Working Paper Series
ISSN 1170-487X

High Performance Simulation for ATM Network Development

by: John Cleary, Murray Pearson, Ian Graham, and Brian Unger

Working Paper 96/11

June 1996

Brian Unger and Ian Graham
Department of Computer Science
The University of Waikato
Private Bag 3105
Hamilton, New Zealand

High Performance Simulation for ATM Network Development

Final Study Report

for

New Zealand TeleCom

John Cleary

Murray Pearson

Ian Graham

Brian Unger

Department of Computer Science

University of Waikato

Abstract

Techniques for measuring and modeling ATM traffic are reviewed. The requirements for cell level ATM network modeling and simulation are then outlined followed by a description of an ATM traffic and network (ATM-TN) simulator. This ATM-TN simulator is built upon parallel simulation mechanisms to achieve the high performance needed to execute the huge number of cell events required for a realistic network scenario. Results for several simulation experiments are reported using this high performance simulator including scenarios for the Wnet and OPERA networks. Finally, a preliminary evaluation of the application of high performance simulation to the design and analysis of ATM network performance is provided.

June 1996

Table of Contents

List of Figures

List of Tables

1. Introduction

This report summarizes research that has been aimed at the measurement and characterization of ATM multimedia traffic, the modeling and simulation of ATM networks, and the design of high performance simulation engines that are capable of supporting very computation intensive ATM network simulations. The results of work performed at Waikato as part of this study for TeleCom is included along with closely related work that has been completed elsewhere.

First, the requirements of a cell level ATM network simulator are outlined in section 2.0. This involves traffic characterisation, ATM switch modeling, and network level modeling. Next, the architecture and major components of an ATM traffic and network (called the ATM-TN) simulator are presented in 3.0. A high performance simulation environment on which the ATM-TN simulator is based is then described in section 4.0. In section 5.0, simulation results for several ATM network scenarios, including one for OPERA, are outlined. Section 6.0 presents a preliminary evaluation of high performance simulation as a methodology for the analysis and design of ATM networks. Conclusions are outlined in section 7.0

The body of this report represents a brief overview of the work accomplished. A detailed review of techniques for the measurement and modeling of ATM cell level traffic is presented in a companion paper [Pearson et. al. 96]. Measurement results for a specific multimedia scenario are included in Appendix A. Preliminary work on a next generation high performance simulation engine is contained in Appendix B.

2. ATM Network Simulation Requirements

The general requirements of an ATM network simulator are to support: network performance analysis under varying traffic types and loads, network capacity planning, traffic aggregation studies, and ATM network protocol research. This spans a wide range of applications from production use by ATM network planners to ATM switch, network and protocol design by researchers.

These general requirements imply the need for cell level models of arbitrary network topologies with tens to hundreds of switches; models of multiple switch architectures and the characterisation of a range of ATM network traffic sources. The requirements of the traffic models, switch and network models, simulator run-time performance, and other simulator requirements are outlined in this report. These model components are all part of a modeling environment called the ATM-TN.

2.1 ATM Traffic Modeling

Three basic types of network traffic models are required in a general ATM network modeling environment: both compressed and uncompressed video traffic, an Internet traffic model of World Wide Web (WWW) browsing interactions, and aggregate ethernet local area network (LAN) traffic. Specifically, each of the ATM-TN traffic models should support:

- point to point traffic definitions for multiple traffic types and varying traffic rates;
- dynamic bandwidth allocation and deallocation;
- statistics for cell loss ratios (CLR), cell transfer delay (CTD) and cell delay variation (CDV); and
- higher level traffic protocols and statistics specific to traffic types, eg., TCP/IP packet level statistics such as error rates, packet sizes and delays.

All of the traffic models should capture the behavior of different types of user demands for communication services. Each type of traffic model will ultimately generate ATM cells as input to the network.

In the future, other traffic models in addition to the above will need to be incorporated. Modeling specific TCP/IP applications and LAN-ATM-LAN traffic including the (ATM Adaptation Layer) AAL functions are examples of anticipated future requirements.

2.2 ATM Switch & Network Modeling

The ATM-TN will model the protocols for ATM switching, call setup and release, and parts of the ATM Adaptation Layer (AAL). Other protocols such as policing and admission control will need to be modeled in the future. The latter will require the ATM-TN core models to be extensible.

The switch models must characterise the flow of ATM cells through the network as determined by the specific architecture of each switch type. The following outlines some of the elements that must be represented. Each type must be modeled using sub-components:

- a dimension (NxN) specifying the number of input ports and output ports on the switch;
- a switching fabric, defining the connections between input ports and output ports (eg., crossbar, shared bus, banyan, delta network, ...);
- a set of buffers and a buffering strategy that specifies how many buffers are available, and how they are configured and used (eg., shared vs. partitioned, input vs. output buffering); and
- routing tables, used to map cells from input ports to output ports. This mapping is done using virtual circuit and virtual path indices (VCIs and VPIs).

Links (or ports) must be modeled implicitly in the exchange of cells between switches (eg., capacity in bits/sec, propagation delay, and bit error rate) rather than explicitly as a separate model component. This approach will significantly reduce the number of simulation events.

The intent is to make it easy to "plug and play" to evaluate the performance implications of different ATM switch architectures, and later, of different switch call admission, policing, and traffic control mechanisms.

2.3 Simulator Performance

The applications of the ATM-TN will require simulations of between 10^9 and 10^{12} cells. In a straightforward characterisation of cell interactions through network switches this implies roughly five to ten times that number of simulation events.

Desirable simulation run-times are less than two hours with a maximum of 12 hours for overnight execution. These execution times are clearly not feasible for simulations of more than about 10^9 cells. Thus, extraordinary approaches will be required to achieve reasonable simulation run-times.

Initially it is not desirable to introduce approximations or hybrid analytic and simulation approaches. It is important that a high fidelity, accurate, model be developed first which can be used to validate simulator optimizations, and then approximate and hybrid approaches. Thus, despite the run-time requirements, an initial ATM-TN version is required that is capable of closely mimicking actual traffic and network behavior. Further, it is desirable that test and debug runs can be executed on ubiquitous PCs and Unix workstations.

In summary, the simulator efficiency will be crucial and care is needed to create very efficient sequential and parallel kernels.

2.4 Production versus Research Issues

A significant part of the ATM-TN simulator requirements are to support "production" ATM network sizing and planning by network planners. This requires a non-programmer interface that offers interactive data management and experiment control tools for large input and output files. The input files include an array of parameters to define specific switch and link characteristics, to define network configurations, and to specify experiment scenarios. Output files should be structured to enable the use of third party spreadsheet, statistical and graphical data analysis tools.

The other major goals of an ATM-TN simulator are to support research. The current ATM networking research issues of interest include: call admission control on entrance to the network, congestion control within the network, usage parameter control (UPC) within the network and switch service disciplines (scheduling policies). Other issues alluded to in the above sections include: multicast routing, switch architectures within network context and network management protocols.

Finally, there are research issues in modeling and simulation methodologies for this application. These include workload and traffic characterisation in terms of measurable parameters, the design of efficient simulation kernels, hybrid analytic and simulation approaches, and the parallel execution of ATM traffic and network models. The requirements of an ATM-TN include being able to address these simulation methodology issues, the open issues in ATM networking outlined above, as well as, the commercially important network planning issues.

3. An ATM Traffic and Network Simulator : ATM-TN

The ATM-TN simulator presented here was designed to meet all of the requirements outlined in section 2. The main design principles were to: (1) accurately mimic ATM network behavior at the cell level for specifiable traffic loads, (2) create a modular extensible architecture since requirements will evolve as new issues emerge, (3) achieve reasonable execution times for ATM networks that consist of hundreds of nodes and thousands of traffic sources. Since (1) implies an extremely computationally intense simulator and (2) implies a long lived simulator it is reasonable to expend substantial effort on (3). Further details on the ATM-TN can be found in Gburzynski (1995).

3.1 Support for Parallel Execution

A cell level simulator for a moderately large ATM network has substantial potential for parallel execution. There is a great deal of independent activity involved in a large number of individual streams of cells flowing through a large network. Our analysis of the run-time behavior of an ATM-TN and related network simulation problems [Unger et al. 94a, Unger & Xiao 94b, Unger & Cleary 93] suggests that optimistic synchronization schemes have potential.

Although a decade of research in optimistic synchronization methods for achieving speedup through parallel execution on multiprocessor platforms has still produced mixed results, it was decided that the emergence of moderately large shared memory multiprocessor systems would have much greater potential for speedup in an ATM-TN simulator [Unger et al. 93]. This led to the design and development of SimKit and WarpKit [Gomes et al. 95].

Simulations written in SimKit can be executed either sequentially or in parallel. The optimized sequential simulator (OSS) has been developed to support very fast, efficient, sequential execution. The second kernel, called WarpKit, supports parallel execution on shared memory multiprocessor platforms based on the Virtual Time (Time Warp) paradigm defined in [Jefferson 85 and Fujimoto 90b].

The WarpKit kernel design is based on research reported in [Fujimoto 90a, Baezner et al. 94 and Unger et al. 90]. The design of both WarpKit and SimKit are aimed at general purpose discrete event simulation problems. However, SimKit (see section 5) enables building custom mechanisms to support very efficient constructs that appear frequently in the implementation of network simulators, or specifically, in the ATM-TN.

3.2 Major Components & Interfaces

The general structure of the ATM-TN simulator is illustrated in Figure 1. The major components include: traffic models, switch models, an ATM Modeling Framework, SimKit, WarpKit, OSS and the TeleCom Modeling Framework (TMF). The TMF and ATM Modeling Framework are outlined below. The traffic and switch models are described in subsequent sections.

The TMF enables the TeleCom engineer to configure a specific simulation model and vary model component parameters. The user constructs a model by specifying network topology, switch architectures, the traffic sources and the links

connecting them. Network provisioning through Virtual Path assignments and Permanent Virtual Circuit allotments are possible. The input is saved in Unix files as per input data set specifications. Currently, testing the validity of input data, eg., that values are within reasonable ranges, is performed by the ATM Modeling Framework (ATM-MF) at run time during the instantiation of various model components.

Figure 1 ATM-TN Simulator Architecture

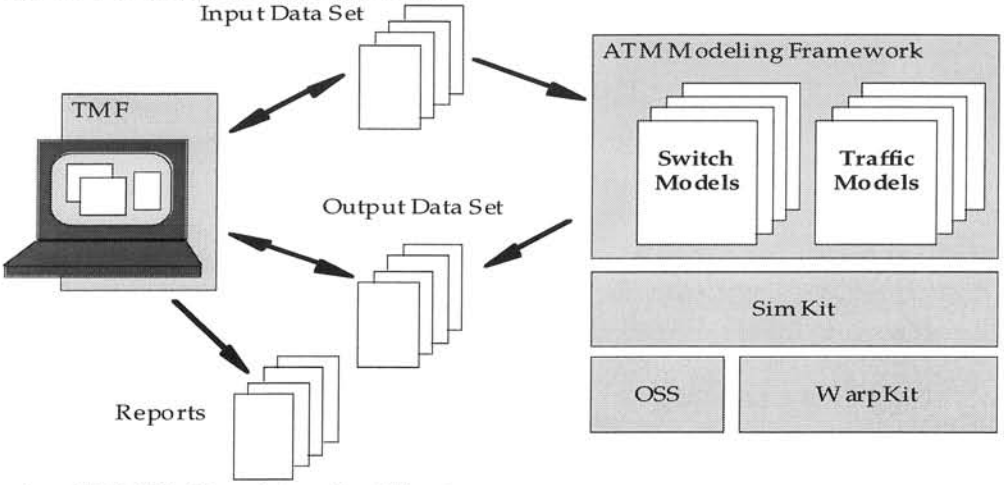


Figure 1 ATM-TN Simulator Architecture

Simulation execution mode, the simulation executive, ie., OSS or WarpKit, and the hardware platform for specific experiments are then selected through a menu driven run time control option in the TMF. The different types of reports to be generated from the simulation output files can also be directly selected through the TMF report generation menu option. Batch modes for simulation execution and report generation enable hands-free simulation execution requiring no subsequent interaction and monitoring during runs.

3.3 The ATM Modeling Framework

The ATM-MF provides support and startup functions for the various model components. These include initialization and control, model construction and termination, model component identification, external event notification, and support for input and output. The MF also supports the modification of traffic and switch model components and the addition of new model components. Permanent and Switched Virtual Circuit modeling are supported using Virtual Path and Virtual Circuit connections. Generic Call Admission Control (CAC), and other Traffic Control functions are modeled in a modular way, supporting extension and user customization.

This architecture supports use of the ATM-TN simulator with or without the TMF user interface. Third party data management tools can be used to create and manipulate input data sets and the output data sets.

This architecture also supports exploring performance improvements along both sequential and parallel tracks. Once validation of the ATM-TN is complete we expect to explore a number of optimizations of the sequential simulator including mechanisms for cell/event aggregation and hybrid switch models. A great deal of work aimed at improving the parallel execution performance of the ATM-TN model is also possible.

3.4 Traffic Models

Traffic sources are used to model any object that generates ATM cells that are to be carried by the network, ie., by a network of switches. An ATM source can be an end-user application, a LAN, or the output port of another ATM switch.

Several distinct components are used in the construction of sources:

- *Input traffic sources.* These are used to model individual end-user applications that generate data carried by the network including data sources (eg., FTP, Telnet, Mosaic, TCP/IP) and video sources (eg., video-conferencing, JPEG, MPEG). Input traffic can modeled directly at the cell level, or at higher levels, eg., burst level, packet level, message level, as appropriate for each source.

- *ATM Adaptation Layer* (AAL). This component is used to translate, where necessary, from input traffic source packets to ATM cells using the AAL specifications. This component is needed for converting packet or message level input traffic to cell-level models (eg., for modeling TCP/IP, LAN-ATM-LAN traffic, and Frame Relay.) This piece is not needed for cell-level input traffic models.

- *Access control mechanisms.* This component will be used to model access control (policing) mechanisms, such as the Leaky Bucket, that may be interposed between a raw input traffic source and the ATM network itself, thereby changing the traffic flow characteristics. A person using the simulator can choose to run simulations either with or without this mechanism.

Each traffic type, contains a call generator that can spawn multiple instances of the traffic model at the site where the traffic type is attached. For the MPEG video model, call arrivals, call durations and call destinations are specified by the simulation user. WWW model arrivals are Poisson. The durations and destinations are specified by the WWW session model.

Many instances of MPEG and WWW transaction models may exist at a site, whereas only a single Ethernet traffic model (representing aggregate LAN traffic) can exist at a single site. The destination for a LAN traffic model is specified by the user. Ethernet packet sizes are not modeled explicitly.

Currently, the traffic descriptors for each traffic model are being determined. These will include the resource requirement parameters PCR, SCR, MBS, and the service requirement portion CDVT. Traffic sources keep track of the number of cells, packets and higher level data units transmitted and their corresponding retransmission counts. Call level statistics include the call blocking probability, the call setup / release delays and call durations. Traffic sinks maintain a count of cells, packets and higher data units received as well as discarded. Delay sensitive traffic models also maintain a per-cell CDV measure. Frame-level statistics are maintained by video sinks, which includes frames received and average number of frames lost or late.

The three basic traffic models, MPEG, ethernet and WWW, are outlined below. A detailed description of the traffic models can be found in [Arlitt et al. 95a].

Further information on the WWW model is presented in [Arlitt & Williamson 95b] and on the ethernet model in [Chen et al. 95].

3.4.1 MPEG and JPEG Video Models

The MPEG/JPEG video traffic model is designed to characterise the unidirectional transmission of a single variable bit rate compressed video stream of data. This video data is both delay-sensitive and loss-sensitive; if cells are lost, image quality is degraded, and if cells arrive late, they are discarded, ie., the effect of delay can be the same as if they were lost.

The MPEG (motion pictures expert group) algorithm is a standardized method of compressing full motion video for storage as digital data. An MPEG video stream consists of a sequence of images, or frames, that are displayed one after the other at short periodic intervals. The standard defines three types of three types of compressed frames called I, P and B frames as follows:

- *I frames* are encoded using only the image data available within the current frame. Thus an I (intraframe) frame represents a complete image, they provide an absolute reference point for the other two image types in an MPEG sequence.
- *P frames* contain motion-compensated data predicted from the preceding I or P frame. P (predicted) frames take longer to encode than I frames, are faster to decode than I frames, and achieve higher compression than I frames.
- *B frames* contain motion-compensated data from both the previous and the next frame (I or P). B (bi-directional interpolative) frames take the longest time to encode but offer the greatest compression.

A JPEG stream consists of all I frames and thus provides compression on each frame as an independent unit of data. MPEG sequences consist of a pattern of I,P and B frames called a group of pictures (GOP). The GOP frame is specified at the start of encoding, eg., an IBBPBBPBBI sequence which is continually repeated.

The MPEG video traffic model in the ATM-TN simulates cell level ATM traffic generated by an MPEG video stream to a viewer. The model generates a given combination of I, P and B frames at a set frame rate. These random sequences are generated using the transform-expand-sample (TES) modeling methodology defined by [Melamed and Hill 95]. Here, TES is used to generate three autocorrelated sequences of frame sizes, one for each type of MPEG frame. These separate sequences are then interleaved according to the GOP to produce the simulated traffic. Thus, this MPEG/JPEG traffic model is a composite TES model.

3.4.2 An Aggregate Ethernet Model

The ethernet LAN model is designed to represent aggregate data packet traffic on existing local area networks such as a university campus LAN. One of the requirements of the ATM-TN is to characterise ATM connections between legacy LAN networks as part of the evolution to ATM. This kind of traffic is likely to form a significant fraction of background load for early configurations of ATM networks.

Recent research confirms that ethernet traffic exhibits a fractal, self-similar, behavior where there is no natural length for bursts. This means that Poisson models do a very poor job of representing ethernet traffic. A stochastic process is said to be self-similar with Hurst parameter H if the process is covariance stationary and aggregate versions of the same process have the same structure as the original process. Thus, bursts are visible in aggregated traffic on a wide range of time scales, eg., from milliseconds to minutes.

This self similar behavior has been observed both in traffic internal to an ethernet LAN, as well as, to traffic leaving a LAN. This behavior has been observed by researchers at Bellcore, and we have observed similar behavior in measurements taken at the University of Saskatchewan. The latter suggest self-similar behavior with a Hurst parameter close to 0.7. The ATM-TN aggregate ethernet traffic model is based on the TES methodology for generating random traffic sequences that have validated first and second order time series statistics, ie., a frequency histogram and autocorrelation function that matches actual LAN measurement data.

3.4.3 A World Wide Web Transaction Model

The ATM-TN WWW model characterizes the cell traffic generated during a Mosaic like browser session exploring WWW servers. A single Mosaic session can generate one or more Mosaic conversations. Multiple destination sites may be involved in a single session, one destination at a time. Each conversation may consist of one or more TCP connections. Each TCP connection represents a single VC connection requiring an independent connection setup. A W W W session can spawn multiple conversations, eg., from one to hundreds, and each conversation can spawn multiple connections, eg., one to ten connections.

The input parameters to a WWW session include: mean number of conversations per session (Geometric distribution), mean conversation gap time (Poisson model), conversation destination, mean number of connections per conversation (geometric), mean connection gap times (Poisson) and the amount of information exchanged during a connection. In the latter, bytes sent by the source are modeled using a log Normal distribution, bytes received from the destination are modeled using a log Erlang distribution, and the mean and standard deviations are input parameters.

A single WWW session may last minutes or hours representing a user searching for information across the Internet. Empirical measurements of Internet W W W traffic collected at the University of Saskatchewan were used to construct and parameterize this model. Traces were collected using the Unix tcpdump and thousands of TCP connections were observed. Four one day traces were collected which contained 5,829 conversations which formed 57% of the total network activity.

3.5 Switch Models

The ATM-TN switch models currently include three simple, and three multistage switch models. These models support point-to-point switched virtual channels (SVCs) and permanent virtual channels (PVCs). The SVC call setup and release procedures modeled closely approximate the UNI 3.0 specification

[ATM 93]. Virtual Paths (VPs) are also characterized and defined by input data as they tend to be long lived connections.

An ATM-TN network model has a number of components including: communication links, traffic source / sinks (TSSs), end nodes and switches. End nodes are a simplified type of switch that is used at the edges of an ATM network. The switch models provide:

- a simple traffic shaping scheme on a per VC basis,
- a simple call admission control (CAC) scheme at each switch,
- a user parameter control (UPC) scheme at each UNI access switch,
- a network parameter control (NPC) scheme per NNI (internal switch),
- separate buffer queues for the CBS/VBR/ABR/UBR service categories and a mechanism for scheduling the removal of cells from these queues. This is done at every output port, and
- separate thresholds for dropping cells with high loss priority and for forward loss congestion notification.

