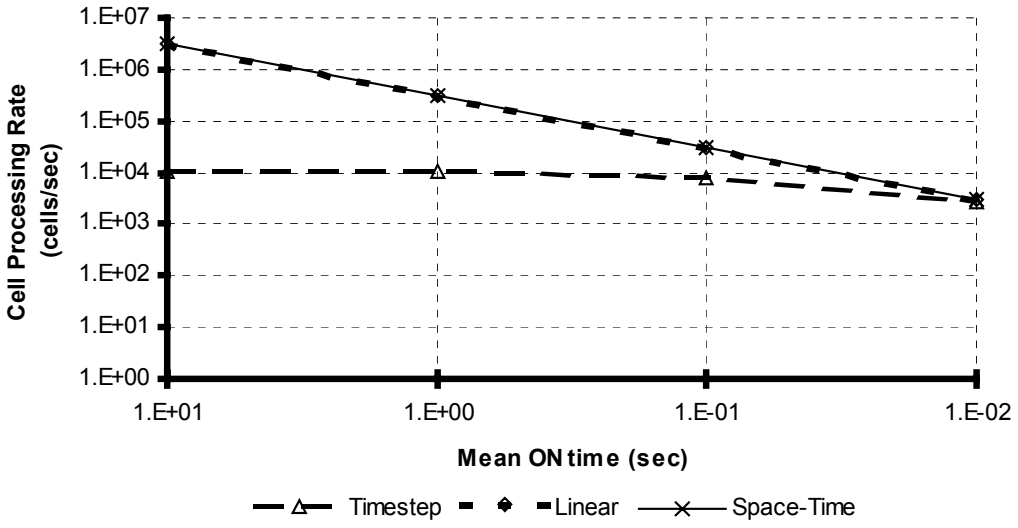

Figure 43 Three Node Space-Time Event List Performance with No Queueing

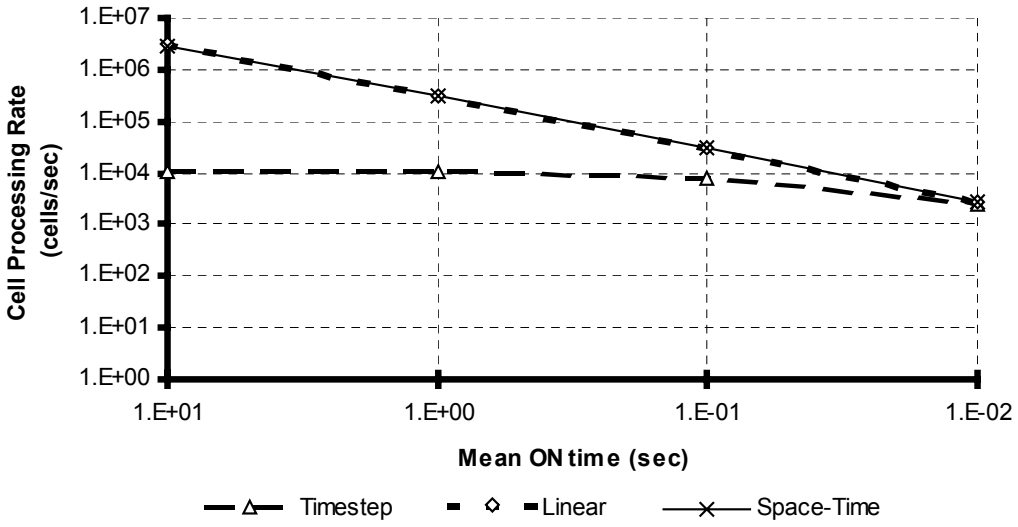

Figure 44 Three Node Space-Time Event List Performance With Queueing and Low Utilisation

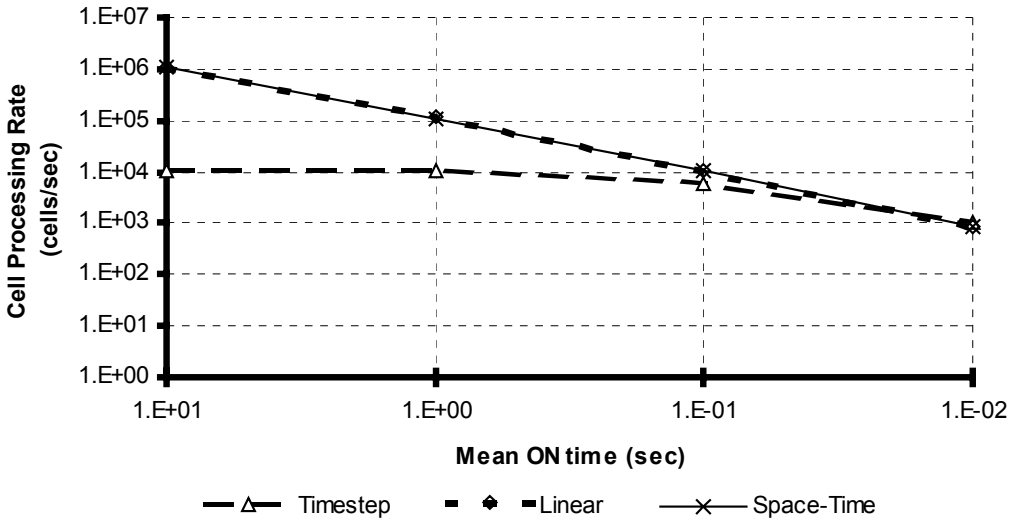

Figure 45 Three Node Space-Time Event List Performance with Queueing and High Utilisation

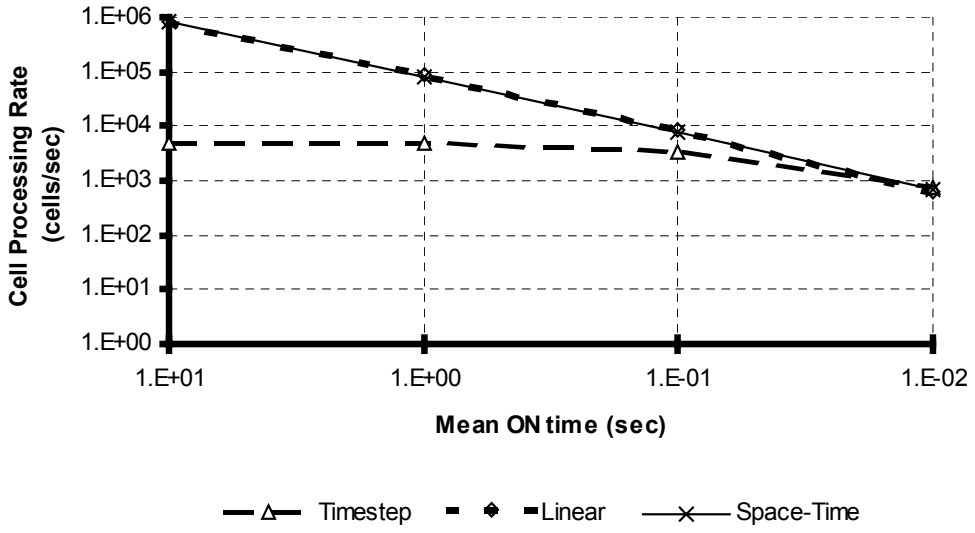

Figure 46 Eleven Node Space-Time Event List Performance with No Queueing

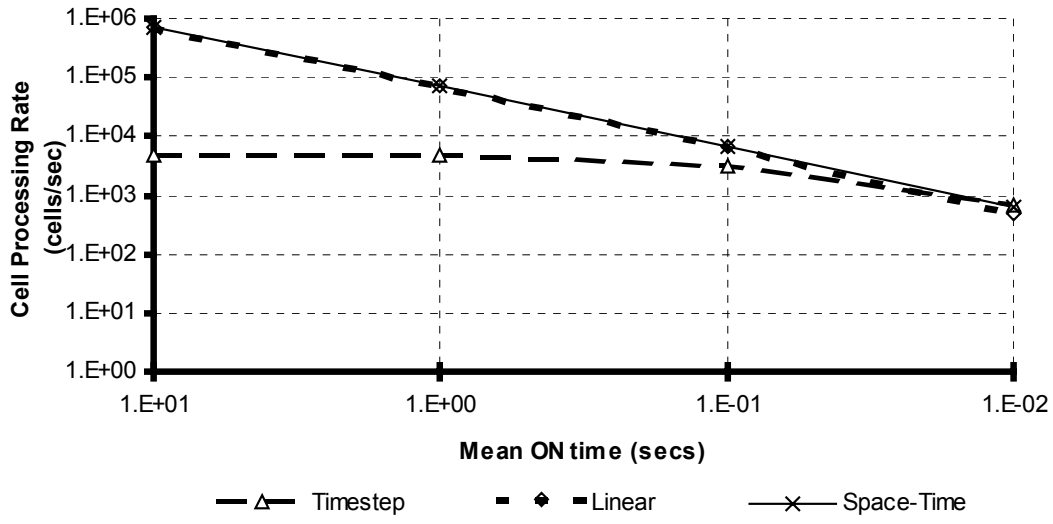

Figure 47 Eleven Node Space-Time Event List Performance with Queueing and Low Utilisation