The basic operation of an ATM switch model is relatively simple. A cell that arrives on input port i with VCI=m and VPI=n is looked up in the routing table, and mapped to output port j with VCI=p and VPI=q. Real switches must be able to do this many thousands of times per second (eg., 53-bytes @ 150 Mbps = 2.83 usec/cell). The VCI/VPI mappings are determined at the time of call setup using an in-band signaling protocol.

The switch models also handle signaling for call setup and release which makes these models much more complex. VCIs are dynamic ie., they are allocated and deallocated on a millisecond-second-minute time scale, while VPIs will be fairly static, ie., they are allocated and deallocated on an hour-week-month time scale.

3.5.1 ATM Signaling

The model of signaling is one of the more complicated parts of the ATM-TN. However, signaling needs to be characterized since the overhead of end-to-end connection setup becomes significant as network speed increases. The model attempts to closely represent the UNI 3.0 specification [ATM 1993].

The model of network functions within switches follows the standard ATM layering, ie., the network layer where the signaling protocol is implemented, the application layer at the end nodes which provide the interface to the traffic models, the signaling ATM adaptation layer which implements segmentation and reassembly, and the ATM layer which manages cell level transactions, are all characterized.

The signaling layer implements ATM connections, and thus the call setup and call release functions. Basic call admission and bandwidth allocation functions are represented. VP, VC, PVC and SVC connections are supported in the switch models.

3.5.2 Switch Model Architecture

The basic switch models have two major components, the control module and the switch fabric. When signaling cells arrive at a switch they are routed to the internal control module. The control module implements the signaling functions by acting on the information carried in these cells and sends out additional signaling cells of its own. This module also makes call admission and routing decisions, and updates the VPI/VCI translation tables within the switch.

The switch fabric transfers cells from an input port to the appropriate output port or to the control module. The switch buffering strategy is implemented in the fabric, eg., output buffering, shared memory buffers, or a crossbar architecture. Multistage switches can be constructed from one of the basic switch types. Most larger switches contain banyon fabrics that can be scaled up to m any thousands of ports. A more complete description of the ATM-TN switch and signaling models is presented in [Gburzynski et al. 95].

4. A High Performance Simulation Environment : SimKit

SimKit is a C++ class library that is designed for very fast discrete event simulation. SimKit presents a simple, elegant Logical Process View of simulation enabling both sequential and parallel execution without code changes to application models. The sequential executive performs well on a variety of UNIX platforms and facilitates debugging and testing. The current parallel executive (WarpKit) is based on the Time Warp paradigm and supports efficient parallel execution on shared memory multiprocessor hardware such as the Silicon Graphics Power Challenge and the Sun SparcServer 1000/2000. A second "conservatively" synchronized parallel executive that is under development is presented in Appendix B.

This section describes the design features of the SimKit System. A brief overview of the Logical Process Modeling View commonly used in Parallel Discrete Event Simulation (PDES) is presented. The SimKit classes are then introduced followed by a brief description of how to build and simulate object oriented models using SimKit.

4.1 Background

The suitability of simulation as a tool to analyze large, complex broadband communication systems depends on very high speed execution. Typical ATM network simulations may require simulating 10^9 to 10^{12} events which at a rate of one event per 50 microseconds requires an execution time of 15 to 1,500 hours. This gives a strong motivation for exploring PDES methods to decrease simulation time by parallel execution.

PDES systems, in general, eliminate the globally shared event list and resort to a synchronization protocol to insure causality between events being executed in the parallel system. The principle distinguishing features of the synchronization approach are the degree of aggressiveness and the risk employed. Synchronization algorithms have been broadly classified as Conservative or Optimistic [Reynolds 88]. The Time Warp mechanism is a well known optimistic approach based upon the Virtual Time paradigm [Jefferson 85]. Time Warp,

essentially non blocking, does not strictly adhere to the local causality constraint. It relies upon a causality error detection and recovery scheme based on a rollback technique.

In the last fifteen years, research in PDES [Fujimoto and Nicol 92, Ferscha and Tripathi 94] have primarily focused on attainable simulation execution speedups. The main reason why PDES has not yet been embraced by the general community is the lack of research contributions toward simplifying the development of simulation models for concurrent execution [Page and Nance 94, Unger and Cleary 93]. Simulation modeling for parallel execution can be a very painstaking process.

PDES assumes a Logical Process Modeling methodology for simulation software development. The system to be simulated is conceptualized as a network of interacting physical sub-systems [Chandy and Misra 79 and 81]. The physical concurrency in these sub-systems translates into computational concurrency in the simulation, which may be utilized through parallel execution. The lack of a global event list, constraints of the synchronization algorithm and the incidental logical process view of modeling make the design of a parallel simulation language difficult. The language should provide a rich set of primitives that facilitate model expression and simulation transparently of the underlying synchronization mechanism while at the same time allowing maximum parallelism to be extracted from the model.

SimKit is a C++ class library that is designed for very fast discrete event simulation. SimKit presents a simple logical process view of simulation enabling both sequential and parallel execution without code changes to the application models. A brief overview of logical process modeling methodology innate within PDES is presented in section 5.2. Section 5.3 lists the issues that dictate parallel simulation language design considerations. The design philosophy adopted in the SimKit System is then outlined followed by the features of the SimKit System.

4.2 Logical Process Modeling Methodology

In PDES, the physical system to be modeled is conceptualized as a system of interacting physical processes (PPs). A distributed simulation model of the physical system is represented by a topologically equivalent system of computational units called logical processes (LPs). Each LP is responsible for simulating events in a sub-space modeling the activities occurring in the corresponding PP. The interaction between PPs is modeled by the corresponding LPs communicating via timestamped messages. The timestamp represents the event occurrence time in the simulation. State transitions in the physical system are modeled by the occurrence of an event, ie., the receipt of a timestamp message at the destination LP. Occurrence of an event may involve modifications to the state and/or the causing of new events in the future.

Distributed simulation accelerates the execution of the program by executing the computational units (LPs) concurrently on multiple processors. By nature, in a distributed simulation system, there is no notion of a global clock or a central event list. LPs execute in parallel by maintaining their own local clock and event list. A synchronization algorithm is used to insure that the local causality

constraint is maintained at each LP. In this logical process modeling methodology the sharing of state between LPs is not possible.

4.3 Design Philosophy

A plethora of parallel simulation languages have appeared in the last decade, each with differing design considerations. These languages include: Common Interface of OLPS [Abrams 88 and 89], Maisie [Bagrodia and Liao 90], ModSim [West and Mullarney 88, Rich and Michelsen 91], MOOSE [Waldorf and Bagrodia 94], SCE from MITRE [Gill et. al. 89], Sim++ from Jade [Baezner, Lomow and Unger 90 and 94], SIMA [Hassam 91], RISE from RAND [Marti 88] and Yaddes [Preiss 89]. The major differences between these languages are their approach to:

- programming paradigm employed and language constructs,
- underlying synchronization protocol and transparency,
- modeling world view,
- run time configuration,
- determinism, and
- efficiency.

The desired characteristics of a parallel simulation language [Abrams and Lomow 90] include: simplicity, modularity, portability, transparency, evolvability, efficiency, scalability, determinism, and generality. "The primary goal of the SimKit System was to provide an event-oriented logical process modeling interface that facilitates building application models for sequential and parallel simulation with high performance execution capabilities."

The design philosophy of the SimKit interface can be summarized as:

- ease of use with the ability to reuse model components,
- indifference towards the underlying simulation control system,
- execution mode transparency,
- encompassing a wide range of applications and user base, and
- efficient event-oriented logical process paradigm,

Specific design features of the SimKit System include an object programming model, transparency, evolvability and efficiency.

Object Oriented Programming Model

The SimKit library is implemented in the commonly-used general-purpose object-oriented programming language, C++. Object oriented techniques allow for reusable simulation models and software components to be developed. The iterative nature of a simulation life cycle is well supported by the object-oriented paradigm. Object-oriented techniques along with good software engineering principles can be used to design reliable simulation software, eg. scoping via objects can be used to create complex simulations involving complex object class hierarchies.

By extending an existing language, the highly optimized library routines for parallel programming provided for a given architecture, can be directly used in the implementation of the underlying synchronization scheme. Moreover, the availability of debuggers, software engineering tools, etc. increase productivity and portability. Application programmers are not burdened with learning a new language and can concentrate on the system modeling task. The SimKit Class library includes only three classes, namely, "sk_simulation" for simulation control, "sk_lp" for modeling application sub-space behavior and state transitions, and "sk_event" to model the interaction between the logical processes.

4.3.1 Transparency

The design of the Parallel SimKit system highlights the need for hiding issues pertaining to the underlying synchronization algorithm, wherever possible. Time Warp relies on a rollback mechanism, hence great care was taken in providing primitives for error handling, output handling and dynamic memory management. However, the application programmer is responsible to efficiently use the various state saving mechanisms provided. State Saving is an artefact of the need to support the rollback mechanism in Time Warp and for the present is left under the jurisdiction of the application programmer as a trade off for efficiency. State restoration during rollback is transparent to the application program. SimKit libraries include a comprehensive set of pseudo-random number distributions with efficient state saving of the seeds. LP allocation to processors in the parallel system is static and may be optionally specified by the programmer.

4.3.2 Evolvability

SimKit provides a general purpose simulation interface. Parallel SimKit has been implemented on the WarpKit executive. Great care was taken to implement a minimal clearly-defined interface to the WarpKit Kernel. The interface consists of three classes, namely, "wk_simulation", "wk_lp" and "wk_event". The SimKit classes (Parallel implementation) are derived from these mirror WarpKit classes. The Time Warp related quasi-operating system services like event-delivery, rollback, and commit are provided as virtual functions in the wk_lp class. This minimizes the impact of modifications to the underlying synchronization system on the Programmer's Interface and consequently the application itself. Evolving schemes and feature extensions can be easily integrated into the simulation control engine without affecting the SimKit programmer's interface.

4.3.3 Efficiency

SimKit is a high performance language that presents an event-oriented logical process view to the model developer. SimKit design goals are to support efficient sequential and parallel execution without code changes to the application model. The sequential simulation control system is a highly optimized simulator that uses the splay tree implementation of the future event list. The parallel simulation control system is capable of exploiting the inherent parallelism in applications having very low event computation granularities.

The parallel run time system, ie., WarpKit, consists of a variation of the Time Warp protocol optimized for execution on shared memory multiprocessors. The global control algorithms are asynchronous in nature, with minimal locking of shared structures. The design of global control algorithms demonstrate a high degree of scalability ie. minimal performance degradation with an increase in the number of processors used. The problem of state saving and restoration in Time Warp is addressed by providing a grab bag of state handler objects that are fine tuned for saving integers, floats, doubles, pointers, variable length state and block states incrementally.

The event-oriented view of simulation enables efficient scheduling of events via invocation of the corresponding LP's event-processing member function rather than the costly context-switching approach in a process-oriented approach. Also, the memory requirements are lower as LP's are active only for the duration of an event ie. no need for a stack per LP in the simulation.

The sequential executive performs well on a variety of platforms and facilitates debugging and testing. The per event scheduling overheads for the sequential executive on an SGI Power Challenge, IRIX64 Release 6.0 IP21 MIPS multiprocessor architecture using the ATT C++ compiler version 3.2.1 with the -O2 compiler option, is 2.2 microseconds. The per event overheads for the parallel executive is 8.6 microseconds.

4.4 Programming Model

The SimKit Programmer's Interface consists of three classes, one type and several convenience functions and macros. The three classes are: "sk_simulation", "sk_lp", and "sk_event". The one type is "sk_time". Also a library for random number generation and distributions is provided which includes classes for uniform, normal, exponential, geometric, binomial, Erlang and Poisson distributions.

The simulation model is constructed by deriving LPs from the sk_lp class and messages (or events) from the sk_event class. The main function is in the modelers domain and may be used to incorporate third party tools like a lex/yacc program code to parse the simulation input parameters. The user instantiates one instance of the sk_simulation class.

This object provides the initialization interface to the run time simulation kernel. The type sk_time is used to represent simulation time and supports standard arithmetic operators associated with the type double. The convenience functions and macros are provided for easy access to simulation structures like current LP, current event, and current time.

The modeler specifies an LP's activity via the sk_lp::process pure virtual member function. The kernel delivers an event to a LP by invoking this function with the event object as a parameter. Typically, the process function is coded as a large case statement with the event type specifying the branching of control. Two other virtual members provided are the sk_lp::initialize and the sk_lp::terminate. These may be optionally defined to initialize the LP and for doing wrap up LP work (eg. printing LP statistics and reports), respectively.

The derived message types may contain other application specific data (besides event type) that are communicated between the LP's. Alternatively, a message may be a query message to determine the value of another LP's state as no shared variables are allowed. A message is created using the SimKit overloaded event's *new* operator. The sk_event::send_and_delete is the only event synchronization primitive that is necessary in this logical process view.

4.5 Time Warp Constraints

The following constraints extraneous to logical process modeling view are native to the parallel synchronization mechanism. Output handling and error handling should be rollback-able. Hence the modeler should use the member functions provided in the sk_lp class. General purpose memory management is another facility that is provided via the sk_lp class. The request for dynamic memory does not return if the allocation request cannot be satisfied due to system resource exhaustion. To support parallel simulation in particular, the simulation execution phase has been divided into 6 clearly defined phases.

State Saving calls must be explicitly programmed with the LP's process function. A grab bag of state savers are provided within the sk_lp class for saving basic data types, variable length data sizes and block state incrementally [Gomes et. al. 95 and Gomes 96]. Options of using Copy State Saving instead of Incremental State Saving mechanisms allows simulations programs to be debugged for correctness before optimizing using ISS mechanisms.

Constraints of the Time Warp mechanism and the parallel executive in particular include: (i) event ownership is retained by the kernel when a LP processes the event (saved in the input queue until fossil collection), (ii) lack of any input facility during simulation execution, (iii) lack of interactivity during simulation execution, output occurs (usually in bursts) when Time Warp commits events, (iv) prohibition against using global memory during simulation execution, and (v) prohibition against LPs invoking member functions of other LPs directly, during simulation execution

4.6 Run Time Configuration

Facilities for collecting statistics about the simulation execution, for tracing simulation execution and debugging are provided. Two control mechanisms for this are provided. One is via compilation flags that conditionally compile into tracing and debugging code and the other is via run time flags that specify which bits of the trace and debug code to activate. Run time flags may be set via configuration file, the command line or the sk_simulation interface.

4.6.1 Execution Phases

Executions starts and ends with a single thread of control executing on a single processor. The six phases are listed below:

1. Program Initialization: The function main begins execution. Command line arguments are stripped off here and the sk_simulation object is instantiated.

2. SimKit and Model Global Initialization: In this phase the sk_simulation object is initialized. Command line arguments are stripped off and SimKit configuration parameters read from an external file (filename may be specified via command line argument). LPs are instantiated and allocated to processors.

   Allocation of LPs to processor is static and may be optionally specified by the modeler via the LPs constructor. Global application data structures (READ only variables) are built during this phase. This phase ends with passing control to the simulation run time system.

3. Logical Process Initialization: Each LP's initialize member function is invoked once. Simulation time does not advance in this phase and no events are received. This phase is used to send out seed events to start the simulation.

4. Simulation Execution: Execution is now under Time Warp control. Events are delivered to LPs by invoking sk_lp::process. Here the constraints mentioned in the earlier section are enforced. This phase terminates either normally due to end of simulation or abnormally due to an error in the simulation.

5. Logical Process Termination: Each LPs terminate function is invoked. This phase is typically used for reporting LP specific statistics.

6. Simulation Clean-Up: The simulation run time system returns control back to the main function. This phase is generally used to tally statistics and output final reports.

5. Preliminary Experiments Using the ATM-TN

Three sets of experiments have been performed using the ATM-TN simulation environment. First a series of model component validation experiments were performed. These were used to test and evaluate the individual traffic and switch model components. These results can be found in [Williamson 95].

The second set of experiments involved a Canadian research network called Wnet. A few of these results are presented here to illustrate how the ATM-TN can be used. The third set of experiments was a very preliminary characterisation of the OPERA network.

5.1 A Western Canadian (Wnet) Experiment

The Wnet is a trial ATM network dedicated to research and development purposes. At the time of this study Wnet spanned the three prairie provinces of Western Canada, ie., Alberta, Saskatchewan and Manitoba. A preliminary analysis of the Wnet ATM network was made using the ATM-TN simulator. Wnet consists of 12 Newbridge ATM switches installed across three Canadian provinces: Alberta, Saskatchewan and Manitoba. A simplified diagram of the Wnet configuration is shown in Figure 2. The links shown are all DS3 at 45 Mb/s. The 11 links shown in the figure are labelled 2, 3, 6, 7, 10, 14, 15, 16, 17, 20 and 21. There is also cross traffic through Wnet, into the EdTel CO switch

to/from British Colombia (BC) in Western Canada and out of the Winnipeg MTS switch to/from Ontario and Eastern Canada.

Results for three benchmark experiments are presented. The first has traffic loads as defined in Table 1 which specifies the required bandwidth and quality of service in terms of cell loss ratio and maximum cell transfer delay. The second has an increased load across the network and reflects an upgraded version of Wnet where links 3, 6, 17, 20 and 21 were changed to OC3. The third benchmark has one more link upgraded. The user bandwidth specification represents the peak bandwidth needed with the average utilization assumed to be 80%.

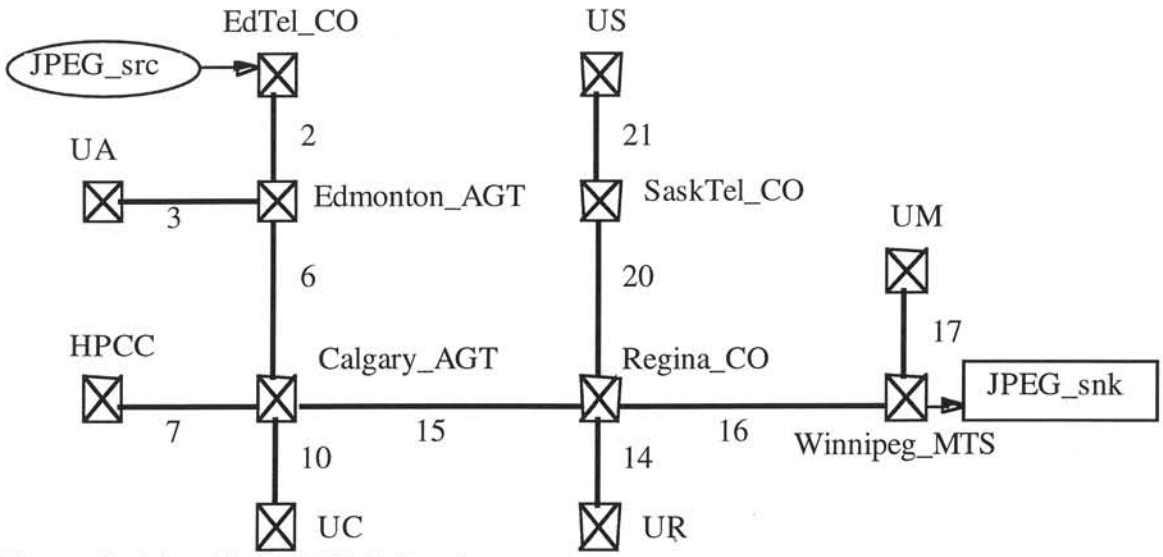


Figure 2 The Wnet ATM Network

Source	Destination	Service	Number	Bandwidth	CLR	Max Delay
EdTel_CO	Winnipeg_MTS	Ethernet	1	4.8	1.00E-7	0.03
EdTel_CO	Winnipeg_MTS	JPEG	2	16	1.00E-7	0.03
UA	UC	Ethernet	1	4.8	1.00E-7	0.002
HPCC	UC	Ethernet	2	4.8	1.00E-7	0.0002
UC	US	Ethernet	1	8	1.00E-7	0.0075
UR	UM	Ethernet	1	2	1.00E-7	0.005

Table 1 Estimated Wnet User Bandwidth & QoS Requirements

Tables 2 and 3 present the results for a 20 second simulation of Wnet operation. In all experiments data collection was started after a 5 second warm-up period. These results can be compared with the user PVC and traffic load requirements. Network performance generally meets the requirements and the link utilizations shown in the table are close to the operating telco estimates obtained from other sources. The only marginal area is link 15 which has a utlization of 87%.