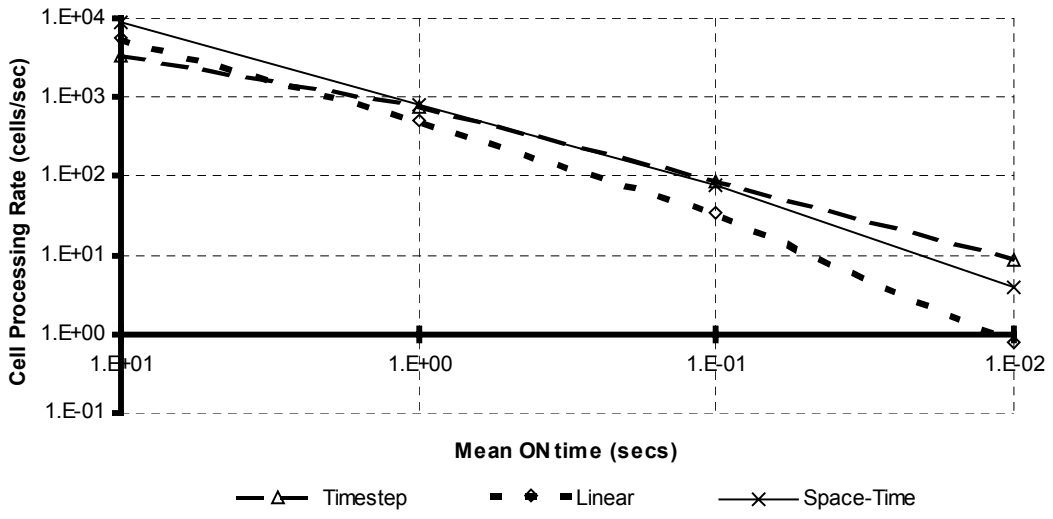

Figure 48 Eleven Node Space-Time Event List Performance with Queueing and High Utilisation

As the event processing load is increased, the cell processing rate of the space-time simulator does not decrease as rapidly as that of the linear event list simulator. This is because of the shortened list searches (occurring each time an event is inserted into the event list) that are a

feature of spatial decomposition. However, the performance of this event list management scheme is not as good as timestepping for high event loads because of the additional overhead of maintaining the time order of the place list.

7.3 Space-Time Event List Management for Cell Rate Simulation

In the previous section, a basic spatially decomposed event list scheme was presented. This scheme reduces the average number of events through which the event list algorithm must search each time a new event is inserted. However, whilst this scheme achieves spatial decomposition without the overhead of idle timesteps at low event loads that accompanies the timestepping scheme, and gives some speedup over the linear event list algorithm, that speedup is small. This is particularly disappointing given that the linear event list is, in general, considered to be the least efficient scheme possible [Henr83]. The reason for the poor performance of the event list schemes described above, particularly during periods of cell queueing, is that they are unable to deal effectively with the generation of multiple simultaneous events such that buffer models in queueing do not generate further (spurious) 'knock-on' events.

In Section 6.4 of this thesis, the application of an event grouping mechanism to eliminate spurious knock-on events was demonstrated. The experiments described here show that substantial speedup can be achieved because of the consequent reduction in the event load. In timestepping, the event list structure is inherently spatially decomposed and hence it is a relatively simple task to identify the events that are scheduled for the same arrival time at a given place. However, because the number of events has been reduced by the grouping mechanism, the simulator now spends a large proportion of its run time in 'empty' timesteps (as with the low utilisation cases). For a simulator running on a

sequential computer, this is clearly a sub-optimal situation. However, the proper management of simultaneous events is still vital.

Because the place at which events are scheduled to occur is inherent in the structure of the space-time event list, this event list can be enhanced to enable all of the events that occur simultaneously at a place to be delivered to a place in a single group This will enable that place to process the events together. Figure 49 shows the enhanced space-time event list structure. As with the Linksim event list (Figure 40), an extra dimension is used to represent all of the events scheduled to occur at a given place at a given time. Each event in the sorted node queue contains a pointer to a linked list of all of the other events scheduled there at that time. When an event is inserted in the sorted node queue, if an event is found with the same scheduled time, then the new event is inserted in the 3rd dimension of the list. When a group of simultaneous events is delivered to a place, the event at the head of the group is removed from the sorted node queue with the links to the simultaneous events remaining intact, thus passing the whole group to the place. This algorithm is more efficient that both the Linksim and ICM approaches for dealing with simultaneous events because:

1. Only one event extraction operation is required for n simultaneous events.

2. A place modelling, for example, a buffer is able to calculate its total incoming cell rate in a single invocation, thus enabling it to generate just a single, correct, cell rate change event per VCC on its output.

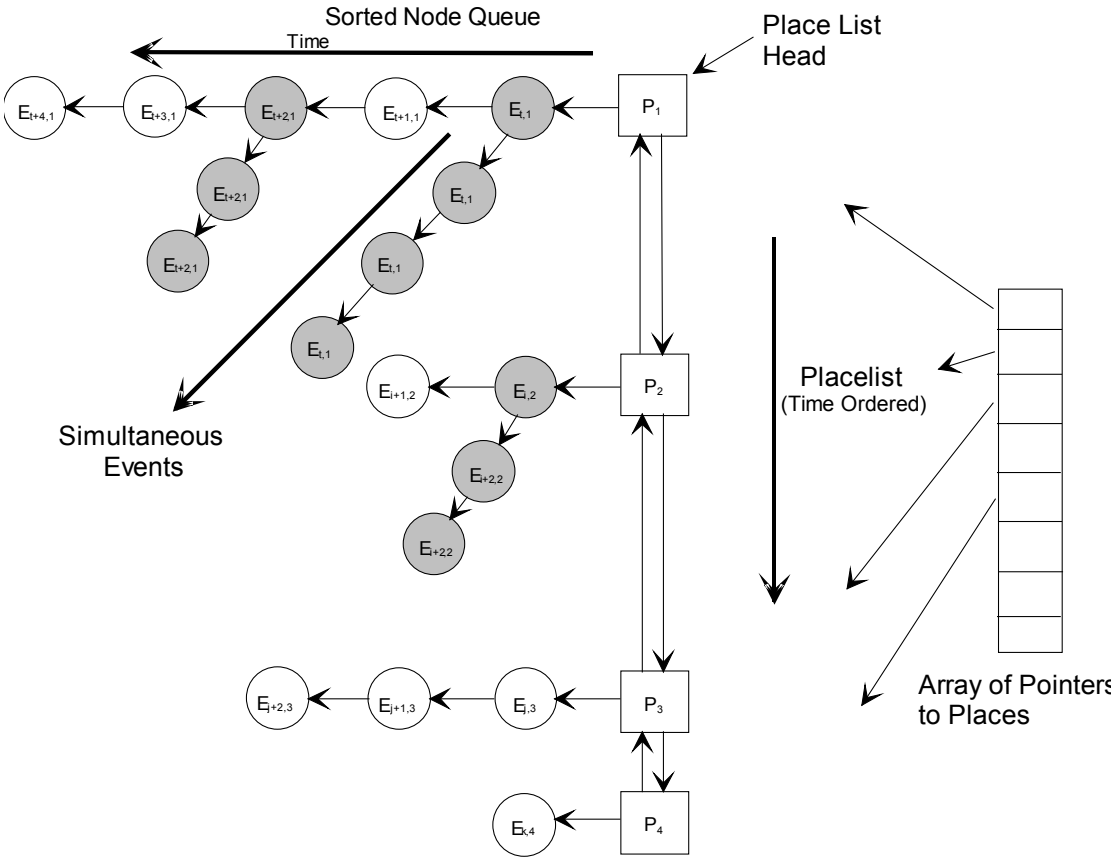

Figure 49 Event List Optimised for Cell Rate Modelling

This scheme is similar to the Linksim event list algorithm. However, whilst Linksim enables events to be delivered as a simultaneous group, only a single event is inserted into the event list at a time. Therefore, a simultaneous cell rate change on n VCCs output from a buffer will cause n event insertion operations.

Consider a buffer in which cells are queued. If there is a simultaneous cell rate change on all of the output VCCs that are routed on to a given output link then the buffer model can build the list of simultaneous events and pass that to the event list algorithm. Therefore only a single event insertion operation is required for all n simultaneous events on that output link. Note that, although the simultaneous event groups are built before they are passed to the event list, it is still necessary for the event list algorithm itself to recognise and deal with simultaneous events that

are individually inserted if these are scheduled for delivery to the same place. For example, consider the simple scenario shown in Figure 50.

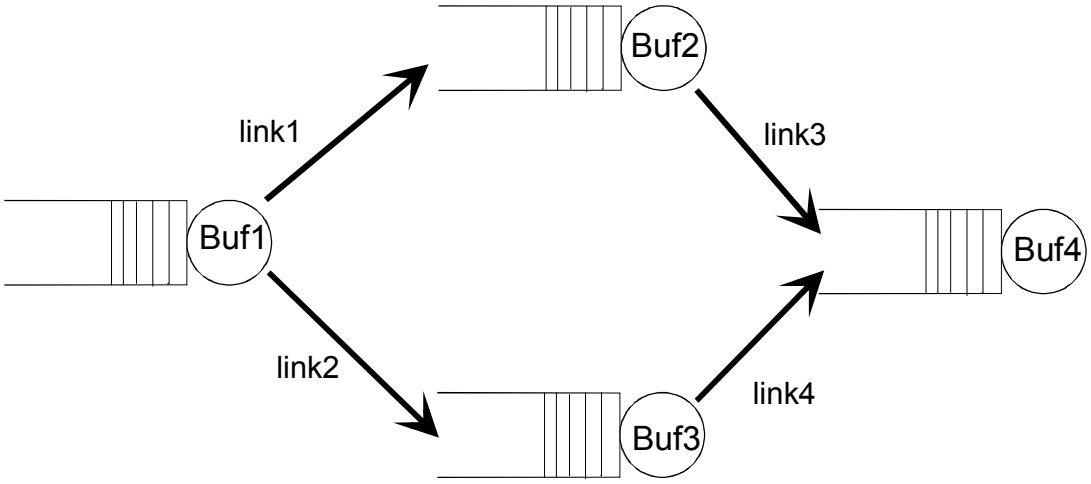

Figure 50 Example Network

Four FIFO buffers are connected by four ATM links of equal delay and with ten VCCs on route Buf1-Buf2-Buf4 and ten VCCs on route Buf1-Buf3-Buf4. All the buffers are served at the same rate and Buf1, Buf2 and Buf3 have cells queued in them. The queue is the same length in Buf2 and Buf3. Therefore, a cell rate change on the input of either buffer will take exactly the same time to appear on the output (this would also be the case if the buffers were simple 'zero depth' models, as used in the experiments above). Consider a cell rate change on a single VCC at the input to Buf1. This will cause a rate change on all 20 VCCs on the output of the buffer (Figure 51). The rate changes will be grouped as two lists of ten events (one list for each output link) by the buffer model itself and passed to the event list. Therefore, only 2 event insertions are required for all 20 events. Each group of ten events is delivered to Buf2 and Buf3, and since they are able to process the events together each buffer produces a single group of ten cell rate changes in its output. The groups of cell rate change events on the outputs of Buf2 and Buf3 will occur simultaneously and therefore must be delivered to Buf4 as a single group. This linking of the two lists is

achieved by the event list such that Buf4 is able to process all 20 simultaneous rate changes on its input together.