In the second benchmark Wnet cross traffic from BC to Ontario was increased to 8Mb/s from the 4.8Mbps of Benchmark 1. The same network topology was used

as in the first benchmark except that links 3, 6, 17, 20 and 21 are upgraded to OC3 to reflect the current configuration of Wnet.

Source Switch	Destination Switch	Service	Number of PVCs	Bandwidth (Mbps)	CLR	Average Cell Delay (sec)
EdTel_CO	Winnipeg_MTS	Ethernet	1	4.8	0	0.0218
EdTel_CO	Winnipeg_MTS	JPEG	2	16	0	0.0218
UA	UC	Ethernet	1	4.8	0	0.0015
HPCC	UC	Ethernet	2	4.8	0	0.0001
UC	US	Ethernet	1	8	0	0.0051
UR	UM	Ethernet	1	2	0	0.0039

Table 2 Simulated QoS Results for Benchmark 1

Name	Source	Destination	Size	Utilization (%)
Link-2	Edmonton_AGT	EdTel_CO	DS3	69.54
Link-6	Calgary_AGT	Edmonton_AGT	DS3	79.72
Link-7	Calgary_AGT	HPCC	DS3	20.64
Link-10	Calgary_AGT	UC	DS3	50.00
Link-15	Calgary_AGT	Regina_CO	DS3	87.03
Link-16	Regina_CO	Winnipeg_MTS	DS3	73.86
Link-17	UM	Winnipeg_MTS	DS3	4.33

Table 3 Simulation Utilization Results for Benchmark 1

The Quality of Service results for Benchmark 2, listed in Table 4, show serious traffic congestion. The 1.4E-3 cell loss ratio is far worse than the required 1.0E-7. The difficulty is link 15 which is still a DS3 link despite the increased Wnet transfer traffic from BC to Ontario. The Utilization for link 15 is plotted versus simulation time in Figure 3. Here it can be seen that cells will be lost when utilization approaches 100%.

Examples of cell delay and delay variation (standard deviation of cell delay) versus simulation time are presented in Figures 4 and 5. The buffer occupancy is plotted in Figure 6. These illustrate how simulation results can be used to examine the impact of changes in traffic load assumptions.

Simulation results are presented in Tables 4 and 5 for Benchmark 3 which is identical to Benchmark 2 except that link 15 upgraded to OC3. These results show no congestion.

Source Switch	Destination Switch	Service	Number of PVCs	Bandwidth (Mbps)	CLR	Ave_Cell Delay (sec)
EdTel_CO	Winnipeg_MTS	Ethernet	1	8	1.44E-3	0.0219
UA	UC	Ethernet	1	4.8	0	0.0015
HPCC	UC	Ethernet	2	4.8	0	0.0001
UC	US	Ethernet	1	8	1.38E-3	0.0051
UR	UM	Ethernet	1	2	0	0.0038

Table 4 Quality of Service Results for Benchmark 2

Source Switch	Destination Switch	Service	Number of PVCs	Bandwidth (Mbps)	CLR	Ave_Cell Delay (sec)
EdTel_CO	Winnipeg_MTS	Ethernet	1	8	0	0.0218
EdTel_CO	Winnipeg_MTS	JPEG	2	16	0	0.0218
UA	UC	Ethernet	1	4.8	0	0.0015
HPCC	UC	Ethernet	2	4.8	0	0.0001
US	UC	Ethernet	1	8	0	0.0015
UR	UM	Ethernet	1	2	0	0.0039

Table 5 Quality of Service Results for Benchmark 3

Name	Source	Destination	Size	Utilization (%)
Link-2	Edmonton_AGT	EdTel_CO	DS3	76.64
Link-6	Calgary_AGT	Edmonton_AGT	OC3	25.15
Link-7	Calgary_AGT	HPCC	DS3	20.64
Link-10	Calgary_AGT	UC	DS3	50.00
Link-15	Calgary_AGT	Regina_CO	OC3	27.73
Link-16	Regina_CO	Winnipeg_MTS	DS3	80.74
Link-17	UM	Winnipeg_MTS	OC3	1.26

Table 6 Utilization Results for Benchmark 3

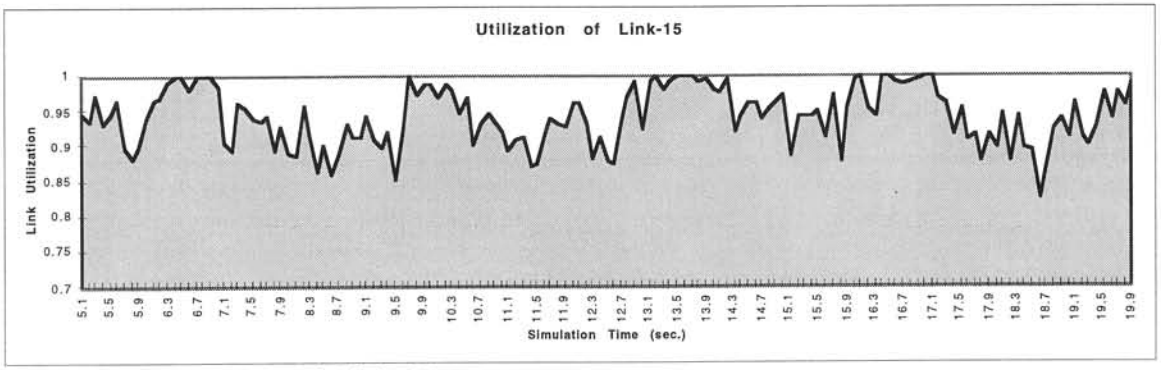


Figure 3 Utilization of Link 15

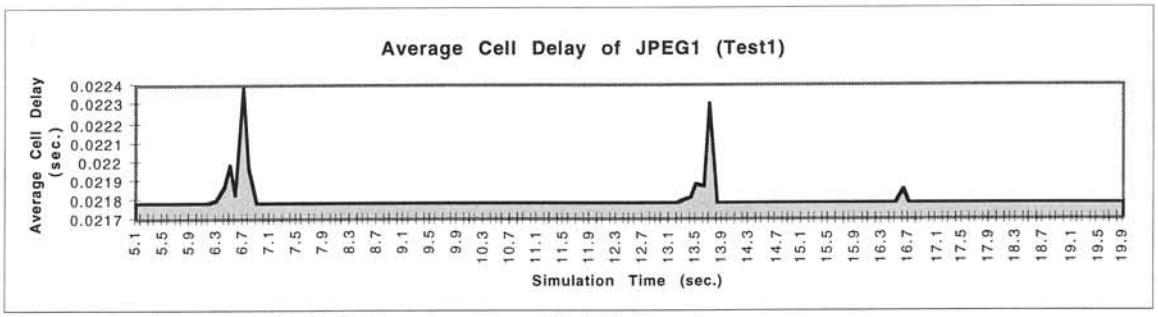


Figure 4 Average Cell Delay for the JPEG1 Load

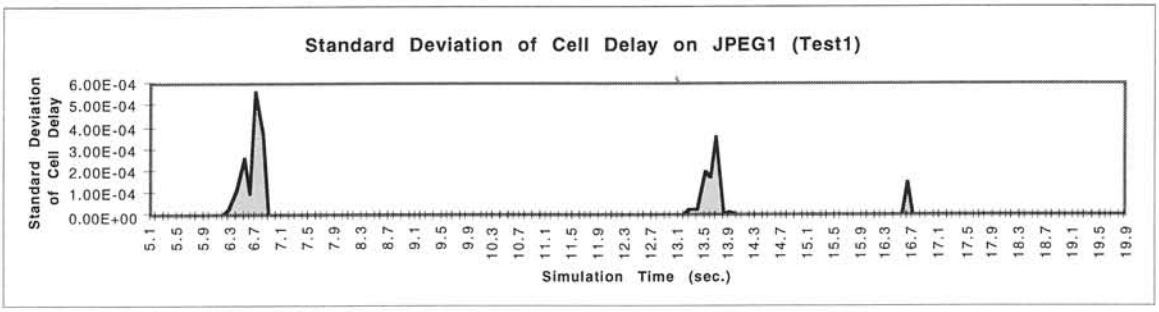


Figure 5 Cell Delay Variation

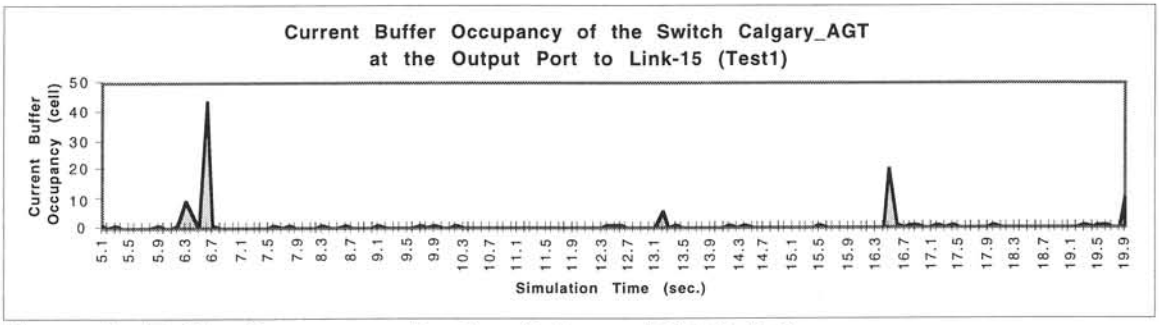


Figure 6 Buffer Occupancy for the Calgary_AGT Switch

5.2 Telecom New Zealand's OPERA Experiments

ATM network technology is being evaluated by TeleCom New Zealand, the Tuia Society and Netway (TeleCom's systems integration company) in a three year program (1996-1998) called OPERA (Organised Programme of Experimentation and Research into ATM). The OPERA network is shown in Figure 7.

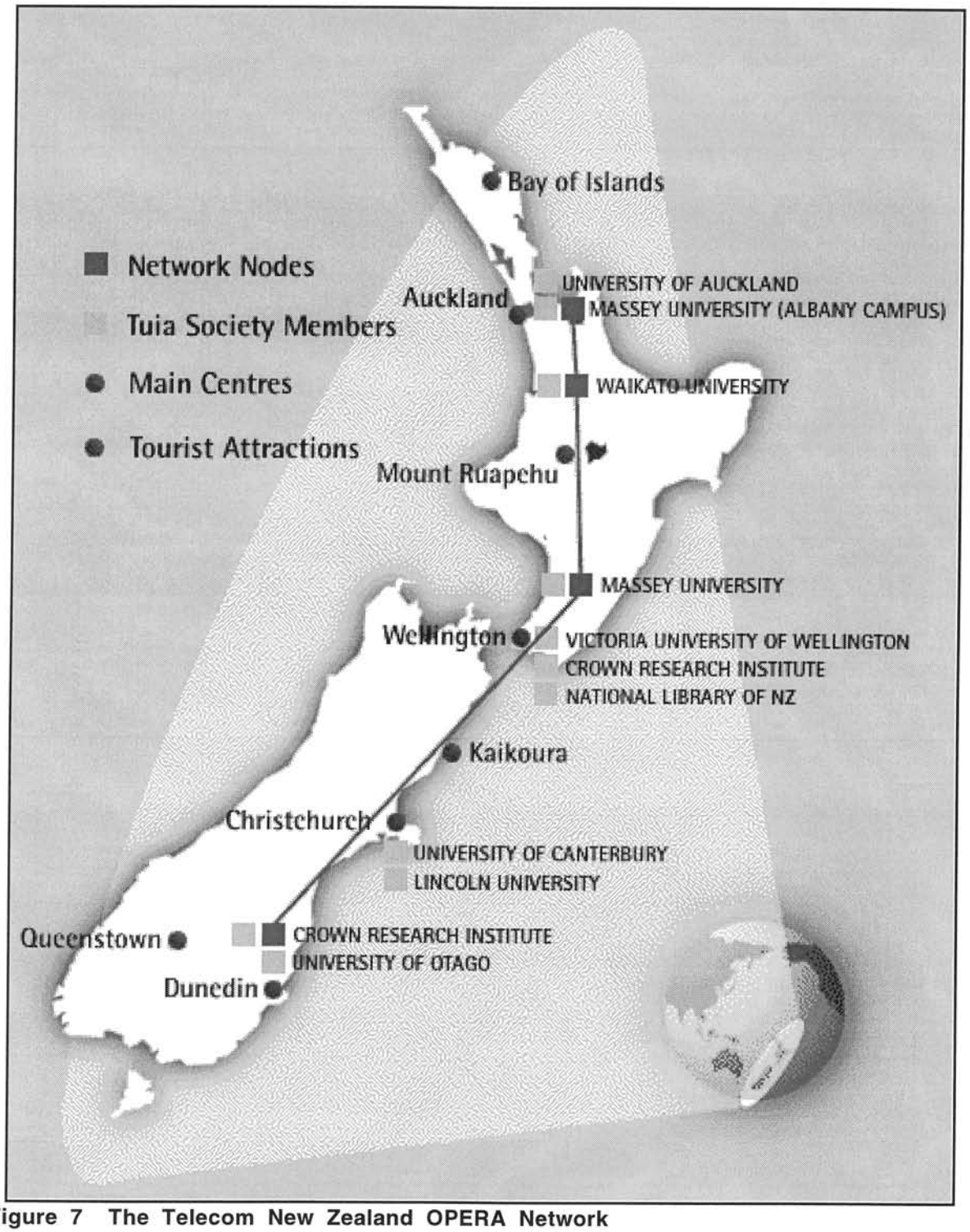


Figure 7 The Telecom New Zealand OPERA Network

There are four main network nodes: two at Massey University, one at Waikato University, and one at the Crown Research Institute in Dunedin. The Tuia Society members are the Universities of Auckland, Canterbury, Dunedin, Lincoln, Massey, Victoria and Waikato, plus the Crown Research Institute and the National Library of NZ.

A preliminary experiment simulating OPERA network operation under a benchmark traffic load was conducted using the ATM-TN Simulator. The backbone and edge network connections used in this experiment are shown in Figure 8 below.

The following models and parameter values were used in this experiment:

- Switches are per port with buffers of size 32 packets on each port
- Ethernets were 10 Mb/s links at 30% utilization - bidirectional (SVC)
- One 25 Mb/s MPEG PVC - 1 frame in GOP, 1 for length of recurring substructure, 30 frames/sec, smoothing within 1 frame, scaling of 0.8
- Mosaic - 50 conversations at each source/sink, average interarrival time between converstions of 5 seconds, destination same probability 0.36, 2.5 TCP connections per conversation, mean interarrival time between connections in each conversaion of 0.5 sec, and the default message length specifications

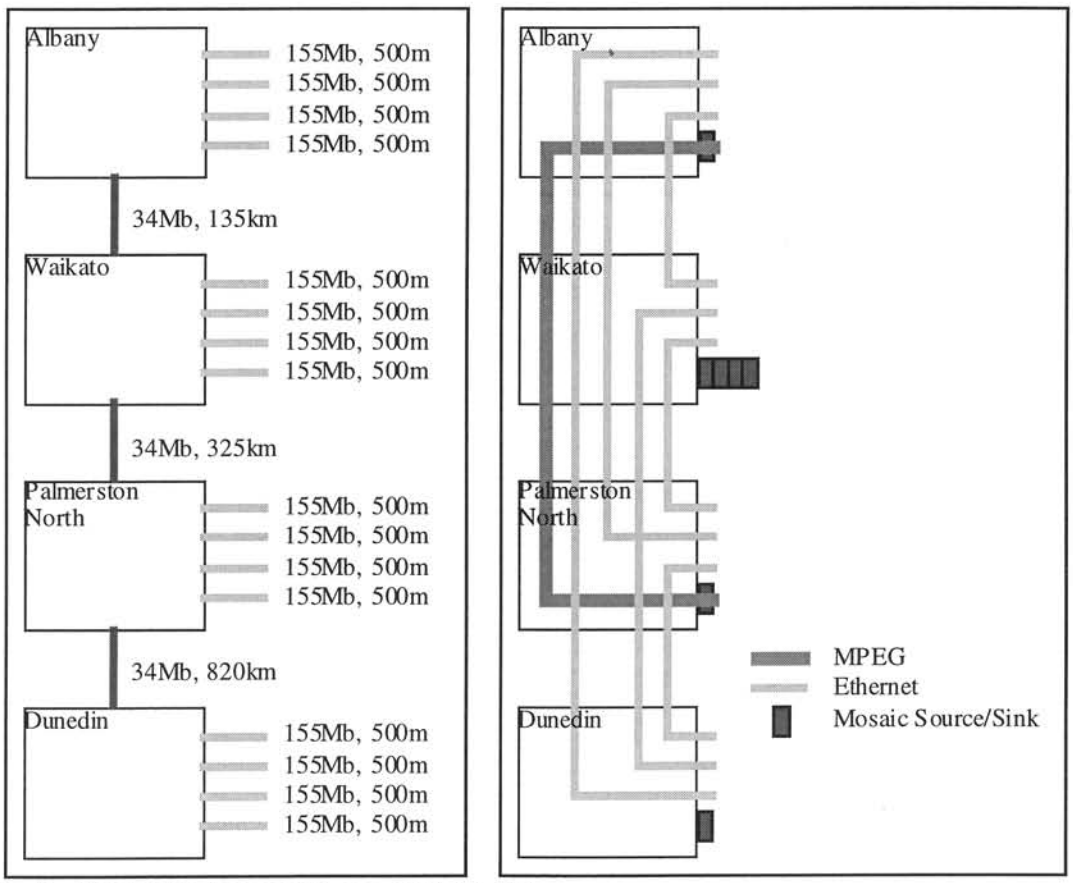


Figure 8 Network Connections and Links

The simulation results reported for this experiment are intended to illustrate how the simulator can be used. They are not intended to represent a realistic OPERA scenario. Each simulation experiment had a 5 second warmup period followed by 20 seconds of simulated time. Data was collected only during the 20 second simulation time, not during warmup. Statistics were collected every 0.1 seconds. It took about 16 minutes to run each simulation on a Sparc Server 1000 and each run produced around 3.5 Mbytes of data and statistics.

Figures 9 and 10 show the mean and standard deviation of the cell delays for the single MPEG traffic stream (from source to sink). Figures 11 and 12 show the same for cell delays of the Ethernet traffic from Albany to Dunedin (from end to end). Figures 13 and 14 plot the average and standard deviation of the cell delays of all traffic that crosses the Albany-Waikato link.

Figures 15 and 16 show the buffer occupancy at the Palmerston North output buffer of the Waikato switch (the first is scaled to the buffer size (32) and the second is scaled to the maximum occupancy (2)). Figures 17 and 18 show the utilisation of the Albany-Waikato link (the second Figure is scaled to make the differences stand out more). Figure 19 plots the number of cells dropped over the Albany-Waikato link. This illustrates that even though the link never reaches full utilisation some cells are still lost. In this case all the cells dropped were from the Mosaic traffic. Finally, the cell delay for each traffic source and sink stream is presented in Table 7.

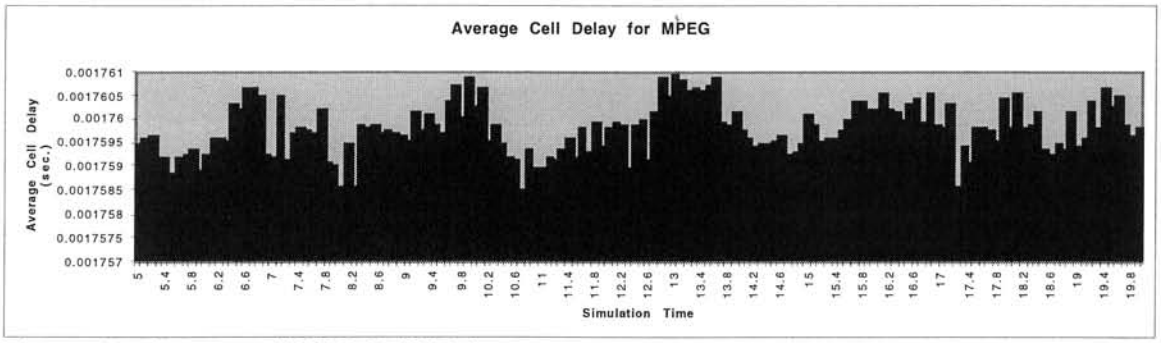


Figure 9 Average MPEG Cell Delay

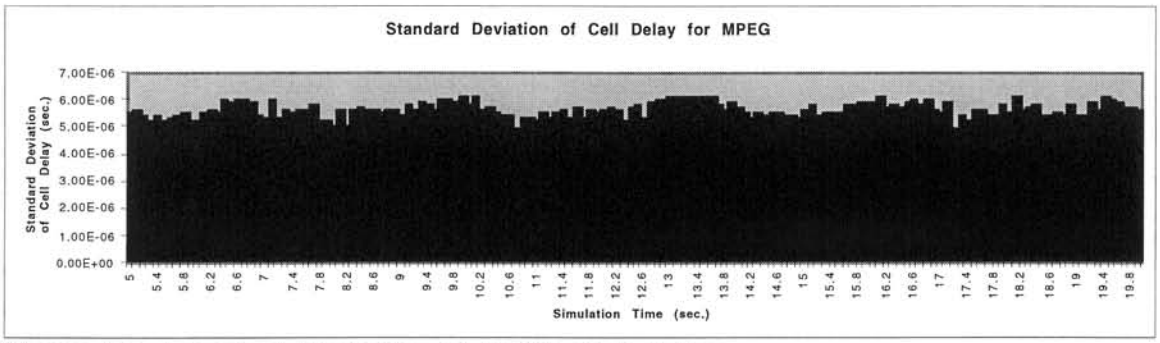


Figure 10 Standard Deviation of MPEG Cell Delay

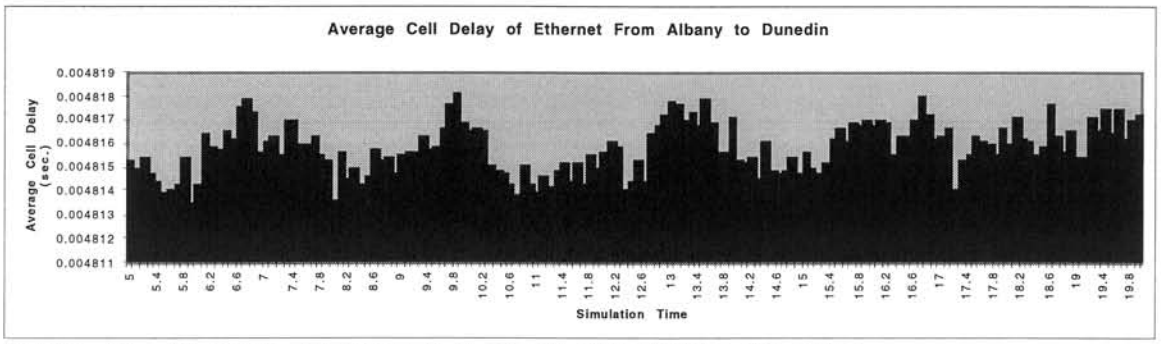


Figure 11 Average Ethernet Cell Delay (Albany to Dunedin)

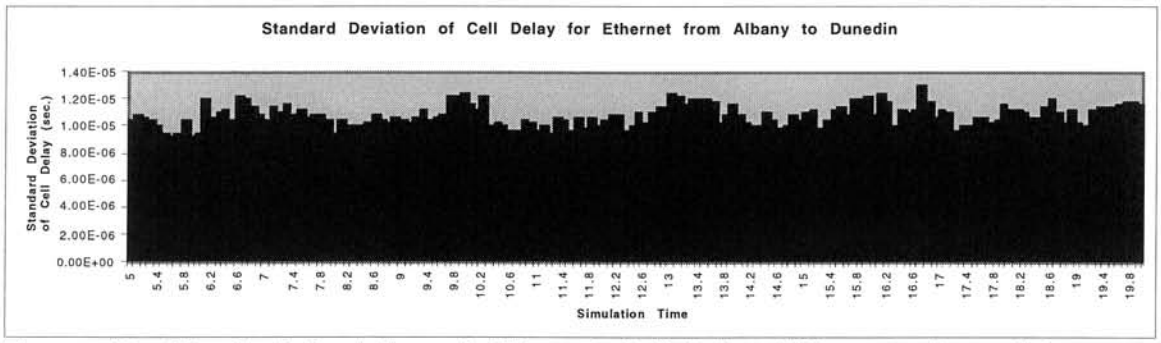


Figure 12 Standard Deviation of Ethernet Cell Delay (Albany to Dunedin)

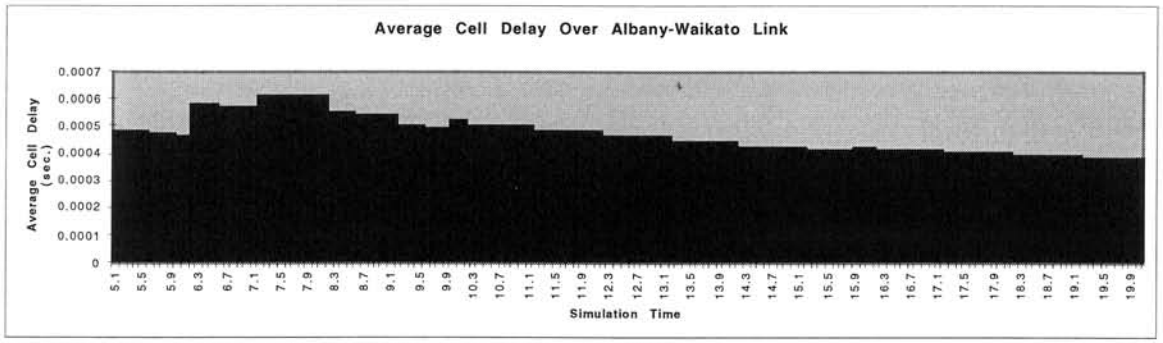


Figure 13 Average Cell Delay (Albany - Waikato)

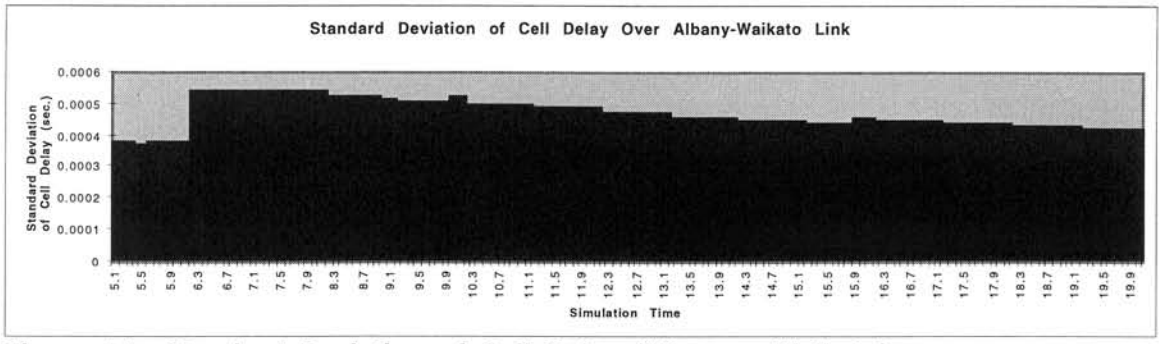


Figure 14 Standard Deviation of Cell Delay (Albany - Waikato)

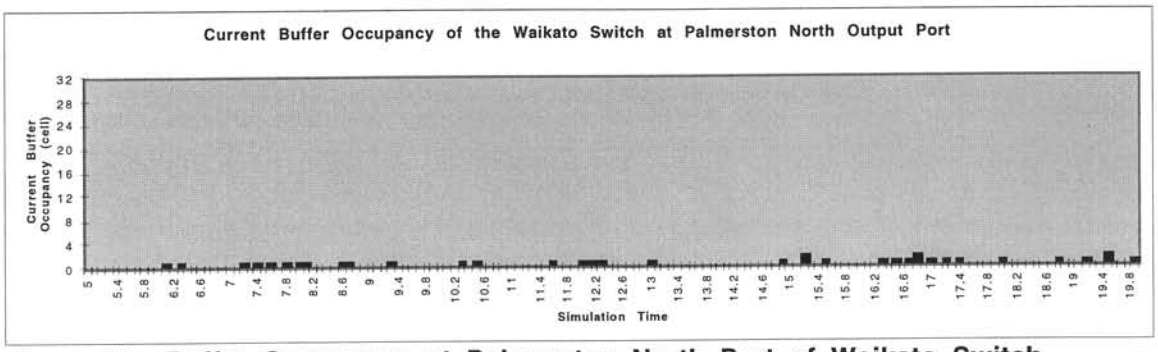


Figure 15 Buffer Occupancy at Palmerston North Port of Waikato Switch

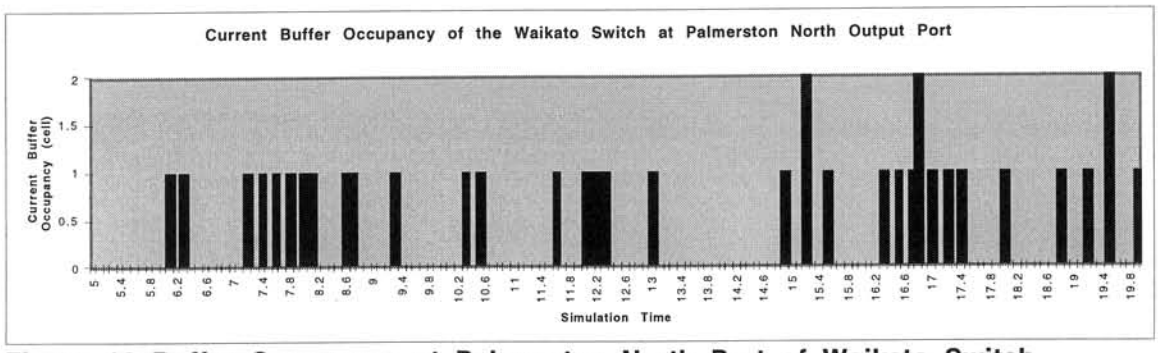


Figure 16 Buffer Occupancy at Palmerston North Port of Waikato Switch

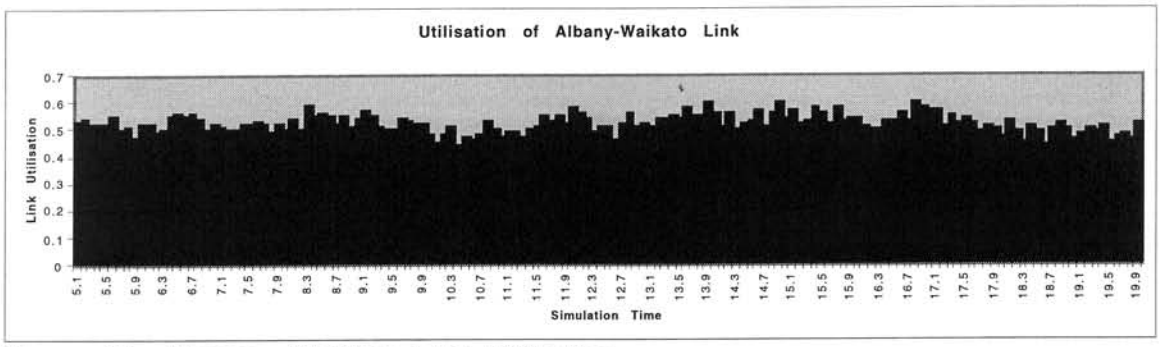


Figure 17 Albany - Waikato Link Utilization

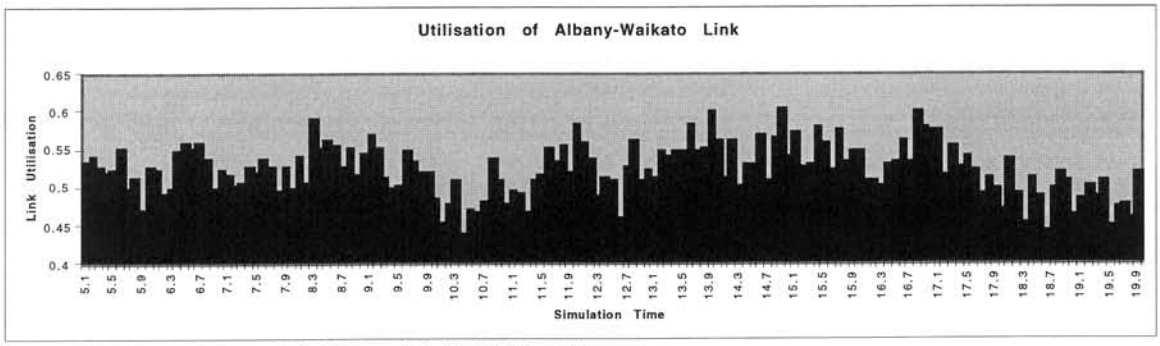


Figure 18 Albany - Waikato Link Utilization

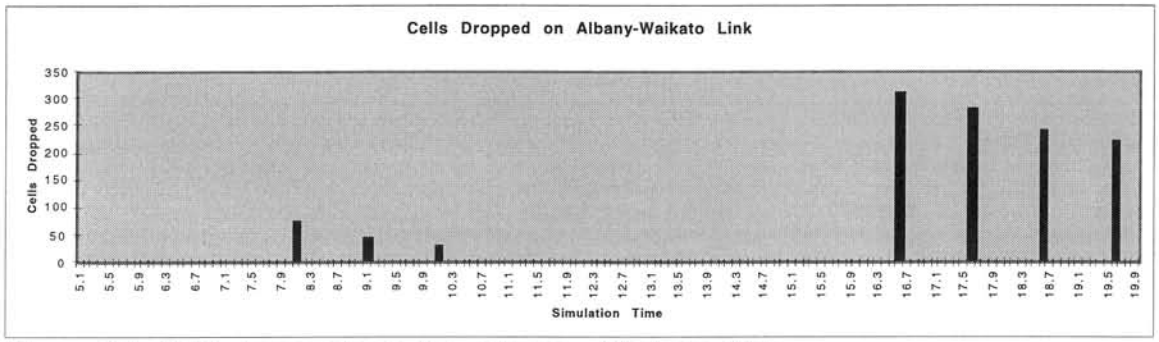


Figure 19 Cells Dropped on the Albany - Waikato Link

Source	Destination	Service	Expected Delay	Average Delay	Max Delay	Standard Deviation
Albany	Waikato	Ethernet	0.000536247	0.000541674	0.000586182	7.10321e-06
Waikato	Albany	Ethernet	0.000536247	0.00053737	0.000551918	2.84322e-06
Albany	Palmerston North	Ethernet	0.00175522	0.00176465	0.00184929	1.0571e-05
Palmerston North	Albany	Ethernet	0.00175522	0.00175772	0.00179254	4.60184e-06
Albany	Dunedin	Ethernet	0.00480569	0.00481581	0.00489959	1.09092e-05
Dunedin	Albany	Ethernet	0.00480569	0.0048092	0.00486004	5.68489e-06
Waikato	Palmerston North	Ethernet	0.00123925	0.00124534	0.00129872	7.36815e-06
Palmerston North	Waikato	Ethernet	0.00123925	0.00124117	0.00127326	3.92735e-06
Waikato	Dunedin	Ethernet	0.00428972	0.00429651	0.00434753	7.73641e-06
Dunedin	Waikato	Ethernet	0.00428972	0.0042924	0.00433996	4.99129e-06
Palmerston North	Dunedin	Ethernet	0.00307075	0.00307184	0.00308322	2.80559e-06
Dunedin	Palmerston North	Ethernet	0.00307075	0.00307186	0.00309524	2.98747e-06
Albany	Palmerston North	MPEG	0.00175522	0.00175981	0.00178253	5.67449e-06

Table 7 Cell Delay for Each Traffic Source and Sink Stream

6. Performance of the ATM-TN Simulator

Preliminary performance results are presented in Figure 20 below for the Wnet benchmark experiments of section 5.2 using optimistic synchronization and WarpKit. These represent results for a simulation of Wnet executed on from 1 to 12 Silicon Graphics R8000 CPUs for an SGI PowerChallenge multiprocessor platform. The dotted line represents perfect speedup, ie., a simulation runs 4 times faster on 4 processors (CPUs).

The "relative" speedup is defined as the time it takes to run the simulation on 1 processor divided by the time it takes on "n" processors using identical software. The latter refers to the parallel system consisting of the Wnet model and scenario on the ATM-TN on SimKit and on WarpKit. The "absolute" speedup is defined in the same way except the sequential execution time for the optimized sequential system (OSS) is used rather than the WarpKit time on 1 processor.

This shows an effective (absolute) speedup of 4.4 on 12 processors. This result suggests that parallel execution on a multiprocessor platform may significantly reduce the required time to execute a simulation experiment. However, there was considerable additional complexity in the Wnet model required to enable parallel execution. It is also not clear that this level of speedup is possible with a range of network models and traffic load scenarios.

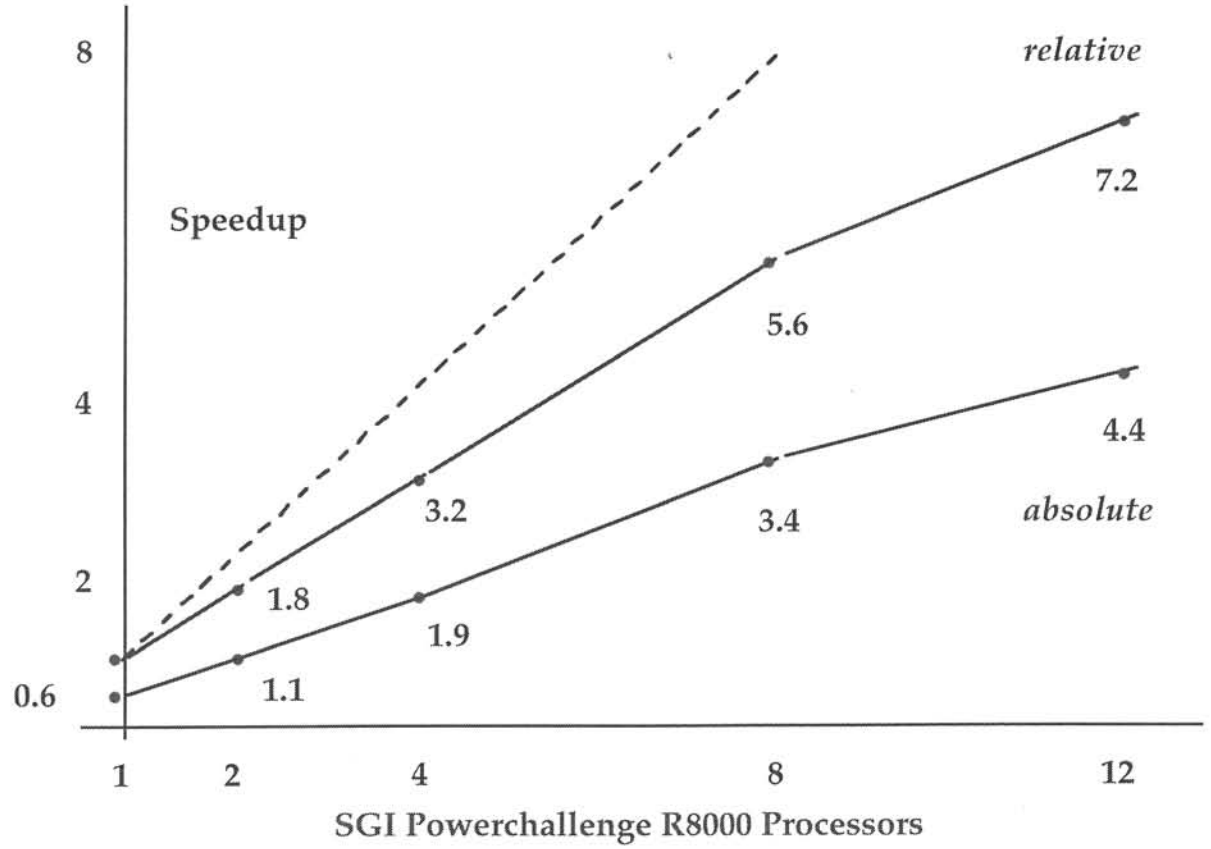


Figure 20 Preliminary Simulator Performance for the Wnet Scenario

7. Conclusions

The preliminary experiments conducted for the Wnet and OPERA scenarios strongly suggest that the dynamic behavior of an ATM network can be adequately captured using simulation. Dynamic cell loss, cell delay and cell delay variation can all be produced using the ATM-TN simulator. Although failures were not introduced in these experiments, it would be relatively easy to introduce a broad range of dynamic and static failure types. Further experiments will be needed, however, to quantify the accuracy of simulation results.