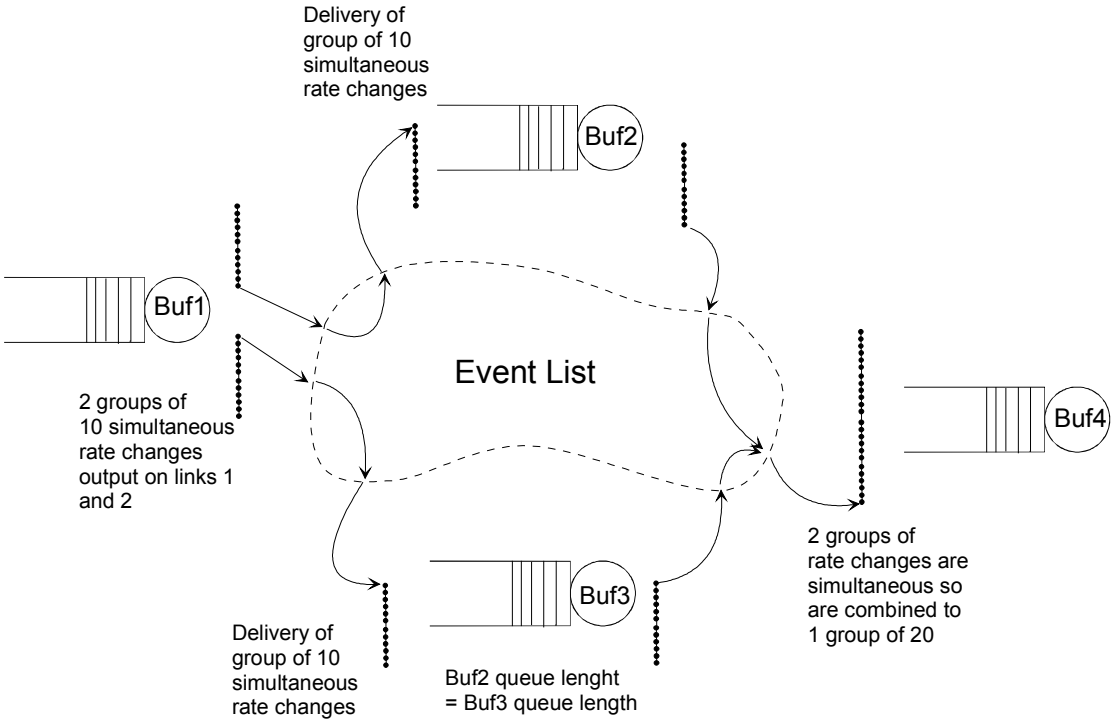

Figure 51 Propagation of Simultaneous Cell Rate Change Events via Event List

7.3.1 Performance of Space-Time Event List for Cell Rate Simulation

In order to demonstrate the speedup that can be achieved through the correct management of simultaneous events, the optimised event list scheme described above was implemented in SPROG. Simulation experiments were conducted for the three and eleven node networks described in Section 6.4.1. The source traffic characteristics were varied as before, but only the scenarios where burst scale queueing occurred were considered. This is because, where queueing does not occur, simultaneous events are not generated by the cell rate queue model unless there are simultaneous input cell rate changes. This is very unlikely, and hence there is little to be gained over and above the basic space-time scheme. The computer platform used for the simulation runs was as before.

Figures 52 to 55 compare the cell processing rate of the optimised space-time event list version of SPROG with the results obtained previously for the simple linear, timestepping, and space-time event list versions. Figure 55 also includes the results obtained previously for the grouped timestepping scheme and the timestepping scheme with the timestep set to ten times the link delay. These earlier results are included here for ease of comparison.

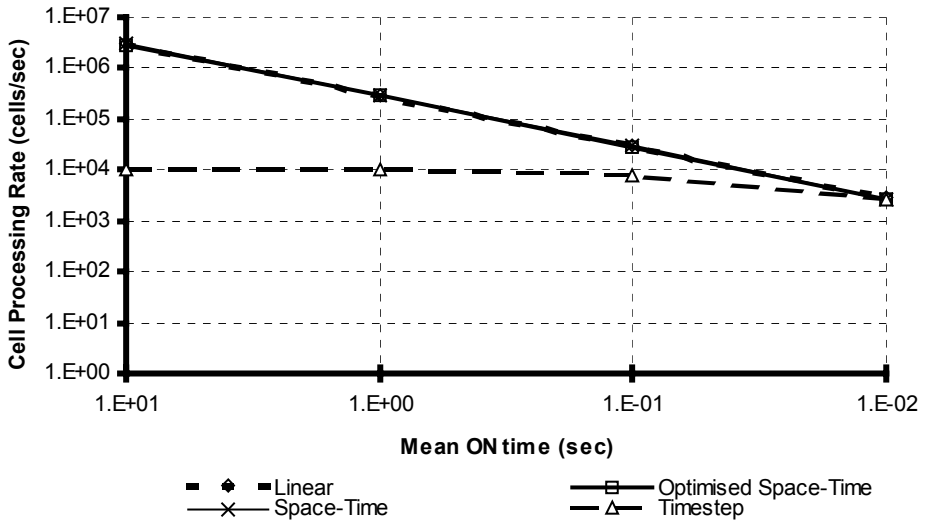

Figure 52 Three Node Optimised Space-Time Event List Performance with Low Utilisation

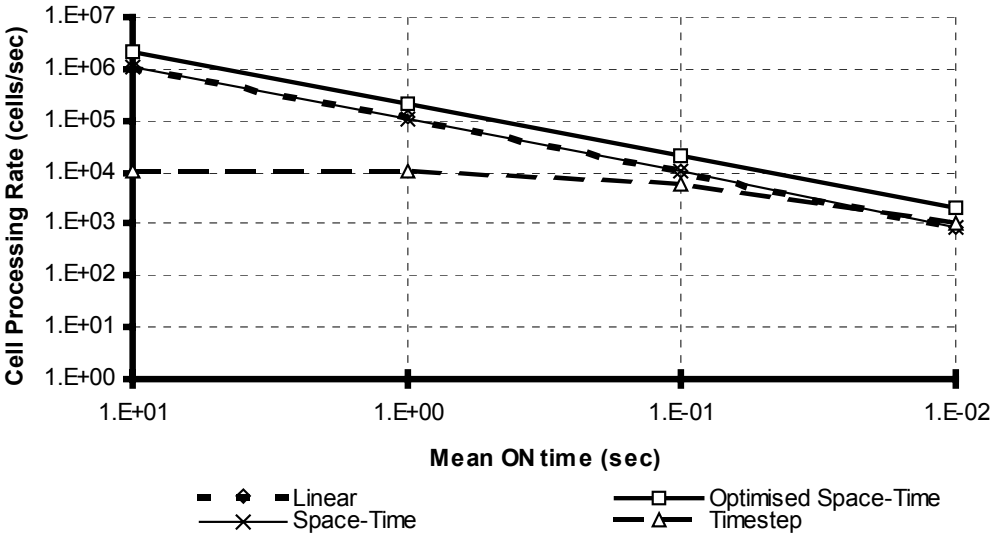

Figure 53 Three Node Optimised Space-Time Event List Performance with High Utilisation

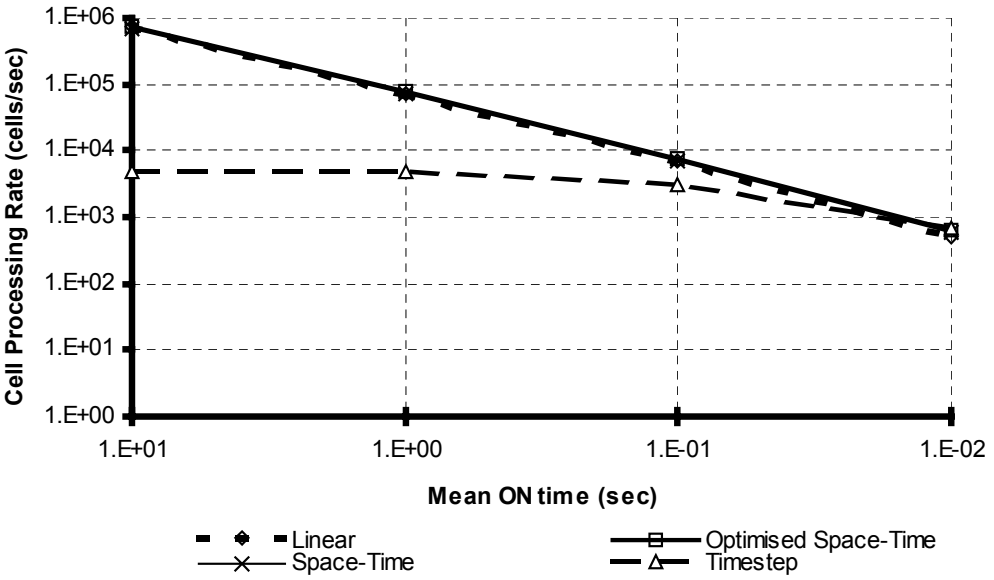

Figure 54 Eleven Node Optimised Space-Time Event List Performance with Low Utilisation

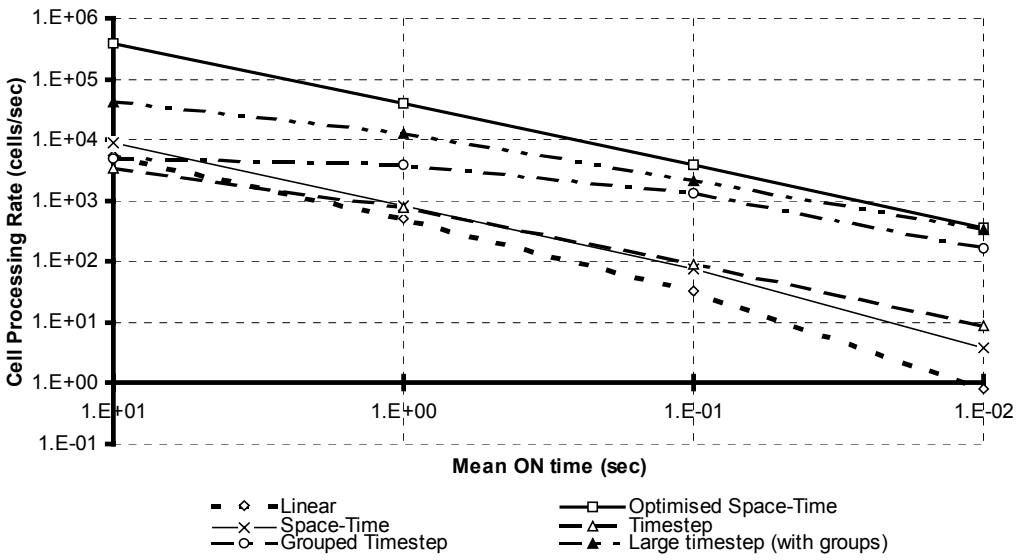

Figure 55 Eleven Node Optimised Space-Time Event List Performance with High Utilisation

In both the network scenarios, the optimised space-time event list consistently outperforms all of the other event list schemes. In the eleven node network in particular, substantial speedups are clearly possible, where up to approximately one hundred-fold increases in cell processing rate over the linear event list are observed. However, it is significant to note that, for very short mean ON (and mean OFF) times, the performance of the timestepping simulator with grouping and a large timestep size tends to that of the optimised space-time simulator (although measurement errors will be introduced, as described previously in Section 5.3.4).

7.4 Verification of Optimised Space-Time Using a Realistic Networking Scenario

While the experiments described above demonstrate that the optimised space-time event list management scheme can give significant speedup, when compared with the simple linear, timestepping, and simple space-time schemes, the results are limited by the fact that the experimental networks are not realistic. These unrealistic scenarios were chosen to simplify the analysis of event propagation and to identify promising candidates for a cell rate event list scheme. A simulation of a realistic network scenario is required to verify the performance of the optimised space-time algorithm.

The network scenario is shown in Figure 56. Twenty switch models and ten ON-OFF traffic sources were interconnected by ATM links. Groups of two switches (one configured as a traffic sink) and a traffic source were used to model the users and customer premises equipment of customer premises networks (CPNs, each one representing the network on one site of a corporate network). The switches and sources within each CPN were connected using links of 10μs delay, while the CPNs were interconnected using links of 100μs delay. 160 VCCs of peak rate 167 cell/s were routed between a randomly distributed pattern of CPN source/destination pairs. The server rates in the switches were all set to 9434 cell/s (queueing only occurring at SW0), while the traffic source ON and OFF times had an exponential characteristic and a mean value of 10s. The cell processing rate at switch SW2 was measured for the linear, timestepping and optimised space-time versions of SPROG simulating the above network. In the timestepping simulator, the timestep was set to 100μs.

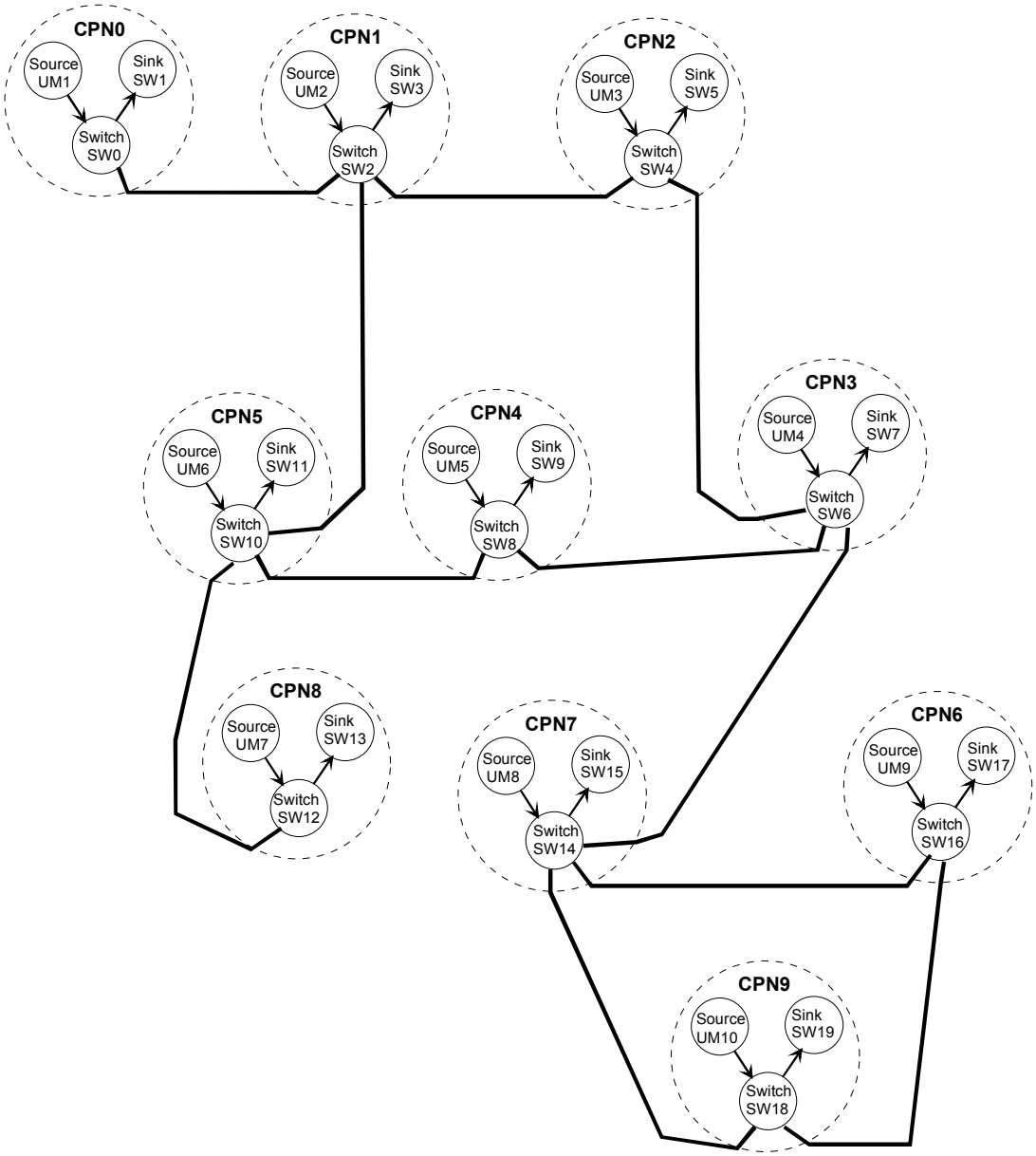

Figure 56 Topology of Realistic Scenario

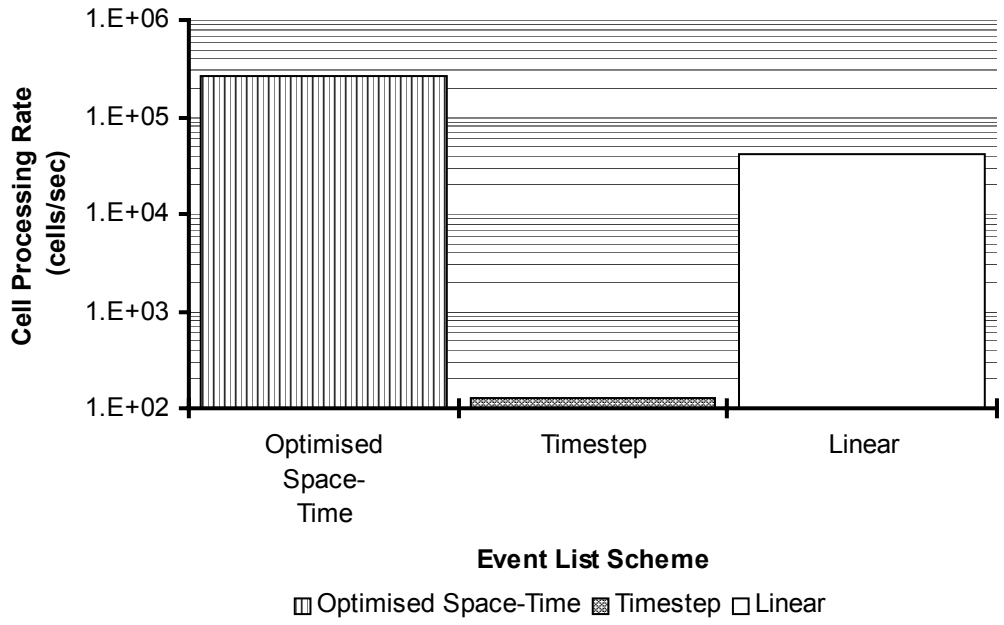

Figure 57 Performance of Event List Schemes in Realistic Scenario

The measured cell processing rates for the linear, timestepping and optimised space-time versions of SPROG are shown in Figure 57. The performance of the timestepping version is disappointing and suggests that there are insufficient events in this network model to allow it to be successfully simulated on a timestepping parallel simulator. The optimised space-time event list exhibits the best performance, being over seven times as fast as the linear event list. However, the speedup over the linear event list is not as great as suggested by the experiments in the previous section of this thesis. This is because the proportion of switch models in which queueing is occurring is lower (equating to the low utilisation case). The advantage of spatially decomposing the event list and managing simultaneous events efficiently is nevertheless demonstrated.