The speed of execution of the ATM-TN simulator is marginally adequate for scenarios of ten to one hundred ATM switches and ten to one hundred traffic source /sink streams. The Wnet scenario running sequentially on 1 SGI R8000 CPU takes about 70 CPU execution seconds per simulated second without compiler optimization. Thus it takes about 70 minutes to simulate 1 minute or 11.5 hours to simulate 10 minutes of Wnet operation.

With the compiler optimization flag "on" this might be reduced to 50 CPU seconds per simulated second or about 8.5 hours for 10 minutes. On twelve processors this would be reduced by the factor of 4.4 shown in Figure 20, giving less than 2 hours to simulate 10 minutes of Wnet operation.

Further speedup via parallel execution may also be possible with a new conservative synchronization mechanism that is described in the paper of Appendix B. Ultimately this may require less added model source code complexity than needed for the optimistic methods of WarpKit.

There are also many situations where a cell level simulation will not be needed. Siemens has made modifications to the ATM-TN to ignore all data cells, ie., only SVC set up and tear down control cells are simulated with one scheduled delay for the data cells across each SVC. Their preliminary experience with this call-level version of the ATM-TN shows performance of about 60 calls per CPU second on a Sparc 10 for a path of two endnodes and three switches. This means about 12 seconds on a Sparc 10 to simulate a busy hour (3,600 seconds) for an E1 line with a capacity of 32 telephone connections (PCR=150).

In conclusion, the simulation of ATM traffic and networks is both feasible and practical. It is expected that adequate accuracy can be achieved to predict dynamic network behavior under varying traffic loads and failures. Further validation experiments will be needed to quantify the accuracy of such simulations.

Long simulation execution times are required to perform simulations of realistic ATM network and traffic scenarios. Currently these can be as high as 12 hours to execute a 10 minute network simulation. However, techniques are being explored that can substantially reduce these execution times. Preliminary results already show that this 12 hour execution time can be reduced to less than 2 hours for a 10 minute cell-level simulation of a 12 switch, 8 traffic stream scenario.

Bibliography

Abrams, M. 1988. "The Object Library for Parallel Simulation", Proceedings of the 1988 Winter Simulation Conference, San Diego, California, 210--219, December.

Abrams, M. 1989. "A Common Interface for Chandy-Misra, Time-Warp, and Sequential Simulators", Proceedings of the 1989 Winter Simulation Conference, Washington, D.C., December.

Abrams, M. and Lomow, G. 1990. "Issues in Languages for Parallel Simulation", Proceedings of the 1990 SCS Multiconference on Distributed Simulation, 22(2)", San Diego, California, 227--228, January.

ACM. 1995. Special Edition on "Issues and Challenges in ATM Networks", Communications of the ACM.

Arlitt, M., Chen, Y., Gurski, R. and Williamson, C.L. 1995a. "Traffic Modeling in the ATM-TN TeleSim Project: Design, Implementation, and Performance Evaluation", Proceedings of the 1995 Summer Computer Simulation Conference, Ottawa, Ontario, July.

Arlitt, M. and Williamson, C.L. 1995b. "A Synthetic Workload Model for Internet Mosaic Traffic", Proceedings of the 1995 Summer Computer Simulation Conference, Ottawa, Ontario, July.

ATM Forum. 1993. "ATM User-Network Interface Specification", Version 3.0, Prentice Hall, New Jersey.

Bagrodia, R. and Liao, W.T. 1990. "Maisie: A Language and Optimizing Environment for Distributed Simulation", Proceedings of the 1990 SCS Multiconference on Distributed Simulation, San Diego, California, 22(2), 205--210, January.

Baezner, D., Lomow, G.A. and Unger, B.W. 1994. "A Parallel Simulation Environment Based on Time Warp", International Journal in Computer Simulation, 4(2), 183-207.

Baezner, D. and Lomow, G. and Unger, B. 1990. "Sim++: The Transition to Distributed Simulation", Proceedings of the 1990 SCS Multiconference on Distributed Simulation, San Diego, California, 22(2), 211--218, January.

Chandy, K. M. and Misra, J. 1979. "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs", IEEE Transactions on Software Engineering", SE-5, 440--452, September.

Chandy, K. M. and Misra, J. 1981. "Asynchronous Distributed Simulation via a Sequence of Parallel Computations", Communications of the ACM, 24(11), 198--206, April.

Chen, Y., Deng, Z. and Williamson, C.L. 1995. "A Model for Self-Similar Ethernet LAN Traffic : Design, Implementation and Performance Implications", Proceedings of the 1995 Summer Computer Simulation Conference, Ottawa, Ontario, July.

Cleary, J. and Gomes, F. and Unger, B. and Zhonge, X. and Thudt, R. 1994. "Cost of State Saving and Rollback", Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS94), Edinburgh, Scotland, 24(1), 94--101, July.

Ferscha, A. and Tripathi, S. K. 1994. "Parallel and Distributed Simulation of Discrete Event Systems", Technical Report CS-TR-3336, Department of Computer Science, University of Maryland, College Park, MD, August.

Fujimoto, R. 1993. "Parallel and Distributed Discrete Event Simulation: Algorithms and Applications", Proceedings of the 1993 Winter Simulation Conference Proceedings, Los Angeles, 106--114, December.

Fujimoto, R. and Nicol, D. 1992. "State of the art in Parallel Simulation", Proceedings of the 1992 Winter Simulation Conference", Arlington, Virginia, 25, 246-254, December.

Fujimoto, R. M. 1990a. "Parallel Discrete Event Simulation", Communications of the ACM, 33(10), 33-53, October.

Fujimoto, R., 1990b. "Time Warp on a Shared Memory Multiprocessor", Transactions of the Society of Computer Simulation, 6(3), 211-239, July.

Fujimoto, R.M. 1990c. "Parallel Discrete Event Simulation", Communications of the ACM, 33(10), 30-53.

Gburzynski, P., Ono-Tesfaye, T. and Ramaswamy, S. 1995. "Modeling ATM Networks in a Parallel Simulation Environment: A Case Study", Proceedings of the 1995 Summer Computer Simulation Conference, Ottawa, Ontario, July.

Ghosh, K. and Fujimoto, R. 1991. "Parallel Discrete Event Simulation Using Space-Time Memory", Proceedings of the International Conference on Parallel Processing, 3, 201-208, August.

Gill, D. H. and Maginnis, F. X. and Rainier, S. R. and Reagan, T. P. 1989. "An Interface for Programming Parallel Simulations", Proceedings of the SCS Multiconference on Distributed Simulation (eds Unger, B. and Fujimoto, R.), Tampa, Florida , 21(2), 151-154, March.

Gomes, F. (1996) "Optimizing Incremental State Saving and Restoration", Ph.D. thesis, in progress, Department of Computer Science, University of Calgary.

Gomes, F., Cleary, J., Covington, A., Franks., S., Unger, B., and Xiao, Z. 1995. "SimKit: A High Performance Logical Process Simulation Class Library in C++", Proceedings of the 1995 Winter Simulation Conference, Washington D.C., December.

Gurski, R., Williamson, C., Arlitt, M., Covington, A., Gburzynski, P., Gomes, F., Ono-Tesfaye, T., Ramaswamy, S., Unger, B.W. & Xiao, Z. 1995. "ATM-TN User Manual", WurcNet Technical Report, July, 12 pages.

Hassam, R. 1991. "SIMA: A Parallel Simulation Environment", Technical Report TRITA--TCS--91007, Dept. of Telecomm. and Computer Science, The Royal Institute of Technology, S--10044, Stockholm, Sweden.

Jefferson, D.R. 1985. "Virtual Time", ACM Transactions on Programming Languages and Systems, 7(3), 404-425, July.

Marti, J. 1988. "RISE: The RAND Integrated Simulation Environment", Proceedings of the 1988 SCS Multiconference on Distributed Simulation (eds Unger, B. & Jefferson, D.), San Diego, Calif., 19(3), 68--72, February.

Mehl, H. and Hammes, S. 1993. "Shared Variables in Distributed Simulation", Proceedings of the 7th Workshop on Parallel & Distributed Simulation (PADS93), San Diego, California, 23(1), 68-75, July.

Melamed, B. and Hill, J. 1995. "A Survey of TES Modeling Applications", Simulation 64 (6), 353 - 370, June.

Nikolaidis, I. and Onvural, R.O. 1992. "A Bibliography of Performance Issues in ATM Networks" Computer Communication Review, ACM Sigcomm, 22(5), 8-23.

Page, E. H. and Nance, R. E. 1994. "Parallel Discrete Event Simulation: A Modeling Methodological Perspective", Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS94), Edinburgh, Scotland, 24(1), July.

Pearson, M., Cleary, J., Unger, B. and Williamson, C. 1996 "Current Techniques for Measuring and Modeling ATM Traffic" Technical Report, Computer Science Department, Waikato University, 36 pages.

Perloff, M. and Reiss, K. 1995. "Improvements to TCP Performance in High-Speed ATM Networks" Communications of the ACM, 90-109, February.

Preiss, B. R. 1989. "The Yaddes Distributed Discrete Event Simulation Specification Language", Proceedings of the SCS Multiconference on Distributed Simulation, Tampa, Florida, 21(2), 139--144, March.

Rajaei, H. and Ayani, R. 1992. "Language Support for Parallel Simulation", Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS92), New Port Beach, California, 24(3), 191--192, January.

Reynolds, Jr. P. 1988. "A spectrum of options for parallel simulation", Proceedings of the 1988 Winter Simulation Conference, San Diego, California, 325--332, December.

Rich, D. O. and Michelsen, R. E. 1991. "An assessment of the ModSim / TWOS Parallel Simulation Environment", Proceedings of the 1991 Winter Simulation Conference, 509--517, December.

Unger, B.W., Goetz D.J. and Maryka, S.W. 1994a. "The Simulation of SS7 Common Channel Signaling", IEEE Communications, 32(3), 52-62.

Unger, B.W. and Xiao, Z. 1994b. "The Fast Parallel Simulation of Telecommunication Networks", Proceedings of the 1994 Conference on New Directions in Simulation for Manufacturing and Communications, Operations Research Society of Japan, 9-16, August.

Unger, B.W., Arlitt, M., Gburzynski, P., Gomes, F., Gurski, R., Ono-Tesfaye, T., Ramaswamy, S., Williamson, C. and Xiao, Z. 1994c. "ATM-TN System Design", WurcNet Technical Report, September, 142 pages.

Unger, B.W. and Cleary, J.G. 1993. "Practical Parallel Discrete Event Simulation" ORSA Journal on Computing,, 5(3), 242-244.

Unger, B.W., Cleary, J. G., Dewar, A. and Xiao, Z. 1990. "A Multi-Lingual Optimistic Distributed Simulator", Transactions of the Society for Computer Simulation, 7 (2), 121-152, June.

Vickers, B.J., Kim, J.B., Suda, T. and Hong, D.P. 1993. "Congestion Control and Resource Management in Diverse ATM Environments", IEICE Transactions on Communications, November.

Waldorf, J. and Bagrodia, R. 1994. "MOOSE: A Concurrent Object-Oriented Language for Simulation", International Journal in Computer Simulation, 4(2), 235--257.

West, J. and Mullarney, A. 1988. "ModSim: A Language for Distributed Simulation", Proceedings of the 1988 SCS Multiconference on Distributed Simulation, (eds Unger B. and Fujimoto, R.), San Diego, California, 19(3), 155--159 February.

Appendix A : Cell Level Measurements of ATM-25 Traffic

Ian Graham and John G. Cleary

Department of Computer Science,

University of Waikato,

Hamilton, New Zealand.

e-mail: {ian,jcleary}@cs.waikato.ac.nz

Abstract

A system for capturing complete cell level header and timing information for bi-directional ATM-25 links is described. The system is able to measure and synchronise multiple links simultaneously. Off the shelf ATM-25 NIC equipment and standard PCs are used giving a very low cost system. Headers for all cells are captured to disk and timing is accurate to sub-microsecond times. Measurements of video, file transfer traffic and artificially generated traffic are reported. Future extensions of the system to higher bit rates and optical links are briefly discussed.

1. Introduction

ATM is a new cell based network standard [ATM, 1993] that is gaining wide commercial acceptance. ATM has been designed to provide acceptable performance over a wide range of different applications including voice, video and data traffic. From a commercial point of view ATM has the advantage that it is capable of supporting many different physical layers operating at speeds from 25 Mbits/sec to over 2 Gbits/sec and over distances from a few meters to intercontinental and satellite links. Of particular interest here is the hope that it can provide efficient service by multiplexing many users and different types of traffic over the same links. Because the basic unit of transmission is a 53-byte cell it is possible that users with time varying transmission requirements can use links whose capacity is less than the total peak requirements of the different users.

However, the behaviour of networks such as ATM can be complex and any performance claims need to be approached with caution. The actual delivered performance of ATM depends on many factors besides the raw efficiency of cell transmission. For example, recent measurements of high volume data and video traffic using TCP/IP has shown it to be fractal or self-similar in nature [Leland *et al*, 1994]. That is, there are similar fluctuations in traffic at all time scales. Such self-similar traffic places extreme demands on ATM networks and may cause unexpected and complete failure of the network as seen by the user. For example, recent experiments have shown zero throughput of TCP/IP traffic unless care is taken to shape the traffic [Cáceres 1991, Manthorpe 1995].

Significant efforts are under way to provide policing and traffic management standards for ATM [Woodruff and Kositpaiboon, 1990; Hong and Suda, 1991; Jain, 1990]. These are intended to improve the efficiency and robustness of the overall network but they add their own complexity and possibilities for more subtle forms of failure. In developing such standards and protocols it is imperative to know how they will behave in real situations and that they will indeed improve overall performance.

Clearly it is important to measure the performance of actual networks. But on its own this is unsatisfactory for a number of reasons. First, measuring actual performance is time consuming and often requires expensive equipment. Second it is unable to say very much about how a network will behave in different cases such as: higher traffic volumes; changed network configurations; or when failures occur. Another possibility is to use analytic models of network performance. Such analytic models can be very valuable as they may allow predictions of performance over a wide range of scenarios. However, deriving such analytic models and approximations can be very difficult and so far there has been little experience in building models for important patterns such as self-similar traffic [Frost and Melamed, 1994; Garrett and Willinger, 1994]. Development of analytic models requires extensive effort to validate the models against actual measured data.

Another approach to understanding and predicting performance is to use detailed simulations of the traffic. Such simulations can also be used to validate analytic models when measured traffic is not available. Recently, models which

provide detailed simulations of individual cells have become available [Chen *et al*, 1995; Arlitt and Williamson, 1995; Arlitt *et al*, 1995]. Simulators that can execute such models at rates of up to 10^5 cell events per second on a single processor and perhaps ten times that rate on multi-processors are also available [Unger *et al*, 1995; Cleary and Tsai, 1995]. Clearly these simulations can be used to probe large networks over significant periods of time. However, developing such models is difficult as they must take into account a host of different factors at many different time scales including both hardware and software interactions. Traffic may depend on: daily variations in bulk traffic; session level variations caused by user interaction and activation of software; operating system interfaces and activity; the operation of individual communication protocols; the performance of switches; and the speed of links. In some cases detailed modelling of such activity might require the emulation of a complete operating system which is clearly infeasible. An alternative is to provide aggregate and parameterised models which approximate the actual behaviour. In any case, it is vital for the success of such models that they are validated against real traffic. This may be to set parameters and details of the behaviour in exact models or to determine bulk parameters for aggregate models. In some cases it may be impossible to model a system without doing measurements either because the total system is too complex to be understood in detail or because of commercial secrecy surrounding vendor specific implementations.

The conclusion from this, is that measuring the detailed behaviour of real traffic over real ATM networks is important, both for the sake of understanding the scenarios themselves as well as to validate simulation and analytic models. It is not necessarily clear what it is that should be measured. Many current switches provide bulk statistics such as the total number of cells transmitted. Such statistics are insufficient for detailed understanding of important parameters. For example, ATM cells can be discarded by switches in the event of buffer overflow or internal switch congestion. The rate at which cells are discarded is a critical parameter which can depend in a very detailed way on the timing of the arrival of cells and auto-correlations of the arrival times of the cells. For example, many traffic types such as TCP/IP and video traffic generate bursts of cells at the peak line rate interspersed with quiet periods (measurements later in this paper demonstrate such behaviour). Bulk statistics are totally unable to see such effects and their use is likely to grossly underestimate cell loss rates. Thus any measurements need to be at the level of individual cells and timing information needs to be accurate to cell transmission times or less.

It is also important to be able to measure a network at multiple points simultaneously. One important parameter in any model is the latency through a switch. To measure this reasonably directly it is necessary to be able to measure cells entering and leaving the switch and to be able to correlate incoming and outgoing cells. This requires that the timing of cell measurements are not only relatively accurate within one cell stream but are also accurate to less than one cell time between different measurement points. It is also necessary to collect sufficient information about each cell payload to make it possible to identify corresponding incoming and outgoing cells.

The importance of self-similar effects in traffic implies that large amounts of data need to be collected so that effects and fluctuations that appear on the order of minutes or hours can be investigated. The only feasible way that this can be done is to collect the data directly to disk or other bulk storage.

It is essential that any measurements that are done do not interfere with the actual traffic being measured. This makes it very difficult to use switches themselves as the measurement point, as at the very least the detailed data will consume bandwidth on internal switch buses or communication links. It is also important for our project, which is taking place within a university research project, that any hardware for measurement be cheap. Most commercial measurement equipment is completely out of the range of what we can afford. One advantage of doing measurements that are purely for the purpose of measuring performance and traffic characteristics is that it is possible to assume that the underlying physical links are intact. The cost of most commercial communications measuring equipment is greatly increased by the fact that it is intended to detect and diagnose faults including hardware and low level link failures.

In summary then we have argued for the utility of an ATM traffic measuring tool which:

- records timing and other information at the cell level;
- does not interfere with the measured traffic;
- measures cell streams at multiple points in a network simultaneously;
- is accurate to fractions of a cell time within one cell stream and between cell streams;
- captures information directly to bulk storage;
- is low cost.

The rest of the paper considers a system which we have developed for measuring ATM traffic at 25 Mbits/sec. The entire system uses low-cost off the shelf NICs (at less than US$500 each) and personal computers. The next section gives details of the construction of the system and how measurements were carried out. Section 3 gives details of the results of measurements done on video and data traffic over a LAN interconnecting PCs and a video server. Section 4 concludes with a summary and some consideration of future work that may be possible.

2. Measurement System

Fig. 1 shows the layout of the measurement system which is based on an ATM Ltd Virata Link network interface card (NIC). This card interfaces 25 Mbits/sec ATM over unscreened twisted pair (UTP) cable to the ISA bus of an IBM-compatible PC. The card has been reprogrammed by the authors to record accurate ATM cell arrival times and the corresponding cell header, but to discard the cell payload. Optionally, the system can record a cyclic redundancy check

(CRC) value for the cell payload, this enables cells to be matched in different parts of the network.

Each ATM cell consists of 53 bytes: 4 bytes of header; a CRC byte; and 48 bytes of payload. Cell transmission is continuous, unassigned (idle) cells being inserted if no data cell is available so that receiver and transmitter remain in synchronisation. At the effective data rate of 25.6 Mbits/sec [ATM25, 1995], each cell takes 16.9 μsec to transmit.

This, then is the speed at which the measurement device must be able to handle cells, and the resolution of the device must be less than a cell time so that adjacent cells can easily be separated. For each cell we record its arrival time and the header. To measure the arrival time we use a 32-bit counter located on the NIC, and therefore must transfer 8 bytes to the PC for each cell: 4 bytes of header; and 4 bytes of time. This gives a maximum output data rate for the device of 463 Kbytes per second. This is well within the capabilities of the PC ISA bus.

In our application the ATM link level code that normally runs in the NIC is replaced by a simple program which waits until an incoming cell enters the receiver interface. It then reads the 4 MHz timer, and copies this value with the cell header to the ISA bus. The rest of the cell is discarded and the process is repeated for the next cell. A data collection program running in the PC copies the output from the ISA bus onto disk, using main memory as a buffer. Timing tests have shown that the header can be extracted from an incoming cell, copied to the ISA bus interface with the counter value, and the rest of the cell discarded, in just under 4 μsec. Data can be transferred on the ISA bus from the NIC to PC memory at just over 1 Mbytes/sec. Both these results are well within the rates required by the system.