7.5 Summary

In this chapter, the application of some of the ideas introduced in the timestepping study of Chapter 6 to sequential cell rate simulation was described. The existing literature relating to efficient event list management was briefly reviewed. The literature contains a great variety of event management schemes, many giving substantial improvements in efficiency over the simple linear event list scheme. However, many of the schemes are assessed in the context of a generic hold model with no consideration of their application in real simulators. In general, these schemes do not attempt to exploit any particular characteristics of a given simulation modelling scheme and are instead aimed at efficient operation in the greatest variety of applications.

This thesis is specifically concerned with the acceleration of cell rate ATM network simulation, and hence event list management schemes that attempt to exploit the characteristics of this particular modelling technique are of interest. This chapter therefore went on to consider the event list scheme in Linksim, one that is specifically optimised for cell rate modelling. Two characteristics of cell rate modelling were then considered: the spatial distribution of the events, and the occurrence of simultaneous events during burst scale queueing. Event list schemes were implemented in SPROG that exploited these characteristics. Although some speedup at high event loads can be achieved by spatially decomposing the event list, the greatest speedup was achieved when a spatial decomposition was combined with the grouping of the delivery of simultaneous events to node models. This enables the cell rate buffer models to account for all of the cell rate changes on their inputs before generating a cell rate change event on their output. Hence the generation of spurious multiple simultaneous cell rate change events by buffers in which cells are queued is eliminated.

The speedup achievable through the correct processing, by the event list, of simultaneous events is so great that it can be argued that any

simulation language or tool that does not properly account for them is not suitable for cell rate modelling. This issue is of particular importance given the increasing application of cell rate modelling to ATM network simulation.

8. Discussion

This chapter reviews the research presented in this thesis and consolidates the principle threads of the work. The issues raised by the research are explored in greater depth.

Section 8.1 reviews the basic principles of ATM network simulation and reiterates its importance despite the increasing presence of real ATM networks. The discussion concentrates on new applications for simulation, in particular in the field of network management. New approaches are discussed that are aimed at making simulators more accessible to network designers in general and less of a tool that is only used by the simulation specialist. Following this, Section 8.2 discusses the main topic of this thesis, that of accelerating cell rate simulation. The two principle threads of parallel cell rate simulation and improved sequential cell rate simulation are drawn together and placed in the context of the wider requirements of ATM network simulator users and developers. Possible alternative schemes for parallel cell rate simulation are also discussed.

8.1 Simulation of ATM Networks

In this thesis, a study of the accelerated simulation of ATM networks has been presented. ATM has been identified, by ITU-T, as the target transfer mode solution for the B-ISDN [I.150]. It is designed to carry all of the anticipated broadband services while maintaining efficient use of the available network resources. The principle features of ATM were briefly described in Chapter 2.

Chapter 3 outlined the motivation for the simulation of ATM networks. ATM simulation is required in the arenas of network hardware

development and network management. The application of simulation to the study of network management problems received particular attention.

An argument that is commonly presented against the inclusion of simulation work in contemporary ATM research projects is that real ATM networks are currently being introduced, so why not use them for experimentation purposes? The basis of the argument against this viewpoint is two-fold:

- the relative cost of real ATM networks
- the relative inflexibility of real ATM networks

ATM hardware is currently expensive in comparison with other, more established networking technologies. Furthermore, the installation and configuration of a large network is a complex task involving the employment of specialist engineers, even for relatively small projects. Once a network is installed it must be maintained, and this further adds to the cost of network ownership. In a modern competitive telecommunications industry, such costs must be matched by revenue earning potential. Therefore, those networks that do exist are primarily commercial in nature and hence it is difficult for research workers to gain access to them. Where access is possible, the range of experiments will be limited in order to minimise any disruption to the revenue earning function of the network. Researchers are often interested in network behaviour under conditions of very heavy load. However, such conditions may be detrimental to the QoS of existing network users. Other scenarios, such as the study of the network under fault conditions may also severely degrade the QoS experienced by these network customers. Clearly the use of a simulated network, as opposed to a real network, means that extreme network scenarios can be studied without affecting existing network users.

The second argument in favour of simulation concerns flexibility. This covers a range of issues that impact on the details of the networking scenarios that can be studied. These issues are:

- scaling
- network functionality
- measurement
- portability

Scaling relates to the maximum size of network (defined by the number of nodes and links, the number of traffic sources or users, and the pattern of demand placed on the network resources by those traffic sources) that can be studied. Existing research networks, such as the Exploit Testbed [SNH96], are small and often use a high proportion of artificial traffic sources. However, simulators enable much larger networks with many tens of switching nodes and tens of thousands of simultaneous calls to be studied. Such simulations can provide realistic scenarios for, in particular, the study of network management and resource allocation issues.

Flexibility in network functionality relates to the subset of ATM functions that are implemented in the network. This raises some important issues with respect to the current state of the technology that is implemented in real networks in comparison with that of interest to research projects. Simulators enable advanced network functions and algorithms to be modelled, many of which are far from reaching realisation in commercially available network hardware (for example, advanced connection admission control algorithms [ARM96.1][Ram97]).

Network functionality also relates to the control that experimenters have over the configuration of the network. Many of the configuration parameters that experimenters may wish to have access to are simply not available in real networks, but can be readily accessible in simulations.

The range of statistics that can be measured is of great importance. In many real networks statistics such as cell delay variation require complex and expensive instrumentation to measure. This is not required when simulation is employed and many such statistics can be generated with ease.

Finally, portability is an important factor. Real networks cannot be moved easily and, if remote access facilities are not available, may require experimenters to travel long distances in order to perform studies. Simulators, in particular those based around software that runs on relatively inexpensive computers, are highly portable and enable the experimenter to have a 'network model on their desktop'.

Hence it can be seen that there remain a wide variety of applications for simulation despite the inevitable introduction of real ATM networks.

Following the discussion of the motivation behind simulation, Chapter 3 introduced the principles of simulation and identified discrete event simulation as being the most appropriate technique for ATM simulation. This is because discrete event simulation can efficiently manage irregular event arrival times because the simulation clock is only advanced when an event is processed. Continuous time simulators advance the simulation clock in a series of fixed time increments, irrespective of the time of events, and hence processor time is wasted counting time intervals if there are no events to process. It is significant to note that the timestepping scheme is in many respects a hybrid of discrete and continuous time simulation. This is because, although the local time within each timestep at each model is advanced on a next-event basis, the global time (and the starting time of the next timestep) is advanced in a series of fixed time increments. Therefore, when there are many events to process per timestep, timestepping can be as efficient as pure discrete event simulation. However, when there are few events to process per timestep it

can be as inefficient as continuous time simulation. This is reflected in the results presented in Chapter 6.

The issue of simulation languages and tools forms another major topic of Chapter 3. The main thread of the research presented in this thesis considers the optimisation of event list management schemes for cell rate simulation, whether the simulators run on a parallel or a sequential computing platform. It is therefore of direct relevance to improving the performance of the simulation kernel of high level simulation tools (such as OPNET [OPN96]) when cell rate modelling is employed. Such simulation tools are becoming increasing popular, for they enable the simulator user to concentrate on the modelling of the network rather than on the details of the implementation of the simulator. Indeed, when 'hand-crafted' simulators are built much of the development effort is expended in the simulation kernel rather than the modelling. However, general purpose simulation tools tend to produce code that is less efficient than that of hand-crafted simulators. This is because hand-crafted simulators are often optimised for a specific simulation problem. General purpose tools, on the other hand, are designed to simulate a wide range of problems and hence are not designed to take advantage of the potential for optimisation inherent in any particular simulation problem. Optimising the performance of a simulation tool's kernel will contribute towards reducing the potential performance deficit by reducing the proportion of computational load attributable to the event list algorithm. Therefore, the efficiency of the models themselves will have a greater impact on the performance of the simulator. The results of the work presented in Chapter 7 raise a number of important issues relating to the management of events in general purpose simulation tools. These are discussed below.

Following the review of simulation languages and tools, Chapter 3 outlined cell level simulation of ATM networks. This is the traditional form of simulation. In cell level simulation, the propagation of each individual cell through the network is modelled. In a discrete event

simulator each cell is represented by an event, and in order to obtain statistically significant results when measuring rare events, such as cell losses from a queue, many billions of cell arrivals at that queue must be simulated. Therefore, cell level simulation results in a very high event processing load giving rise to very long simulation times (often in the region of many hundreds of hours for the measurement of low cell loss ratios at a queue). Such lengthy run times are unacceptable for many applications. This is particularly true for applications that require the simulator to interact with a network management system such as a TMN. In such scenarios, the simulator and the TMN must share a common perception of time. In the ICM project, this problem was solved in the following manner [Bocci95.2]. A Q-Adapter Function (QAF) was used to interface the simulator to the TMN. The current simulation time was provided to the QAF by the simulator, and this was then broadcast by the QAF to the TMN system. TMN applications were therefore able to adjust their perception of time to the simulator clock.

Whilst the ICM approach enabled TMN applications to successfully synchronise themselves with the simulator, it did require some customisation of the TMN platform software, OSIMIS [ICM95]. In other systems this may not be possible, and hence it is necessary for the simulation time to advance in approximately real time.

Techniques that reduce the run time of simulations are known as *accelerated simulation techniques*. Chapter 4 considered two of these:

- cell rate modelling
- parallel simulation

Cell rate modelling relates to the way in which ATM traffic flows are represented within the simulator. It relies on the reduction of the event processing load by using each event to represent a change in the rate of flow of ATM cells, rather than an individual cell. Because the discrete

nature of the cells is not modelled, the simultaneous arrival of cells at a queue is neglected. Therefore cell scale queueing cannot be modelled. However, it is possible to model bust scale queueing due to the input cell rate of a queue exceeding the rate at which the cells are served. The inability to model cell scale queueing results in slightly lower statistics for cell loss ratio when compared with a cell level simulation of the same network. However, in many cases such slight reductions in accuracy do not present a problem. For example, in the case of the ICM simulator the requirement on the accuracy of the statistics generated was ±10% [ICM93]. The validation of cell rate modelling presented in [Pitts93] demonstrates that a considerably better accuracy is possible while achieving significant speedup in comparison with cell level modelling.

Parallel simulation is an implementation technique that attempts to provide speedup by exploiting the inherent parallelism in a network simulation. Section 4.3 reviewed a number of basic criteria that must be satisfied in order for this to be effective. These include:

- communication between processors must be minimised
- processing must concentrate on local tasks
- workload must be evenly balanced across the network of processors

Many different decomposition and synchronisation schemes are described in the literature that all attempt to exploit the inherent parallelism in the network model. Whilst parallel simulation has shown some promise in terms of offering speedup over sequential simulators, many schemes have been implemented with only limited success because there is often a conflict between the above criteria. Where these problems are overcome, significant speedup can be achieved. In particular, the speedup is dependent on achieving an appropriate mapping between the distribution of processor load, the pattern of processor intercommunication, and the capabilities of the underlying computer hardware. In terms of ATM

simulation, this means that there must be a close match between the characteristics of the network simulation model and the computer that the simulator is to run on.

This section has discussed the existing state of the art in the accelerated simulation of ATM networks. However, the purpose of the research presented in this thesis has been to take existing accelerated simulation techniques and to investigate both the possibility of combining them to give further speedup, and to study the implications of this work for sequential cell rate simulation kernel design. This thesis has described new studies in both of these areas, the results of which are discussed below.

8.2 Accelerating Cell Rate Simulation

Two methods for accelerating cell rate simulation are considered:

- parallel cell rate simulation
- advanced event list structures that are optimised for sequential cell rate simulation

8.2.1 Parallel Cell Rate Simulation

Parallel processing techniques are intuitively attractive for the simulation of telecommunications networks. This is because there are many concurrent activities that occur in spatially distinct regions of a network. However, it is important to note than in simulation it is not the actual network itself that is simulated, but rather a somewhat abstract model of its behaviour. This statement is derived from the process by which a simulation is developed and from the definition of simulation. When a simulation of a network is developed, the first task is to develop a model of the architecture and operation of the network. This model can take the

form of a series of architectural diagrams, or it can be a detailed mathematical model of the network, or it could be a representation of the network using some formal description language. A good network model will only represent those aspects of the network that are of specific interest to the developer in order to minimise unnecessary computational overhead in the final simulator. Finally the model is implemented, usually by coding it in software.

An example of the structured approach to simulator development is that taken by the ARMAN project [ARM96.1]. Here, a simulation is developed by first gathering a number of requirements as to the exact features of the network that are to be studied. From this, a network model in the form of a formal specification is produced. This is the ITU-T Specification and Description Language (SDL) [Z.100][Belina89]. The SDL specification can then be implemented using the OPNET simulation tool [OPN96]. Hence it can be seen that a simulation is an implementation of the network model and not of the network itself.

The main requirement that must be satisfied for a parallel simulator to achieve significant speedup over its sequential equivalent is for there to be some exploitable parallelism in the network model and not in the network itself. It has been demonstrated that parallel simulation can achieve speedup when applied to cell level modelling of ATM. However, cell rate modelling uses a significantly different traffic model. Therefore, the applicability of parallel simulation to this modelling technique must be examined before complete cell rate parallel simulators are built. In this thesis, the timestepping approach was taken as an example time synchronisation scheme to enable the behaviour of cell rate ATM models in a parallel environment to be studied.

8.2.1.1 Timestepping

The timestepping approach was chosen because:

- an example simulator using this technique already existed
- this simulator was a product of a major research project in which the author participated
- timestepping can operate in both a parallel and a sequential computing environment, and therefore its effect on the speed and accuracy of the simulator could be studied without the need for complex parallel computing hardware.

The purpose of the study described in this thesis was to ascertain if:

- there is exploitable parallelism in a cell rate ATM model
- timestepping is able to exploit that parallelism and, if not, modify it so that it is effective
- the effect of timestepping on the accuracy of the simulator

Timestepping will only be effective if the synchronisation scheme does not represent a significant overhead on the simulator.

The results presented in Chapters 5 and 6 demonstrate that there is a relationship between the overhead of the timestepping scheme and the accuracy of the results obtained. When the timestep size is greater than one link delay, there is an error introduced into the delivery times of events that propagate along that link. In cell rate modelling this causes a quantisation of the length of bursts resulting in significant errors in cell loss measurements. These errors increase for larger values of timestep and for increased numbers of traffic sources multiplexed through the buffer in which the measurements are made. However, it is desirable for the timestep to be as large as possible because of the nature of event occurrence in cell rate simulation models. Cell rate modelling accelerates

the rate of simulated time advancement when compared with cell level modelling because it dramatically reduces the number of events that must be processed. Therefore, in order to reduce the amount of processor time wasted processing timesteps in which no events occur (and hence no useful work is done by the simulator), the timestep must be large enough so that the time spent processing events significantly outweighs that spent processing empty timesteps. The results presented in this thesis concur with this and demonstrate that, for the network models studied:

- for low event densities, processor usage is dominated by processing 'empty' timesteps
- timestepping is inefficient unless there are very large numbers of traffic sources

Pitts [Pitts93] suggests that cell rate modelling may be ideally suited to parallel simulation because the increased floating point arithmetic required in queueing models (in comparison with cell level modelling) means that processing is concentrated on local tasks rather than those that may require communication between processors. However, this is highly dependent on the exact model implementation. The complexity of queueing models used in the study described in this thesis was deliberately minimised in order to amplify the effect of the efficiency of the event management and time synchronisation schemes used in the simulation platform.

The basic problem when attempting to parallelise cell rate modelling is that there are very few events when compared with cell level modelling, and that the distribution of those events across the network model is less even. In cell rate modelling, the number of events required to describe a connection is not dependent on the cell rate of that connection, but rather on the nature of the variation in the cell rate during the connection. For example, constant bit rate connections (such as simple voice services) will require only two cell rate change events (at connection set-up and at

connection release) to represent the complete connection at the burst level. However, variable bit rate connections may require many hundreds of cell rate change events to model their duration. For such a variation in inter-event times, a purely event driven synchronisation scheme is most efficient. However, timestepping is a hybrid between an event driven scheme (the local time at each model advances on an event driven basis between timesteps), and a time driven scheme (the global time and local time at a model are advanced according to the start time of the next timestep when there are no more events to process in the current timestep,). Timestepping is therefore inefficient when the time driven characteristics dominate (when there are few events to process).

In conclusion, the study of timestepping presented in this thesis suggests that timestepping is unable to efficiently exploit any parallelism in the cell rate model. Indeed, the problems highlighted here are typical of those encountered when attempting to parallelise other ATM modelling techniques using synchronous time synchronisation schemes. However, despite the negative conclusion of the timestepping study, it has highlighted two points that have important implications for the design of sequential cell rate simulators:

- the exploitation of a spatially decomposed event list is beneficial
- the proper management of the simultaneous events that are generated during periods of burst scale queueing is vital for efficient cell rate simulation

The implications of these for sequential cell rate simulators are discussed below.

8.2.1.2 Alternative Parallel Cell Rate Simulation Schemes

Timestepping attempts to exploit spatial parallelism in the cell rate network model. However, other authors have described the application of a time decomposition scheme to cell rate parallel simulation [Nikolai93].

These studies only describe small network scenarios. However, the ability of the model to scale to realistically large networks is important. Furthermore, time decomposition is not appropriate if dynamic interaction with network management systems is required. Optimistic time synchronisation schemes could also be used for parallel cell rate simulation. However, once again the requirement for dynamic interaction with a management system would limit the application of such schemes. For example, any measurement errors that occur during periods of speculative execution must be smaller than the tolerance acceptable to the management system. Furthermore, is questionable whether a TMN system could cope with the variation in simulated time across the network or the changes in local simulated time due to rollback events. These would be large compared with those experienced in cell level parallel simulators because the time between events in a cell rate simulation is much greater. The ability to model overlaid signalling networks and systems would also be limited because they generally rely on the ability to model each individual signalling message.

Despite the above mentioned problems, there are some alternative areas in which parallelism could be exploited in the cell rate model. For example, a network management system's perception of a network is often of the logical network rather than the physical network. i.e. it maintains a view of the structure of the mesh of VPCs and VCCs and the routeing of cells along them, rather than the physical hardware of the network. Therefore, it is appropriate to model the logical network rather than the physical network. In such a model, VCCs only interact when they pass through a buffer in which cells are queued. Therefore, during periods of low utilisation, VCCs can be simulated concurrently. Clearly, however, the simulation time advancement must be synchronised between the VCCs.

During periods of burst scale queueing, cell rate changes on a single VCC will cause rate changes on all other VCCs sharing that queue. These will propagate out from the queue across the network as wave front of cell rate

changes, each one of which is independent of the others. Each cell rate change forming a part of that wave of cell rate changes is independent and can therefore be processed concurrently. These ideas are regarded as for further study and are therefore discussed in the Further Work section of this thesis.