However, these measurements must be very closely synchronised to be useful. The crystal oscillators on the NICs typically have a frequency tolerance of 5 ppm, and are subject to drift due to ambient temperature changes. Although 5 ppm is quite small, it corresponds to one cell time (16.9 μsec) in just over 3 seconds. Thus some form of clock synchronisation must be used if true comparative measurements are to be made. We have used the ARM 60 fast interrupt request (FIQ) signal for clock synchronisation. The FIQ is a high priority interrupt that requires little state to be saved, and has a worst case latency, in our program, of less than 0.5 μsec. Typically the latency will be less than this, with a minimum of about 0.25 μsec. The clock synchronisation system uses a single master processor and any number of slave systems. A counter on the master NIC produces an FIQ four times per second, this signal is buffered and transmitted to the slaves. On the master processor the interrupt is serviced, and a special record written to the ISA bus consisting of the clock reading and a code that can be distinguished from any valid cell header. In the slaves the interrupt is latched, and a similar record is written before the interrupt is cleared

When data are analysed these time marker records are compared for the master and slave processors. A moving window linear regression is used to model the drift of each slave against the master, and to correct the time values measured by the slaves. This technique was tested by making two simultaneous

observations on the same link. Over 90% of the measurements were within 0.5 μsec of each other and none disagreed by more than 1.0 μsec.

When measuring some parameters, such as the delay through a switch or cell loss statistics, it is necessary to uniquely identify cells by their payload, even though the header changes when the cell passes through the switch. Rather than record and compare complete payloads it is sufficient to calculate a CRC on the cell payload and to use this as a cell identifier. This CRC is optionally calculated by the data capture program in the NIC, and can be recorded with the cell arrival time and the header.

Although 'real' traffic is useful for some measurements it is unpredictable, and our applications have not managed to develop network loadings of more than about ten per cent. In order to generate repeatable traffic conditions, in particular higher network loadings and long bursts of cells, we have programmed NIC cards as traffic generators. The traffic generator used in these tests can generate contiguous cell bursts of any length, separated by a programmable time interval.

Test set-up

Our initial experiments have been concerned with measuring traffic on a network consisting of PCs and an ATM Ltd Virata Store connected by an ATM Ltd Virata Switch (see Fig. 2). The Virata Store is an 8 GByte RAID array, interfaced to an ATM 155 Multi-mode fibre interface, and is supplied with file sharing software compatible with Windows for Workgroups. The PCs are interfaced to the 25 Mbits/sec UTP ATM links with ATM Ltd Virata network interface cards.

The Virata Switch is connected to the Store at 155 Mbits/sec and to a number of PCs using Windows for Workgroups file sharing over the TCP/IP protocol. The results reported below were obtained by tapping the ATM-25 lines between the switch and the PCs, using resistive matching networks. In a typical experiment MPEG-1 compressed video data stored on a PC or the Virata Store is displayed on another PC using an MPEG decompression card.

3. Results

To illustrate the kinds of data that can be captured with the system we show here two examples — traffic measurement on compressed video (MPEG-1) data, and cell delays through a lightly-loaded switch, with both 'real' and artificially generated traffic. MPEG-1 requires a data rate of about 1.5 Mbits/secs, or 183 Kbytes/sec, well within the theoretical capacity of an ATM-25 link. However, our measurements show that using the TCP/IP protocol results in a very bursty use of the link, with the full capacity being used for short periods followed by long periods of inactivity. In Fig. 3 we show measurements of cell arrival times at the display PC for MPEG video, binning the arrival times into 10 msec intervals. The data arrive in short bursts, of about 20 msec, separated by silences of 50 msec or more. In each burst the traffic approaches 50 per cent of the theoretical maximum throughput of an ATM-25 link. Looking more closely at the data, using a binning interval of 100 μsec, shows that even these short bursts have an internal structure. Each burst observed at 10 msec resolves into ten sub-bursts that completely saturate the link for periods of about 1 msec. Each sub-

burst contains up to 64 cells, transmitted at full cell rate. This has obvious consequences for the design of buffering schemes for ATM interfaces in both switches and terminal equipment.

An important characteristic of any ATM switch is the delay that it introduces into the network. With our measurement system it is possible to characterise the switch delay under realistic loading conditions by observing cells as they enter and leave the switch, corresponding cells being recognised by the CRC of their payload. Of course, the accuracy of this type of measurement depends on the close synchronisation of the clocks at the two measurement points. Figure 5 shows a histogram of cell delays for the Virata switch measured under the load of one MPEG circuit. This delay will depend on a complex interaction of the design of the switch and applications software. In this diagram the Y-axis has been greatly expanded, to show the very small percentage of cells that are delayed more than about 20 μsec. In fact 95 per cent of cells are delayed less than 25 μsec. However, there is a long tail, and delays of up to 85 μsec are experienced by a very few cells.

A clearer picture of the performance of the switch is obtained by using artificial traffic. Figure 6 shows the cell delay distributions obtained by subjecting the Virata switch to cell bursts of from 1 to 1,000 cells into one 25 Mbps port. The bursts are separated by nine times their width, giving an average network loading of ten per cent. As the burst length increases so does the average delay of cells, and the width of the delay distribution. To investigate this further we plotted the delay for each cell in a burst against its position in the burst, a plot for five typical bursts is shown in Figure 7. The delay increases linearly with the position in the burst, it appears that the switch is not quite fast enough to deal with contiguous cells. During some bursts there is a jump in the delay time, usually around the 100 cell point. Presumably this latter effect is due to other activity within this single-bus switch. When there is no jump cell delay within a burst can be very accurately fitted by a linear function:

$$\text{delay}(n) = 22 + 0.042\, n \ \mu\text{sec}$$

This type of information starts to make possible the accurate validation of switch models.

4. Future Directions

In this paper we have described a low cost system for measuring ATM-25 traffic. It is capable of collecting sufficient volume of accurate data to make it useful in calibrating and verifying detailed simulation and analytic models of ATM traffic.

Current major restrictions, of the system are that:

- it is capable of measuring only the lowest speed standard ATM link (ATM-25);
- data is not being captured to disk;
- only the CRC for the payload, rather than the complete payload is being captured.

These restrictions arise from two limitations. The first is the CPU time required to capture and process each incoming cell. At the moment we are taking less than 4 μsecs per cell using an ARM60 RISC processor clocked at 16 MHz [GEC Plessey, 1992]. This is tantalisingly close to the 2.7 μsecs required for a cell arrival at 155 Mbits/sec , which it should be possible to achieve with a slightly faster processor. Links at 34 Mbits/sec and 45 Mbits/sec could be accommodated. The second limitation is the bus bandwidth of the personal computer being used to collect the data. The ISA bus being use on our current PCs has a total measured bandwidth of about 1 M byte/sec, and we are currently upgrading to PCI based buses which are capable of over 100 Mbytes /sec, well within the requirements to capture full cell payloads at 155 Mbits/sec.

Acknowledgments

This work has been supported by a grant from New Zealand Telecom Ltd. and by the New Zealand Public Good Science Fund. We would like to thank ATM Ltd. for providing detailed specifications of their Virata NIC, and for other help with the project.

References

Arlitt, M., and Williamson, C.(1995) "*A Synthetic Workload Model for Internet Mosaic Traffic*," Proc. Summer Computer Simulation Conf., Ottawa, pp. 852-857, June.

Arlitt, M., Chen, Y., Gurski, R., and Williamson, C.(1995) "*Traffic Modelling in the ATM-TN Telesim Project*," Proc. Summer Computer Simulation Conf., Ottawa, June.

ATM Forum (1993) *ATM UNI Specification, Version 3.0*, Interop Inc., Mountain View Ca..

ATM25 (1995) *ATM Physical Layer Specification* . ATM25 Desktop Alliance, http://www.atm25.com/atm25/

Cáceres, R., Danzig, P.B., Jamin, S., and Mitzel, D.J.(1991) "*Characteristics of Wide-Area TCP/IP Conversations*," ACM SIGCOMM, **21**(4).

Chen, Y., Deng, Z., and Williamson, C. L.(1995) "*A Model for Self-Similar Ethernet LAN Traffic: Design, Implementation, and Performance Implications*," Proc. Summer Computer Simulation Conf., Ottawa, pp. 831-837, June.

Cleary, J. G., and Tsai, J-J.(1995) "*Conservative Parallel Simulation of ATM Networks*". Research Report, Dept. Computer Science, University of Waikato, October.

Frost, V., and Melamed, B.(1994) "*Traffic Modeling for Telecommunications Networks*," IEEE Communications, **32**(3), pp. 70-81, March.

Garrett, M., and Willinger, W.(1994) "*Analysis, Modeling and Generation of Self-Similar VBR Video Traffic*," Proc. ACM SIGCOMM'94, London, August.

GEC Plessey (1992) *ARM Product Family: ARM60 Microprocessor*, GEC Plessey, Swindon, UK.

Hong, D., and Suda, T.(1991) *"Congestion Control and Prevention in ATM Networks,"* IEEE Network, **5**(4), pp. 10-15, July.

Jain, R.(1990) *"Congestion Control in Computer Networks: Issues and Trends,"* IEEE Network, **4**(3), pp. 24-30, May.

Leland, W., Taqqu, M., Willinger, W., and Wilson, D.(1994) *"On the Self-similar Nature of Ethernet Traffic,"* IEEE/ACM Trans. on Networking, **2**(1), pp. 1-15, February.

Manthorpe, S.(1995) *"Buffer and Bandwidth Allocation for TCP Data Traffic: Experimental Results,"* First Int. ATM Traffic Expert Symposium, Basel, Switzerland, April.

Unger, B.W., Gomes, F., Zhonge, X., Gburzynski, P., Ono-Tesfaye, T., Ramaswamy, S., Williamson, C., and Covington, A.(1995) *"A High Fidelity ATM Traffic and Network Simulator,"* Winter Simulation Conference, Washington, D.C., December.

Woodruff G., and Kositpaiboon R.(1990) *"Multimedia Traffic Management Principles for Guaranteed ATM Network Performance,"* IEEE J. Selected Areas in Comm., **8**(3), pp. 437-446, April.

Figure 1. ATM Traffic Measurement System.

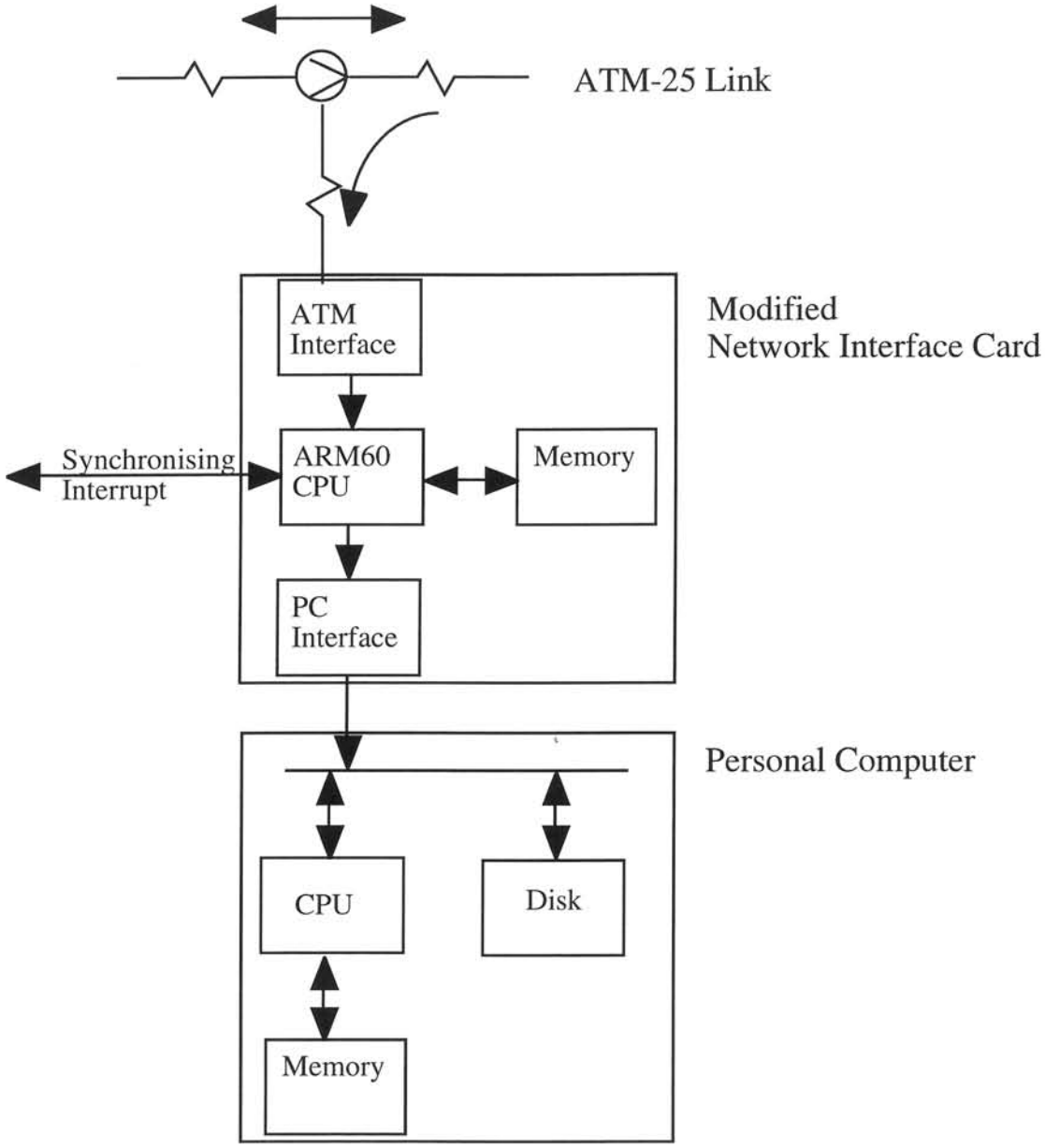


Figure 2. Measurement Configuration.

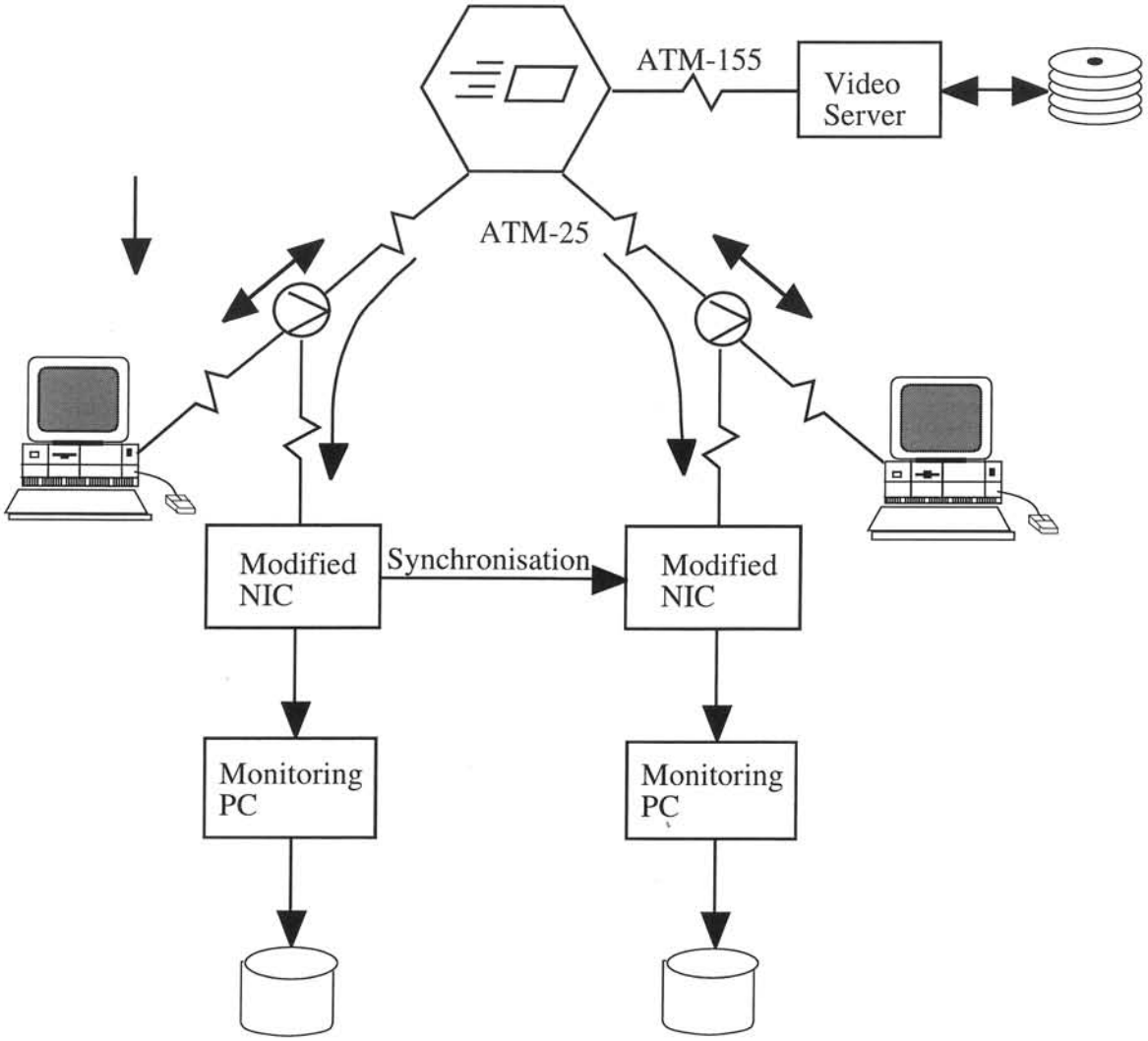


Figure 3 MPEG-1 Traffic at 10 msec Resolution.

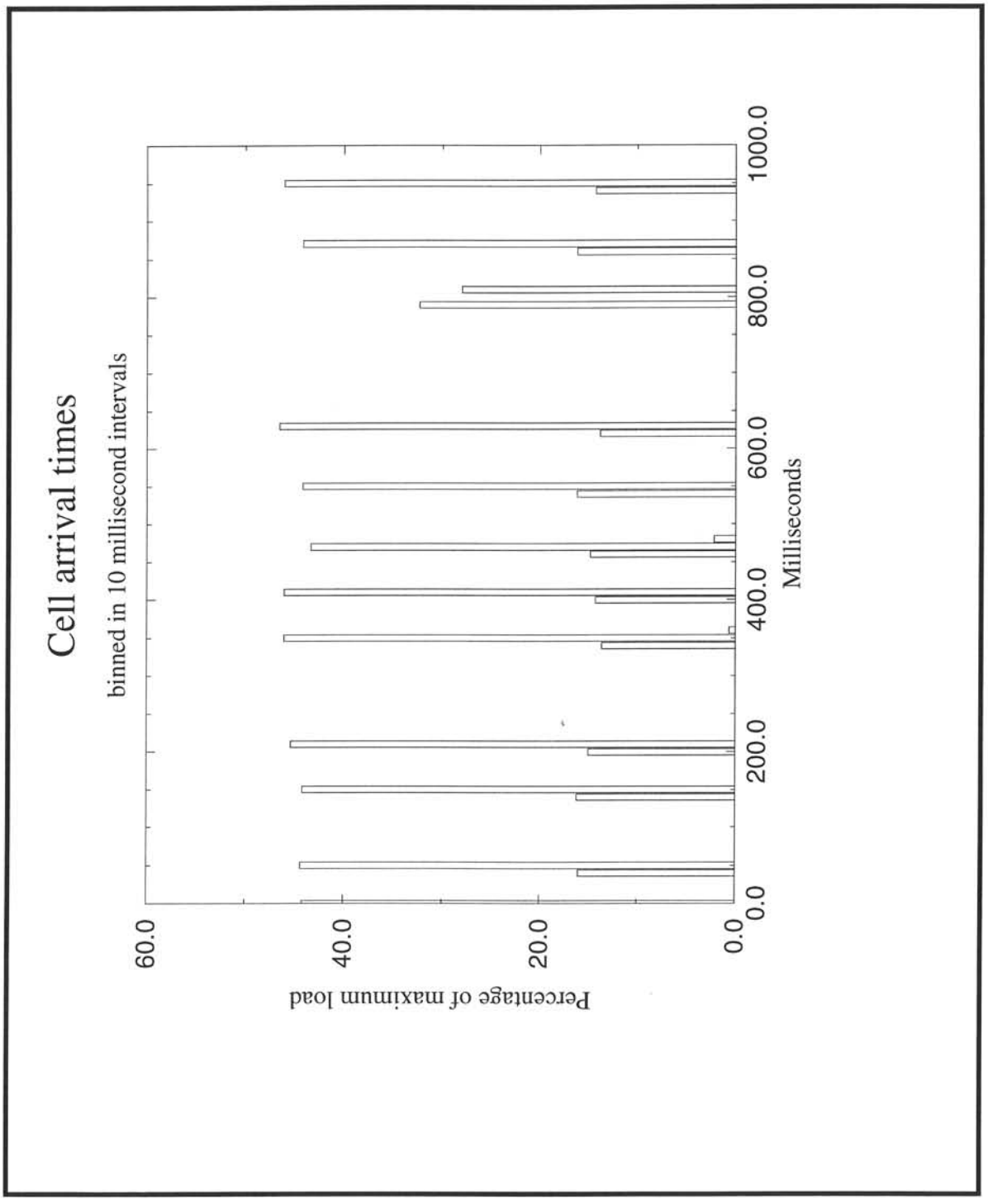


Figure 4 MPEG-1 ATM Traffic at 100 μsec Resolution.