8.2.2 Optimised Sequential Cell Rate Simulation

Sequential simulators are attractive because the computer hardware is relatively inexpensive in comparison with parallel hardware. Furthermore, the performance of a sequential simulator is less dependant on achieving an optimal mapping between the underlying hardware and the simulation scheme. Furthermore, experience has shown that sequential cell rate ATM simulators that employ efficient event management schemes are capable of simulating very large ATM networks (tens of thousands of simultaneous connections) at rates that exceed real time [ARM96.2].

The literature review presented in Chapter 7 reveals a wide range of advanced event list algorithms in existence in discrete event simulators. These are aimed at maximising the event list performance by minimising the time taken to insert an event into the list, or minimising the time taken to identify and extract the next scheduled event. Many of these algorithms are based on a binary tree structure . However, whilst detailed analyses of the performance of these algorithms is presented with respect to a generic 'hold model', none of them appear to have been studied in the context of cell rate ATM modelling.

The Linksim simulator is reviewed because it contains an event list that is especially tailored to cell rate modelling. It takes advantage of the following two optimisations suggested by the timestepping study of Chapter 5 of this thesis:

- a spatially decomposed event list

- correct handling of simultaneous events

The new work using SPROG demonstrated that, even in isolation, the spatial decomposition of the event list in a sequential simulator can give speedup over a simple linear event list (although not to the extent suggested by the timestepping study because of the overhead of maintaining the list of places). However, the most significant optimisation is the correct handling of multiple simultaneous events. These are always generated by a cell rate queue model during periods of cell queueing, and therefore it is vital that these are processed as a single group in order to prevent multiple spurious knock-on events at a subsequent downstream node where queueing is also occurring. The optimised space-time event list implemented in SPROG is an improvement over the Linksim scheme because it enables whole groups of simultaneous events to be delivered to node models in a single invocation. Furthermore, models that know that they will generate a set of simultaneous events of their output are able to submit that whole group to the event list in a single invocation of the SendEvent function. However, this structure does have the disadvantage that some of the event list management is now carried out by the models rather than the simulation services library. This problem can be rectified by using the property of inheritance that is inherent in the object oriented structure of SPROG. Functionality can be added to the simulation services library that includes a function that builds a list of simultaneous events that is then inserted in the event list in a single call to SendEvent.

In the ARMAN project, outside of the work described in this thesis, the problem of multiple simultaneous cell rate changes (that are not correctly handled by OPNET) was solved, by the author, using a concept known as *VPC bursts* [ARM96.1]. In this, rather than an event on the output of a queue representing a cell rate change on a single VCC, an event represents a cell rate change on a VPC as a whole. Therefore, one event on a VPC can be used to represent simultaneous cell rate changes on all the VCCs on that VPC.

In conclusion, it is clear that whilst spatial decomposition of the event list structure can improve the efficiency of the event list, the most significant gain is through the correct handling of simultaneous events. Correct handling of simultaneous events eliminates the generation of spurious cell rate change events in cascaded cell rate queues. Indeed, the results suggest that any simulation tool that is unable to deliver multiple simultaneous events to models as a group is unsuitable for cell rate ATM network simulation.

8.3 Further Work

The research presented in this thesis has concentrated on event list performance in a research simulator when running hypothetical network scenarios. The complexity of the network models was deliberately minimised in order to exemplify the effect of the efficiency of the event list algorithm on the overall performance of the simulator. In order to fully understand the factors affecting cell rate ATM simulator performance, it is useful to also study the effects of the design of the models themselves. In particular, the implementation of a full cell rate queue model, such as that of LINKSIM (rather than the simple 'zero depth' model implemented in SPROG), is required. The simulator execution could then be profiled when modelling real networking scenarios. Through this, efficiencies could be made in the design of the models, in addition to those already made to the event list algorithm.

The study of concurrent cell rate simulation concentrated on a spatial decomposition of the cell rate ATM network model and showed that this is unable to exploit sufficient parallelism when a timestepping time synchronisation scheme is used. It is clear that alternative parallel schemes must be considered. Further work is proposed that investigates alternative approaches to parallelising the cell rate model. For example, the possibility of basing the decomposition paradigm of the current state

of the queues in the network (as discussed in Chapter 8) rather than using a spatial decomposition. Furthermore, the application of optimistic schemes (such as Time Warp) should be investigated.

9. Conclusions

Many applications remain for ATM network simulation despite the introduction of real networks; the reasons for this were explained in detail earlier in the thesis. There therefore remains a strong demand for high speed, user friendly, workstation based simulators for both hardware design and, increasingly, network management studies. Such simulators are also useful, not only to researchers, but also to network managers for network planning and administration.

Traditional cell level simulation of ATM networks is computationally intensive, the result being long times for simulation runs. However, whilst a number of techniques have been recorded in the literature for reducing the time of simulation runs, there is still a need for further acceleration of the simulation speed. In particular, simulators that run in real time are required for the development and testing of new network management systems.

The combination of the existing accelerated simulation techniques of cell rate and parallel simulation is one possible way of improving simulator speeds. The timestepping approach, presented here as a synchronisation scheme for parallel cell rate simulation, is inefficient if event densities are low because CPU time is wasted processing empty timesteps. However, when the simulator is modelling highly congested networks with very high traffic loads, the event density in a cell rate simulator is high enough to minimise the number of wasted timesteps. In such situations, timestepping is a practical time synchronisation scheme in parallel cell rate ATM network simulators.

The study of timestepping performance has also enabled a number of other conclusions to be drawn that can be used to improve the performance of simulators running on sequential computer platforms. The

first of these conclusions is that spatial decomposition of the event list structure can reduce the time required to insert an event in the event list. This is because, provided the location of the event is known, then the local event lists that represent the places that an event is scheduled for will always be shorter than a single linear global event list.

Secondly, it is clear that cell rate modelling requires that the simulation platform is able to deliver simultaneous events to a model as a group so that they can be processed together. This is important because, whenever there is a cell rate change on the input to a buffer model in which cells are queued, multiple simultaneous cell rate change events will be generated on the output of the buffer. These must be processed together as a group by any model that receives them on it's input. Simulators that can not guarantee this are not suitable for cell rate modelling.

Abbreviations

AAL	ATM Adaptation Layer
ABR	Available Bit Rate
ABT	ATM Block Transfer
ARMAN	ATM Resource Management
ATM	Asynchronous Transfer Mode
B-ISDN	Broadband ISDN
CAC	Connection Admission Control
CBR	Constant Bit Rate
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CEC	Commission for the European Communities
CFS	Common Functional Specification
CLR	Cell Loss Ratio
CODEC	Coder/Decoder
CPU	Central Processing Unit
DTI	Department for Trade and Industry
EPSRC	Engineering and Physical Sciences Research Council
FIFO	First-In-First-Out
ICM	Integrated Communications Management
ISDN	Integrated Services Digital Network
ITU-T	International Telecommunications Union Telecommunications Standardization Sector (Formerly CCITT)
MADS	Multi-Purpose Aid for Distributed Simulation

NE	Network Element
NP	Network Performance
NPC	Network Parameter Control
NT	Network Termination
QoS	Quality of Service
RACE	Research into Advanced Communications in Europe
TMN	Telecommunications Management Network
UPC	Usage Parameter Control
VBR	Variable Bit Rate
VC	Virtual Channel
VCC	Virtual Channel Connection
VCI	Virtual Channel Identifier
VP	Virtual Path
VPC	Virtual Path Connection
VPI	Virtual Path Identifier

Bibliography

Publications by the Author

[Bocci94] Bocci M, Pitts J M, Scharf E M; "Performance of Time Stepping Mechanism for Parallel Cell Rate Simulation Of ATM Networks"; 11th IEE UK Teletraffic Symposium; March 1994

[Bocci95.1] Bocci M, Pitts J M, Cuthbert L G; "Exploiting Concurrency Through Knowledge of Event Propagation in Cell Rate ATM Network Simulation"; 12th IEE UK Teletraffic Symposium; March 1995

[Bocci95.2] Bocci M, Scharf E M, Georgatsos P, Hansen M, Thomsen J, Swift J; "ATM Network Simulation Support for TMN Systems"; 3rd International Conference on Intelligence in Broadband Services and Networks; Heraklion, Crete, September 1995

[Swift94] Swift J, Bocci M, Chen J R, Pitts J, Scharf E M; "Application of ATM Network Simulation to TMN Studies"; 11th UK Teletraffic Symposium; March 1994

ITU-T Recommendations

[I.121] ITU Recommendation I.121; "Broadband Aspects of ISDN-BISDN"; July 1991

[I.150] ITU Recommendation I.150; "B-ISDN Asynchronous Transfer Mode Functional Characteristics"; November 1993

[I.350] ITU Recommendation I.350; "General Aspects of Quality of Service and Network Performance in Digital Networks, Including ISDN"; November 1993

[I.371] ITU Draft Recommendation I.371; "Traffic Control and Congestion Control in B-ISDN";SG13; Geneva, May 1996

[M.3010] CCITT Recommendation M.3010; "Principles for a Telecommunications Management Network"; December 1991

[Z.100] ITU-T Recommendation Z.100; "CCITT Specification and Description Language (SDL)"; Rev 1; ITU T; Geneva; June 1994

Other References

[Amd88] Amdahl G M; “Limits of Expectation”; International Journal of Supercomputer Applications; Vol. 2, No. 1, pp88-97; 1988

[Amm92] Ammar H H , Deng S U; ”Time Warp Simulation Using Time Scale Decomposition”; ACM Transactions on Modeling and Computer Simulation; Vol. 2, No. 2; April 1992

[ARM96.1] Project ARMAN; “Specification and Development of Enhancements to Simulation Tool”; trp0081x; Sept 1996

[ARM96.2] Project ARMAN; “System Performance Evaluation”; trp0131x; Dec 1996

[Belina89] Belina F, Hogrefe D; “The CCITT Specification and Description Language SDL”; Computer Networks and ISDN Systems 16; pp311-341; 1989

[CCITT89] “CCITT COM XVII-R 4-E”, Report of the BBTG Meeting, Jan/Feb 1989

[Chand89.1] Chandy K M, Sherman R; “The Conditional Event Approach to Distributed Simulation”; Proc. SCS Multiconf.. on Distributed Simulation, pp. 93-99, 1989

[Chand89.2] Chandy K M, Sherman R; “Space-Time and Simulation”; Proc. of SCS Multiconf. on Distributed Simulation; Vol. 21; March 1989

[Comf79] Comfort J C; “A Taxonomy and Analysis of Event Set Management Algorithms for Discrete Event Simulation”; Proc. IEEE Computer Society Annual Simulation Symposium; pp115-146; 14-16 March 1979

[Comf82] Comfort J C; “The Design of a Multi-Processor Based Simulation Computer - I”; in Annual. Simulation. Symposium, pp45-52, 1982

[Comf83] Comfort J C; “The Design of a Multi-Processor Based Simulation Computer - II”; in Annual. Simulation. Symposium, pp197-209, 1983

[Comf88] Comfort J C, Gopal R R; “Environment Partitioned Distributed Simulation with Transputers”; Proceedings SCS Multi-conf. on Distributed Simulation, pp103-108, 1988

[Cuth93] Cuthbert L G, Sapanel J C; “ATM - The Broadband Telecommunications Solution”; IEE; ISBN 0-85296-815-9; 1993

[Evans86] Evans J B; “Experiments with Trees for the Storage and Retrieval of Future Events”; Information Processing Letters; Vol 22, pp237-242; 1986

[Flynn66] Flynn M J; “Very High Speed Computing Systems”; Proceedings of the IEEE; Vol 54 No 12; December 1966

[Fost95] Foster I; “Designing and Building Parallel Programs”; Addison Wesley; 1995; ISBN 0-201-57594-9

[Fujim95] Fujimoto R M, Nikolaidis I, Cooper C A; “Parallel Simulation of Statistical Multiplexers”; Discrete Event Dynamic Systems: Theory and Applications, 5; pp115-140; July 1995

[Gaf88] Gafni A; “Rollback Mechanisms for Optimistic Distributed Simulation Systems”; Proc. SCS Multiconf. on Dist. Simulation, pp. 61-67, 1988

[H100] CEC RACE Common Functional Specification H110; “Network Management Terminology and Definitions”; Issue C; December 1992

[Henn90] Hennessy J L, Patterson D A; “Computer Architecture - A Quantitative Approach”; Morgan Kaufmann Publishers Inc; ISBN 1-55860-069-8; 1990

[Henr83] Henriksen J O; “Event List Management - A Tutorial”; In Proc. Of 1983 IEEE Winter Simulation Conference; Washington DC; pp543-551; 1983

[Hind92] Hind A; “Parallelisation of a Circuit-Switched Telecommunications Network Simulator”; 9th IEE UK Teletraffic Symposium; April 1992

[Hind94] Hind H; “Parallel Simulation Techniques for Telecommunication Network Modelling”; PhD Thesis, University of Durham; January 1994

[ICM92.1] CEC RACE R2059 ICM Deliverable 2; “Initial TMN Architecture, Functions and Design Approaches”; R2059/CRA/ATG/DS/R/003/b1; December 1992

[ICM92.2] CEC RACE R2059 ICM Deliverable 3; “Selection of Networks, Network Interfaces and Definition of Testbed Laboratory”; R2059/VTT/TEL/DS/R/004/b1; December 1992

[ICM93] CEC RACE ICM WP5; “ICM Simulator Requirements and Specifications”; Version 3; 22 February 1993

[ICM95] CEC RACE ICM Deliverable 18; “Detailed ICM TMN Testbed Component Specifications and Descriptions”; R2059/CRA/ATG/DS/P/018/b1; December 1995

[Jeff85] Jefferson D; “Virtual Time” in ACM Trans. Programming Languages and Systems; Vol 7 No. 3, pp. 404-425, July 1985

[Jones86] Jones D W, Henriksen J O, Pegden C D, Sargent R G, O’Keefe R M, Unger B W; “Implementations of Time - Panel Discussion”; Proc 1986 Winter Simulation Conference; pp409-416; 1986

[Kings89] Kingston J H; “Analysis of Tree Algorithms for the Simulation Event List”; Acta Informatica; Vol 22, pp15-33; Springer-Verlang 1985

[Law91] Law A M, Kelton W D; "Simulation Modelling and Analysis"; McGraw-Hill International; ISBN 0-07-100803-9; 2nd Edition 1991

[Lin89] Lin Y, Lazowska E D; "Conservative Parallel Simulation for Systems with no Lookahead Prediction"; Tech. Rep. 89-07-07, University Of Washington; July 1989

[Lin90] Lin Y, Lazowska E D; "Exploiting Lookahead in Parallel Simulation"; IEEE Trans. on Parallel and Distributed Systems, Vol 1 No. 4, pp 457-469; Oct 1990

[Lit94] Little M C, McCue D L; "Construction and Use of a Simulation Package in C++"; C User's Journal; Vol12, No 3; March 1994

[Lub88] Lubachevsky B D; "Bounded Lag Distributed Discrete Event Simulation"; Proceedings of SCS MultiConf. on Dist. Simulation, pp. 183-191, 1988

[McCor81] McCormack W M, Sargent R G; "Analysis of Future Event Set Algorithms for Discrete Event Simulation"; Communications of the ACM; Vol 24, No 12, pp801-812; Dec 1981

[MIME91] MIME, RACE project R1084; "Set of Specifications for MESH Systems for ATM Networks"; Deliverable 10; July 1991

[Misra86] Misra J; "Distributed Discrete-Event Simulation"; Comp. Surveys; Vol. 18, No. 1, pp39-65; March 1986

[NEME92] CEC RACE R1005 NEMESYS Deliverable 11; "Conclusions of Project NEMESYS"; 05/KT/AS/DS/B/028/b1; December 1992

[Niko93] Nikolopoulos S D, MacLeod R; "An Experimental Analysis of Event Set Algorithms for Discrete Event Simulation"; Microprocessing and Microprogramming; Vol 36, pp71-81; 1993

[Nicol88] Nicol D M; "High Performance Parallelized Discrete-Event Simulation of Stochastic Queueing Networks"; Proceedings of SCS 1988 Winter Simulation Conference; pp306-314

[Nicol90] Nicol D M; "The Cost of Conservative Synchronisation in Parallel Discrete Event Simulation"; Technical Report, ICASE, Jan 1990

[Nikolai93] Nikolaidis I, Fujimoto, R, Cooper C A; "Parallel Simulation of High-Speed Network Multiplexers"; Proceedings of 32nd Conference on Decision and Control (IEEE); Dec 1993

[OPN96] Mil3 Inc; Washington DC; OPNET Online Documentation; Version 2.5B; 1996

[Phillips91] Phillips, C I; Cuthbert, L G; 'Concurrent Discrete Event Driven Simulation Tools'; IEEE Journal on Selected Areas in Communications, Vol. 9 No. 3, April 1991.

[Phillips92] Phillips, C I; "Concurrent Discrete Event-Driven Simulation Techniques for Telecommunications Networks"; PhD Thesis; May 1992

[Pitts90] Pitts J M, Sun Z; “Burst Level Teletraffic Modelling and Simulation of Broadband Multi-Service Networks”; 7th UK Teletraffic Symposium; April 1990

[Pitts91] Pitts, J M; Sun, Z; Scharf, E; 'A Comparison of burst-level and cell-level approaches to the simulation of ATM networks'; 13th International Teletraffic Congress: Discussion Circles; 19/26 June 1991

[Pitts92.1] Pitts J M, Schormans J A; “Renewel Theory Validation of Burst Level Technique for ATM Simulation”; Electronics Letters; 16th Jan 1992 Vol 28 No 2 pp106-107

[Pitts92.2] Pitts J M, Schormans J A, Scharf E M; “Burst Level Simulation: A comparison with Cell Level Simulation and Queueing Analysis”; 9th IEE UK Teletraffic Symposium; April 1992

[Pitts93] Pitts J M; “Cell-Rate Simulation Modelling of Asynchronous Transfer Mode Telecommunications Networks”; PhD Thesis; July 1993

[Pryk91] de Prycker M; “Asynchronous Transfer Mode: Solution for Broadband ISDN”; Ellis Horwood; ISBN 0-13-053513-3; 1991

[Ram97] Ramalho M F; “Application of an Automatically Designed Fuzzy Logic Decision Support System to CAC in ATM Networks”; PhD Thesis; University of London; January 1997

[Raw92] Rawling M, Francis R, Abramson D; “Potential Performance of Parallel Conservative Simulation of VLSI Circuits and Systems”; Proc. Of 25th Annual Simulation Symposium, Orlando, Florida; April 1992

[Rhei91] Rheiher P, Bellenot S, Jefferson D; “Temporal Decomposition of Simulations Under the Time Warp Operating Systems”; Proceedings of SCS Multiconference on Parallel and Distributed Simulation; Vol. 23; Jan 1991

[Rich89] Richter R, Walrand J C; “Distributed Simulation of Discrete Event Systems”; Proceedings of the IEEE, Vol 77, No 1, January 1989

[Schor94] Schormanns J, Phillips C, Pitts J, Scharf E, Manthorpe S, Smith R; “MICROSIM II Users Guide and Technical Reference Manual”; Queen Mary & Westfield College; Version 2.08; May 1994

[SNH96] Swiss National Host; “Swiss National Host Description”; v2.0, July 1995

[Wich82] Wichmann B A, Hill I D; “An Efficient and Portable Pseudo-Random Number Generator”; Applied Statistics, Algorithm AS 183; pp188-190; 1982

[Wiener88] Wiener R S, Pinson L J; “An Introduction to Object-Oriented Programming and C++”; Addison-Wesley Publishing Company Inc.; ISBN 0-201-15413-7; 1988