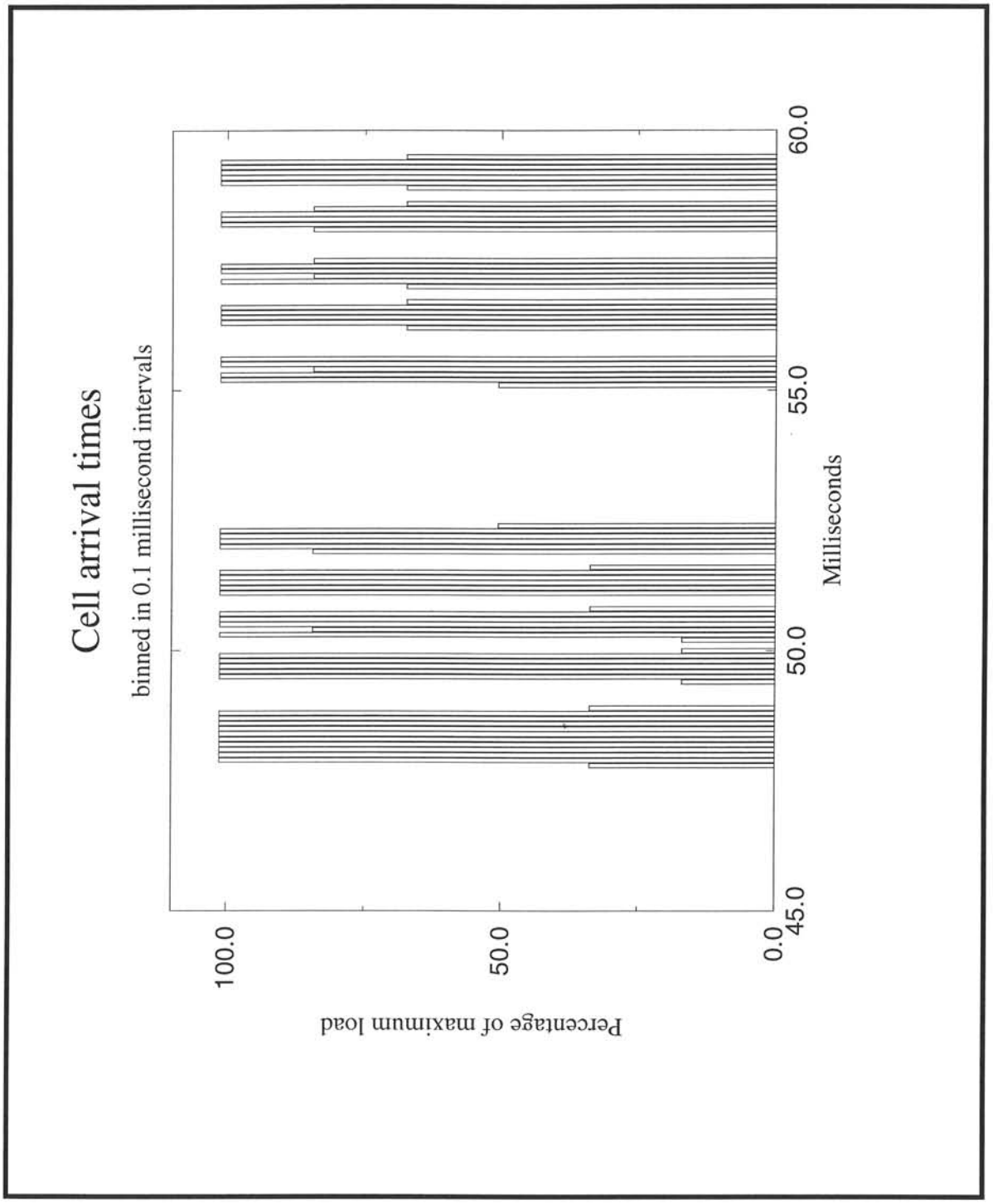


Figure 5 Measured Switch Delays.

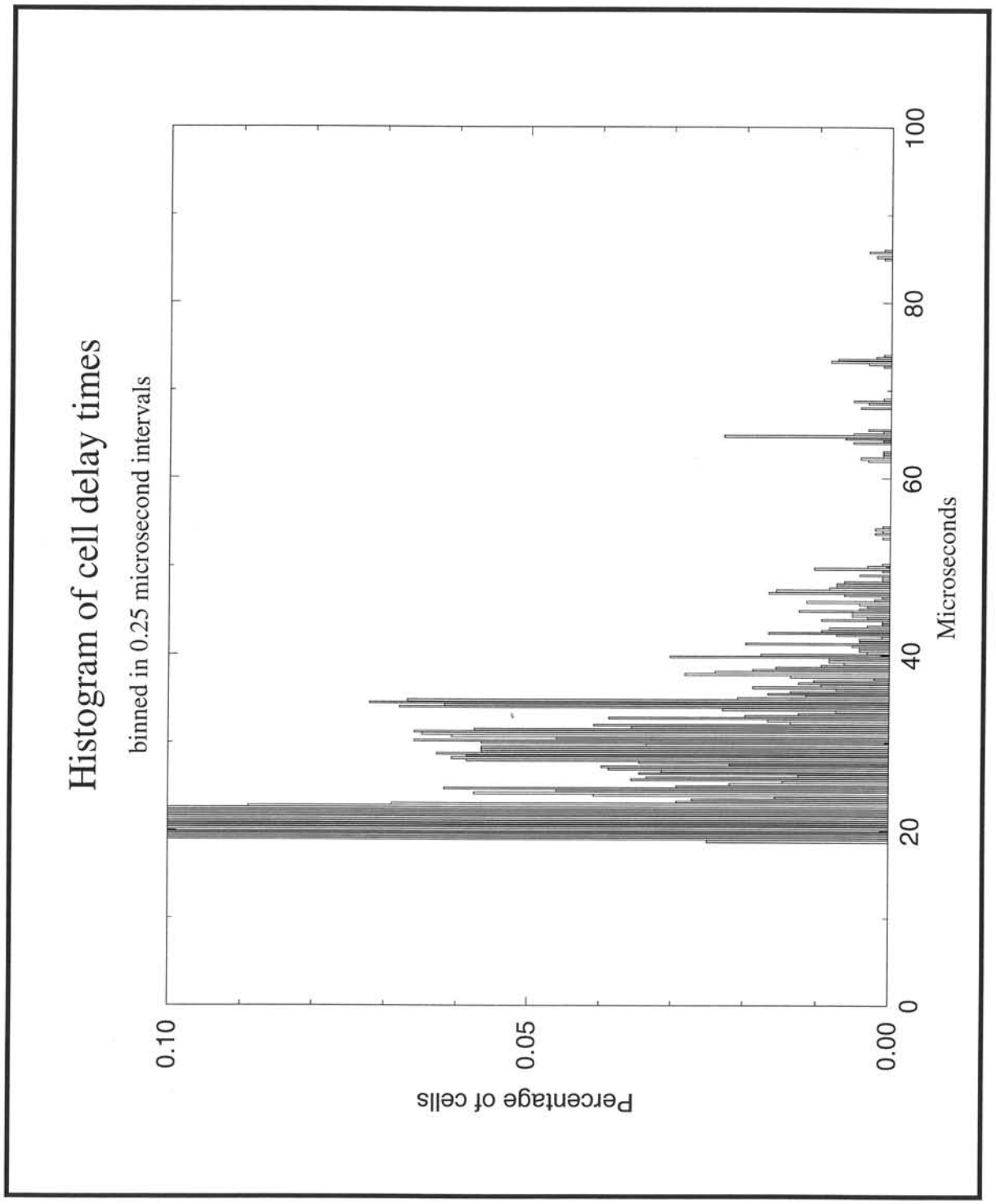


Figure 6 Cell delay distributions for bursty traffic.

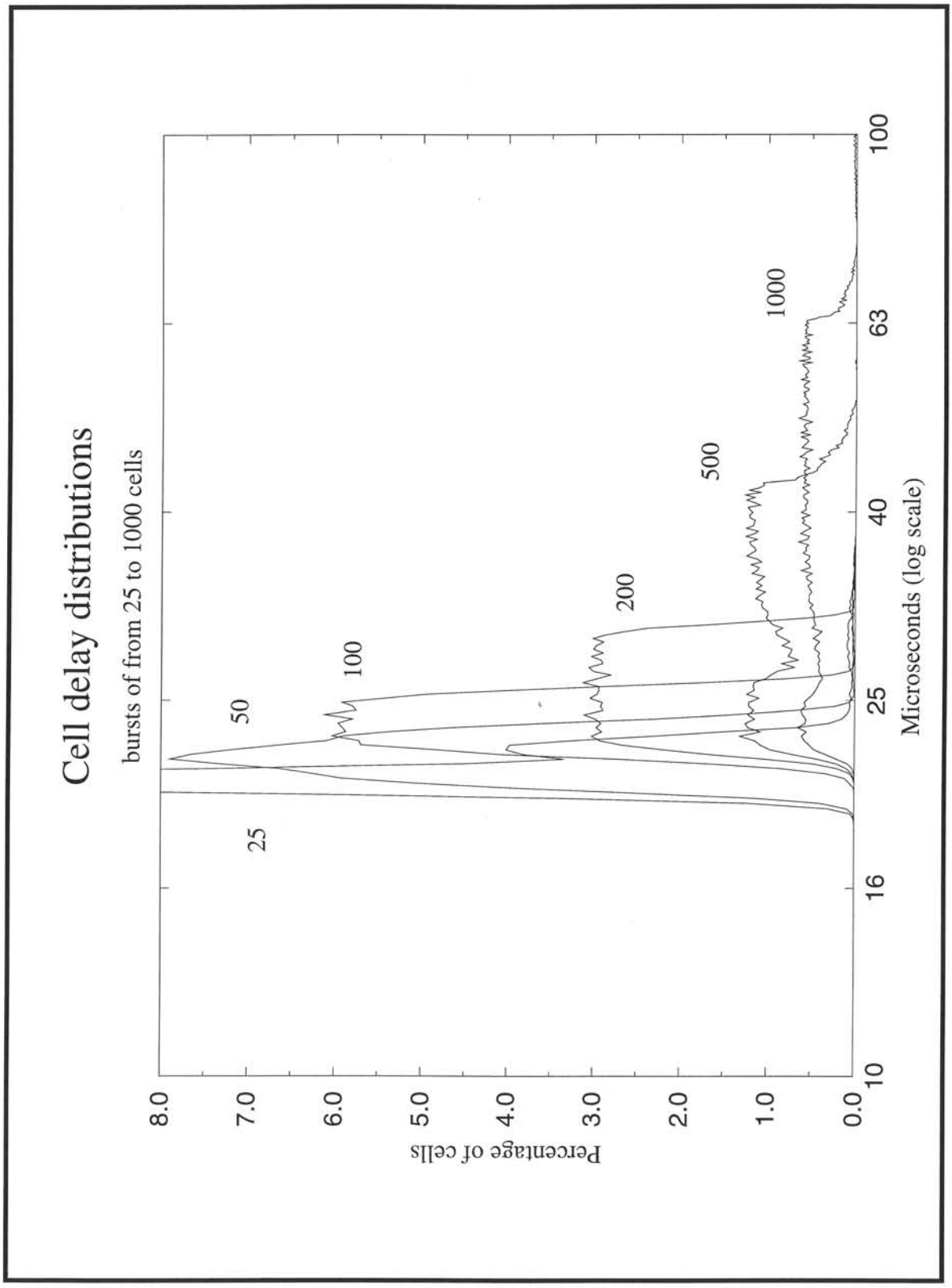
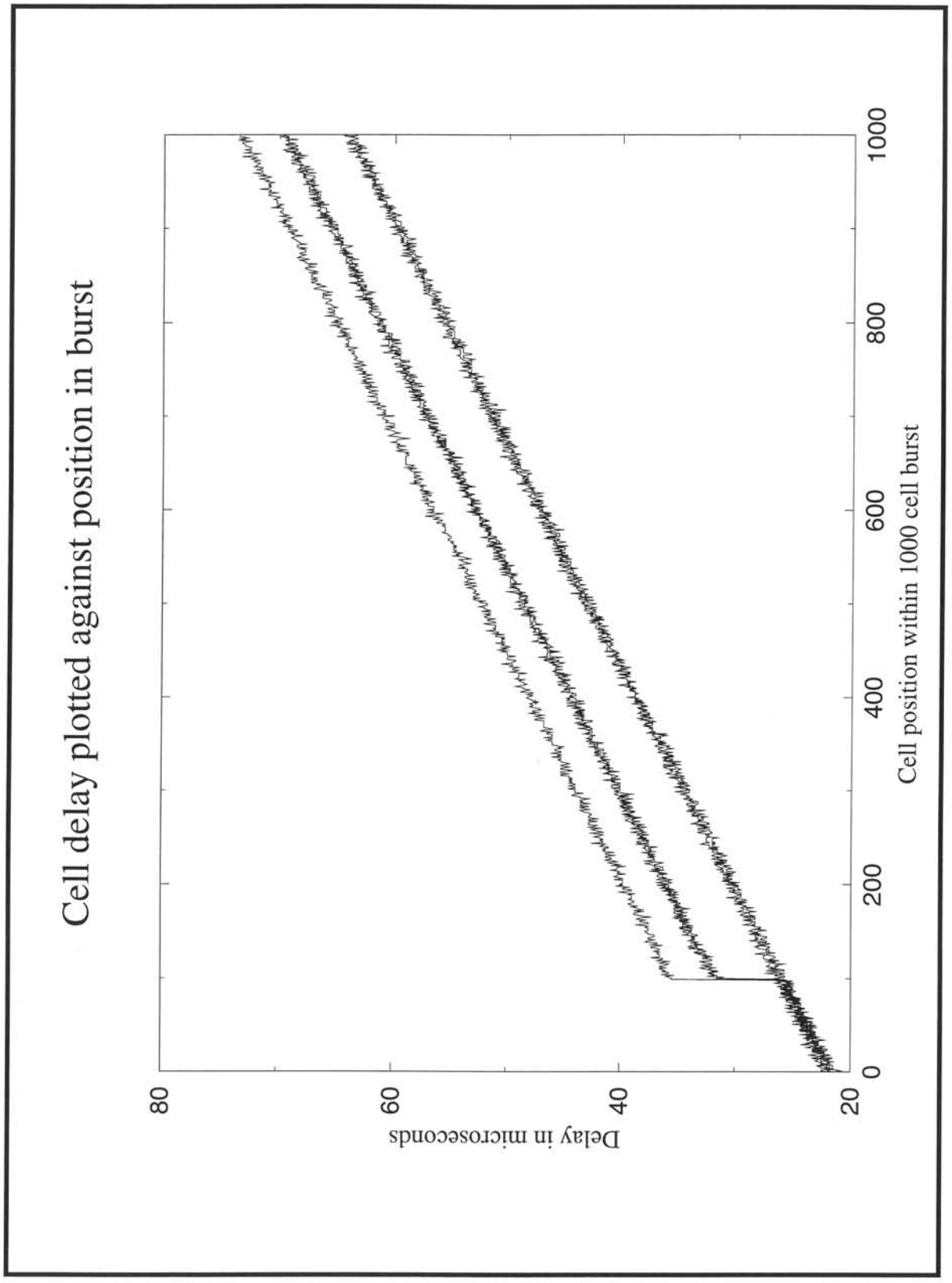


Figure 7 Cell delay variation duting 1,000 cell bursts.



Appendix B: Conservative Parallel Simulation of ATM Networks

Department of Computer Science, University of Waikato, New Zealand.
email: {jcleary,jjtsai}@cs.waikato.ac.nz

Abstract

A new conservative algorithm for both parallel and sequential simulation of networks is described. The technique is motivated by the construction of a high performance simulator for ATM networks. It permits very fast execution of models of ATM systems, both sequentially and in parallel. A simple analysis of the performance of the system is made. Initial performance results from parallel and sequential implementations are presented and compared with comparable results from an optimistic TimeWarp based simulator. It is shown that the conservative simulator performs well when the "density" of messages in the simulated system is high, a condition which is likely to hold in many interesting ATM scenarios.

Introduction

ATM is a recent standard for broadband communications [3,7]. It is a rate independent system which uses fixed size (53 byte) *cells* for all transmission. The motivation of this work is the need to simulate large numbers of cells in ATM systems. Such simulations, which need to explicitly model the passage of cells through the fabric of switches and over the physical communications links, are particularly demanding for a number of reasons. There are indications that it is necessary to simulate in the order of 10^9 to 10^{12} cells in one run to measure some effects such as cell loss rates. Also the models of ATM at the individual cell level have very low compute grains. For example ATM-TN a detailed ATM model produced by the Telesim project [15] has average per simulation event granularities of 17 μseconds on a Sparc-2 55MHz CPU [11]. These granularities are close to the per event overheads that can be expected from sequential simulators. For example, the sequential simulator used to obtain the granularities has a measured per event overhead in the range of 11 to 18 microseconds (depending on the size of the event list).

The need for large numbers of high fidelity simulation events motivates the requirements for a high performance simulator. The low granularities imply that any simulator must be very efficient with per event overheads close to those of the best sequential simulators. Results reported later will show that expected simulation scenarios have significant available parallelism. Thus there is a strong motivation to construct parallel simulators to take advantage of this. The difficulty here is to keep the overheads low enough that the parallelism can be effectively used. To this end the conservative simulator is designed for a shared memory multiprocessor environment. This follows the lead of recent TimeWarp parallel simulators [8].

Any such simulation splits into two main parts: the models of the ATM switches, access points and links; and the models of the traffic generators. This paper is concerned mainly with the interior ATM objects and not so much with the traffic generators. There are two reasons for this: the ATM part is by far the most compute intensive; and it is more stereotyped and consistent than the generators.

ATM-TN is a generic cell-level ATM model [15]. The model deals with individual cells and passage through switches. As well it includes high-level models of TCP/IP, World-Wide-Web traffic, and self-similar traffic [1,2]. The current work is based around this model.

The proposed algorithm belongs to the class of conservative simulations. These, by definition, block until a process can ensure that it will not violate causality by processing the next event [10]. Nil messages, as proposed by Chandy and Misra [5,13], provide a method for communicating processes to exchange information regarding the lower bound on the time stamps of future messages. The effectiveness of nil messages depends greatly on the amount of lookahead available [9]. This is apparently application dependent [14,16]. In any case, the possible imposition of extra overhead by the use of nil messages has caused much criticism of the usefulness of such approaches.

Several mechanisms based on nil messages have been developed since it was first introduced. Chandy and Sherman [6] present an alternative. Instead of nil messages, a conditional message is sent to the receiving process only when the sender will otherwise become blocked. As long as there are (real) messages waiting in the receiver's input buffer, conditional events are not used. Also, Jha and Bagrodia [12] suggested a scheme combining the above approach and conventional nil messages. Cai and Turner [4] proposed

a "carrier null message" approach. In this approach, an additional field is included to the nil message so that a process can identify a nil message initiated by itself. When such a message is detected, the blocking is ended. The effectiveness of this approach depends on how much earlier such a carrier can be detected compared to regular nil messages. Although performance speedup can be observed in certain cases with these approaches, the applicability of these to large complicated communication networks (eg ATM) remains unknown.

In the next section the simulation algorithm itself is described, together with both sequential and parallel implementations. Section 3 focuses on a key data structure and its optimisation. Section 4 considers the theoretical performance of the conservative simulator and compares this with event based simulators including TimeWarp. As well actual performance results from a toy problem on a distributed version of the conservative simulator are reported together with comparable results from a TimeWarp based simulator. Initial results from a full port of ATM-TN are also provided. The paper concludes with a summary.

Simulator

In the simulator all *processes* receive (0 or more) *input links* and generate (0 or more) *output links*. *Messages* are sent down links between processes. Each message has a *send* and *receive* time. The critical part of the proposed algorithm is that each time a process is executed it merges all its input lists and executes one *event* for each incoming message, this continues until one of the input links is empty where-upon the process *suspends*. Because the merge is done in time order the process suspends on the earliest empty link, the earliest time that a message can be received down that link is the process's *current time* (CT).

The generic algorithm is:

```
while there are messages in system
   select next process to execute;
   while lowest time stamped
    incoming link has a message
    select lowest message;
    process message;
   end while;
   set current time of process to
    minimum time on any in-link;
   update last time on all outgoing
    links to minimum possible
    receive time of next message
(will be ≥ current time +
lookahead of link);
   suspend process;
end while;
```

This algorithm is different from many event driven simulators, in that, when a process is selected for execution it consumes all of its incoming messages before suspending. That is, scheduling is not done on an event by event basis. In the worst case no event will be executed when a process is selected, although the current time of the process may be advanced which will cause the time on the output links to also advance. Such an "empty" execution corresponds roughly to a nil message in a Chandy-Misra conservative simulator [5] and to the conditional messages used in [12]. Many algorithms are possible for selecting the next process to be executed. The aim is to minimise the cost of scheduling both by minimising the number of suspension / scheduling steps and by minimising the cost of suspension and scheduling.

It is assumed that the links are monotonic - that is messages are received in the order of their receive times which is the same order that they are sent (physically this can be interpreted as "messages cannot pass each other in the links"). Of concern is the *lookahead* of each link which is the minimum difference between the receive and send time of a message. This must be positive (it may be 0 in

some cases), and in many interesting cases will be non-zero.

Consider a simulation model as a directed graph where the links are arcs and the processes are nodes. The arcs are labelled with the lookahead of the link. For any loop (sequence of nodes and arcs that arrives back where it starts) the *loop time* (LT) is the sum of the lookaheads along the loop. The *Minimum Lookahead Time* (MLT) for a link or process is the minimum LT of any loop which passes through the link or process. I will assume that no process (and by implication link) has a zero MLT. In particular all ATM models will be seen to have non-zero MLTs.

The algorithm above can be executed by randomly selecting a process, executing it, and then selecting the next process and so on. So long as this process is fair - that is every process is eventually selected for execution - then it is easily shown that the simulator will make progress (the GVT or minimum time of any unprocessed message in the system will increase). The problem is to minimise the overheads of suspension, that is, the aim is to maximise the number of events executed and minimise the number of suspensions.

Sequential Execution

The first (sequential) algorithm always schedules (one of) the processes with the smallest CT. In such a simulator the best that any process can do is to advance its simulation time by its MLT on each execution/suspension cycle[1]. If the MLT of the i'th process is given by MLT_i then a lower bound for the average number of suspensions per simulation time unit, C, will be:

$$C \geq \sum_i \frac{1}{MLT_i} \qquad (2.1)$$

The process of scheduling on the basis of the CT is susceptible to many optimisations. Given that the processes can be scheduled at random, clearly scheduling on the basis of the CTs can be sloppy and still the algorithm will work. Note that selecting for execution on the basis of CT is different from using an event list (in particular, the times used for scheduling may correspond to no actual events in the system and the number of entries in the scheduling list is always equal to the number of processes in the system).

Static Schedules

Another possible algorithm is to have a loop in which each process is executed exactly once. This will clearly be sub optimal if the processes have differing MLTs - those with large MLTs will be executed too often. However, it seems that static schedules in which a process appears in inverse proportion to its MLT can approach close to the lower bound. The example systems in Figure 1 have close to optimum static schedules. In Fig 1a the MLTs for the three processes are: A=1, B=2, C=3. From formula (2.1) the lower bound for the number of suspensions per unit of simulation time is C=1.83.. Using either of the static schedules below achieves C=2..:

C B A A C B A A ..

or

C B A A B A C B A A B A ..

For Fig 1b the MLTs are: A=1, B=3, C=3 and the optimum of C=1.67.. is actually attained by

C B A A A C B A A A ...

Clearly the overheads of such static schedules can be made very small. It is not necessary to maintain an event list or any dynamic data structure. For small numbers of processes they could

[1]Consider the loop passing through the process that has the minimum LT. Assume that all the incoming links along the loop other than the one under consideration have minimum receive times that are infinite (or sufficiently far in the future that they do not contribute to the minimum). Then the minimum receive time on the incoming link will be equal to the sum of the lookaheads around the loop, that is, the MLT. It is this time that will limit the advance of the process when it is executed.

be directly compiled as an unrolled sequence of procedure calls.

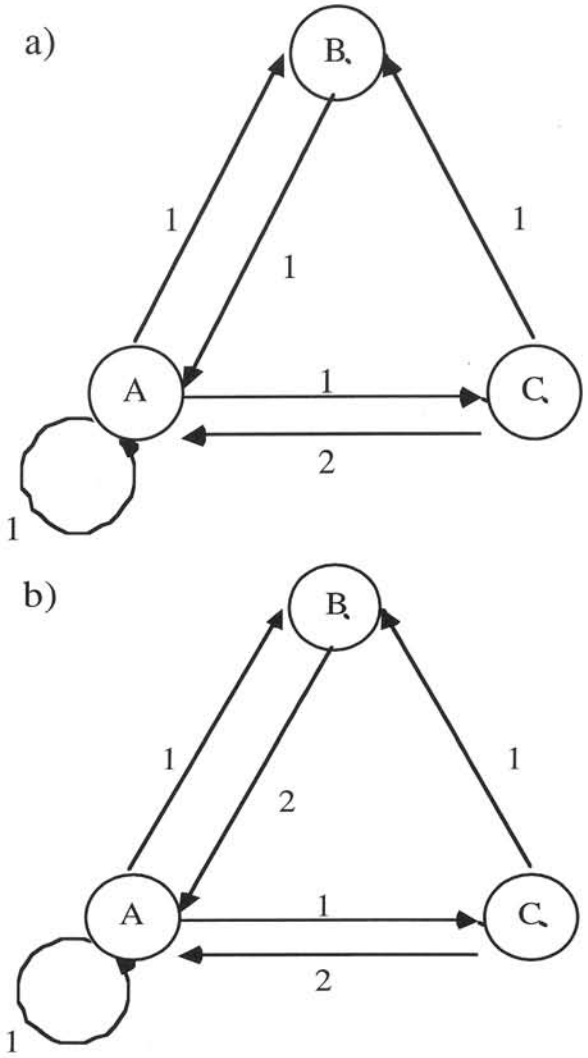


Figure 1. Example Process Graphs.

Parallel Execution

Consider a distributed system where the processes are each mapped to a processor. A static schedule can be followed locally on each processor. In the limit when each process is mapped to one processor the static schedule will be the repeated execution of the same process. In this case the process is effectively polling its input links.

For ATM systems this parallel case (with all processes mapped to their own processor) may be **more** efficient than the sequential one. The critical process, that is the one that on average runs the slowest may never suspend, as the other (lightly loaded) processes will run faster than the critical one and so the critical process will always have messages on all its input links.

Thus the critical part of the computation will be essentially free from any suspension overheads, it will spend all its time executing in the user modelling code.

Link Data-structures

A key data-structure in any simulator based on these ideas is the one that holds the messages in the links. It should allow fast insertion and extraction of events and also be able to be shared between processes in a parallel system. The sharing is made easier by the fact that only two processes ever access the link - the sender and the receiver. The rest of this section consists of a series of refinements of an initial simple data structure. The refinements are designed to give greater efficiency and to provide effects such as bounding the time advance of the sending process. It is notable that the resulting data structure does not need any locking to work correctly, thus avoiding one important source of overheads.

Linked-list

Figure 2 gives a description and pseudo-code for a simple version of the link data structure. It uses single unidirectional pointers between the messages in the link together with two external pointers into the link: Top and Bottom.

This code works correctly even in the case where the sender and receiver are running in parallel. In this case `Top` is modified only by the sender and is read by the receiver. `Bottom` is used only by the receiver. To ensure correct parallel execution the order of assignments in `send` is important, in particular `Top` must be assigned after all the other fields have been set up. As

well storing a pointer or a time-stamp must be an atomic operation.

Optimized

The first optimisation is to make the length of the list in the link constant. This achieves two things: it removes the overheads of allocating and deallocating links and it also allows the length of the list to be used as a flow control mechanism. For example, a process with no input links (a message generator) will have to suspend if its output links become full. Fixing the length allows the `next` fields to be initialised so that the list forms a circular loop and thus `next` need never be updated. There is one major change needed to the code for `send` to ensure that `Top` does not over-run `Bottom` (that is that too many messages are not inserted onto the link). This also means that the sender must be prepared to deal with the `send` function returning an indicator that the message cannot be sent - in this case the sender must suspend. The other code change is to `initialise` which now creates a fixed loop of links. This code works correctly in parallel.

The final optimisation is to put all the links adjacent to one another in a single block of memory. The main advantage of this is that it minimises cache misses when accessing the messages. The only code that needs to be changed is the initialise function (this is left as an exercise for the reader).

Two further refinements have been added to the implementation. First, links from an LP back to itself are treated specially - it is not necessary to delay on them when they are empty. Second, LPs forming a cycle of small LT can be clustered into a single LP. This optimisation has great impact on ATM-TN because a switch in ATM-TN is modeled by a number of LPs, which exchange control signals with small lookaheads forming low MLT loops.

Performance

Comparison With Event Driven Simulators

It is possible to make a simple comparison between the amount of computing expected in the type of conservative system outlined above and other event driven simulators such as TimeWarp. Let E be the maximum number of events possible in one (simulated) time unit, χ the cost of doing one suspension, ε the cost of executing one event and λ be a parameter that expresses the fraction of the maximum possible number of messages that are actually sent. Then the total cost of advancing one simulated time unit in the conservative scheme is given by the expression:

$$t_c = \lambda\varepsilon E + \chi C$$

For an event driven system the cost is proportional only to the number of events:

$$t_o = \lambda\varepsilon E$$

Clearly as λ decreases the event driven system will eventually win out over the conservative as the costs of an event driven system are proportional to the number of events. The per event overheads of the conservative simulator will be inversely proportional to λ - an effect which is clearly seen in the empirical results below. This analysis is generous to real parallel event driven systems such as TimeWarp where the per event overheads will be higher than the conservative system because of the need to do state saving and rollback.

Empirical Results

The conservative simulator was implemented on a SPARC-1000 platform with 8 55MHZ Sparc processors. An initial series of experiments were conducted to collect performance measurements from the proposed mechanism in order to compare against a standard event list

based sequential simulator, as well as a parallel simulator based on Time Warp. The benchmark used was a 4-dimensional hypercube communication network. Upon receiving a message, a node forwards the message, with an exponential delay with mean 1.2, to a neighbouring node. The selection of a destination node is random. The transit time on each link is set constant to 1.2 time units. As in ATM systems it is assumed that a message prevents another message being sent for one time unit. The experiments use various message populations, ranging from 1 to 1000. Thus the number of pending events per process will vary from 1/16 to 64.

The event granularity for the basic sequential simulation varied from 11 μsecs to 18 μsecs (for a large event list). It is difficult to separate the user code and the simulator overheads but the user code takes about 4 μsecs (mainly for random number generation). A spin-loop is also employed in some cases to investigate the impact of larger event grains.

The average execution time per event versus the message population is shown for a single processor in Fig. 4a. Following the theoretical predictions above, the execution time per event for the conservative simulator increases as the message popu lation decreases. The execution time for the sequential and TimeWarp simulators show an opposite trend increasing slightly as the message population increases - this is probably caused mainly by an increase in the size of the event list (the sequential simulator uses a splay tree and the TimeWarp simulator a calendar queue).

The other three subgraphs of Fig 4 show the overhead on 2, 4 and 8 processors. The performance of the conservative simulator improves smoothly in the parallel execution. Interestingly the TimeWarp simulator has difficulties at low message populations on the parallel runs. There has not been a chance to investigate this more closely but it may indicate the onset of some form of dynamic instability.

The results from the proposed mechanism are encouraging. Even at very low granularities the conservative mechanism shows speedup over a significant range of message populations. Preliminary estimates indicate that average message populations in ATM simulations will be well above the cross-over where the conservative system performs better than TimeWarp.

Fig. 5a shows the absolute speedup of the conservative simulator and Time Warp compared to the sequential simulator without any added granularity. Four different message populations, namely, 20, 100, 200 and 1000, were used. For large message populations, the proposed mechanism shows a speedup of 3.96 when running on 8 processors. Even with moderate message population, eg 100-200, the speedup observed on 8 processors is 2.82. Interestingly, the single processor version of the distributed conservative algorithm shows a slow down of about 37% with a message population of 1000. In principle it should run at about the same speed as the sequential simulator (indeed a sequential implementation of the conservative simulator runs slightly faster than the standard sequential simulator for large message populations). The problem seems to lie in the implementation of the threads package on the Solaris operating system. This requires a system call to access memory which is local to a thread. The structure of the C++ system we are using forces such a call on every send. This seems to account for the slow down (investigations are proceeding on how to avoid this overhead).

In Fig. 5b-d, the same set of experiments are repeated, but with different event granularities. Each figure shows the speedup when a spin-loop of granularity around 10 μS, 100 μS, 1000 μS is added to each event (in addition to the original computation of the application). As seen from the

figures, when the granularity increases, the speedup increases. Even with a small increase in event granularity (10μS), the speedup is elevated to about 4.91 on 8 processors when message population is high. Recent timing results on an implementation of the ATM-TN model indicates that the average event grains are about 17μS slightly higher than the grain used in Fig 5b. Time Warp also improves its performance drastically on larger event grains, leading to smaller differences between the two techniques. The results here are kind to TimeWarp in that the added compute grains include no provision for any additional state saving overhead.

ATM-TN Port

The full ATM-TN model has been ported to the a parallel version of the new simulator with encouraging preliminary results. A benchmark consisting of a per-port switch connecting three endnodes has been used to test the system. Two of the endnodes have ethernet traffic source / sink modules running. The switch has three ports, at 45 Mbps each. The collected data in Table 1 shows that when the load on ethernet is low (λ=0.045), both the sequential simulator and the Time Warp simulator outperformed the conservative algorithm. However, as the ethernet load increased (λ=0.11 and 0.18), the execution time per event for the conservative simulator decreases greatly while the Time Warp based simulator and sequential simulators remain the same. The real strengh of the conservative simulator appears on executing on multiple processors. In one experiment, the speedup (vs. sequential simulator) on 4 processors is close to 2.5. Further experiments are planned for more realistic ATM networks. Table 1 shows the execution time per event (in μS) for the different simulators.

Summary

A new shared memory based conservative simulator has been proposed and implemented. It is capable of exhibiting high efficiencies with overheads on a parallel implementation less than twice that for an optimised sequential simulator. It is well suited to its proposed domain of application in ATM models with its high efficiency overcoming problems of very low compute grain sizes. The parallel simulator compares well with a TimeWarp simulator over a wide range of parameters.

λ	con	con (4)	TW	TW (4)	Seq
0.045	32.02	15.69	40.16	27.22	25.37
0.110	26.53	12.01	42.40	27.06	25.52
0.180	24.74	11.80	42.15	26.97	28.07

Table 1. Run Times for ATM-TN

The conservative algorithm relies on a number of features of ATM networks to achieve good performance:

- all nodes (switches) have a small number of incoming and outgoing links;
- the capacity of links is limited by the (fixed) length of cells (this ensures a minimum lookahead for links);
- most "interesting" simulations will have links loaded close to capacity.

Clearly the algorithm is not universal as there will be problems where other mechanisms including TimeWarp will outperform it, however, it is well suited to ATM models and similar communications systems which are demanding and economically important.

Acknowledgments

This work was supported by grant 95-UOW-S14-4321 from the New Zealand Public Good Science Fund and by Science Applications International Corp. We would like to thank Xiao Zhonge and the rest of Telesim team for providing us with both sequential

and parallel simulators and the ATM-TN model.

References

1. Arlitt, M., and Williamson, C.(1995a) *"A Synthetic Workload Model for Internet Mosaic Traffic,"* Proc. Summer Computer Simulation Conf., Ottawa, June.

2. Arlitt, M., Chen, Y., Gurski, R., and Williamson, C.(1995b) *"Traffic Modelling in the ATM-TN Telesim Project,"* Proc. Summer Computer Simulation Conf., Ottawa, June.

3. Boudec, J.L.(1992) *"The Asynchronous Transfer Mode: a Tutorial,"* Computer Networks and ISDN Systems, pp. 279-309.

4. Cai, W., and Turner, S.J.(1990) *"An Algorithm for Distributed Discrete Event Simulation,"* Proc. Distributed Simulation Conference, San Diego California, pp. 3-8, JanuaryJanuary.

5. Chandy, K.M., and Misra, J.(1979) *"Distributed Simulation: a case study in design and verification of distributed programs,"* IEEE Trans.Software Eng., **5**(5), pp. 440-452, September.

6. Chandy, K.M., and Sherman, R.(1989) *"The conditional event approach to distributed simulation,"* Proc. Distributed Simulation Conference, San Diego, California, pp. 93-99, March.

7. Comm. ACM.(1995) *"Special Edition on Issues and Challenges in ATM Networks,"* Comm. A.C.M., February.

8. Das, S., Fujimoto, R., Panesar, K., Allison, D., and Hybinette, M.(1994) *"A Time Warp System for Shared Memory Multiprocessors,"* Winter Simulation Conference, December.

9. Fujimoto, R.M.(1988) *"Performance measurements of distributed simulation strategies,"* Proc. Distributed Simulation Conference, San Diego, California, pp. 14-20, February.

10. Fujimoto, R.M.(1990) *"Parallel Discrete Event Simulation,"* Comm. A.C.M., **33**(10), pp. 30-53, October.

11. Jade Simulations International Corp.(1995) *"Deliverable for ATM-TN Performance Project,"* Science Applications Internationl Corp., August.

12. Jha, V., and Bagrodia, R.L.(1993) *"Transparent implementation of conservative algorithms in parallel simulation languages,"* Winter Simulation Conference, Los Angeles, pp. 677-686, December.

13. Misra, J.(1986) *"Distributed Discrete-Event Simulation,"* ACM Computing Surveys, **18**(1), pp. 39-65.

14. Nicol, D.M.(1988) *"Parallel discrete-event simulation of FCFS stochastic queuing networks,"* ACM SIGPLAN Notices, **23**(9), pp. 124-137.

15. Unger, B.W., Gomes, F., Zhonge, X., Gburzynski, P., Ono-Tesfaye, T., Ramaswamy, S., Williamson, C., and Covington, A.(1995) *"A High Fidelity ATM Traffic and Network Simulator,"* Winter Simulation Conference, Washington, D.C., December.

16. Wagner, D., and Lazowska, E.(1989) *"Parallel simulation of queuing networks: limitations and potentials,"* Proc. International Conf. on Measurement and Control, pp. 146-155.

```
Top: pointer to next location to have a message stored;
Bottom: pointer to next message to be received;
structure link:
        time: receive time of message (or earliest time of next
                message if pointed at by Top);
        next: pointer to next message (undefined if
                pointed at by Top);
        data: user defined content of message (undefined if
                pointed at by Top);
end structure;
initialise: %Initialise link.
        t:= new link;
        t.time:=0;
        t.next := nil;
        Top:=t;
        Bottom:=t;
end initialise;
send(receive_time,next_time,msg): %Send a message.
next_time is the earliest possible receive time of the next message;
        Top.data:=msg;
        Top.time:= receive_time;
        t:= new link;
        t.time:= next_time;
        Top.next := t;
        Top:= t;
        send := true;
end send;
early_time(time): %Update earliest possible receive time:
        Top.time:= time;
%Receive a message - returns true if Bottom is left pointing at a valid message - in
any %case Bottom.time is left as the earliest time a message can arrive down the link;
receive:boolean;
        if Bottom = Top then
                receive:=false;
        else
                t:=Bottom;
                Bottom:=Bottom.next;
                deallocate t;
                receive:= boolean:(Bottom<>Top);
        end if;
```

Figure 2. Code for Link Datastructure.

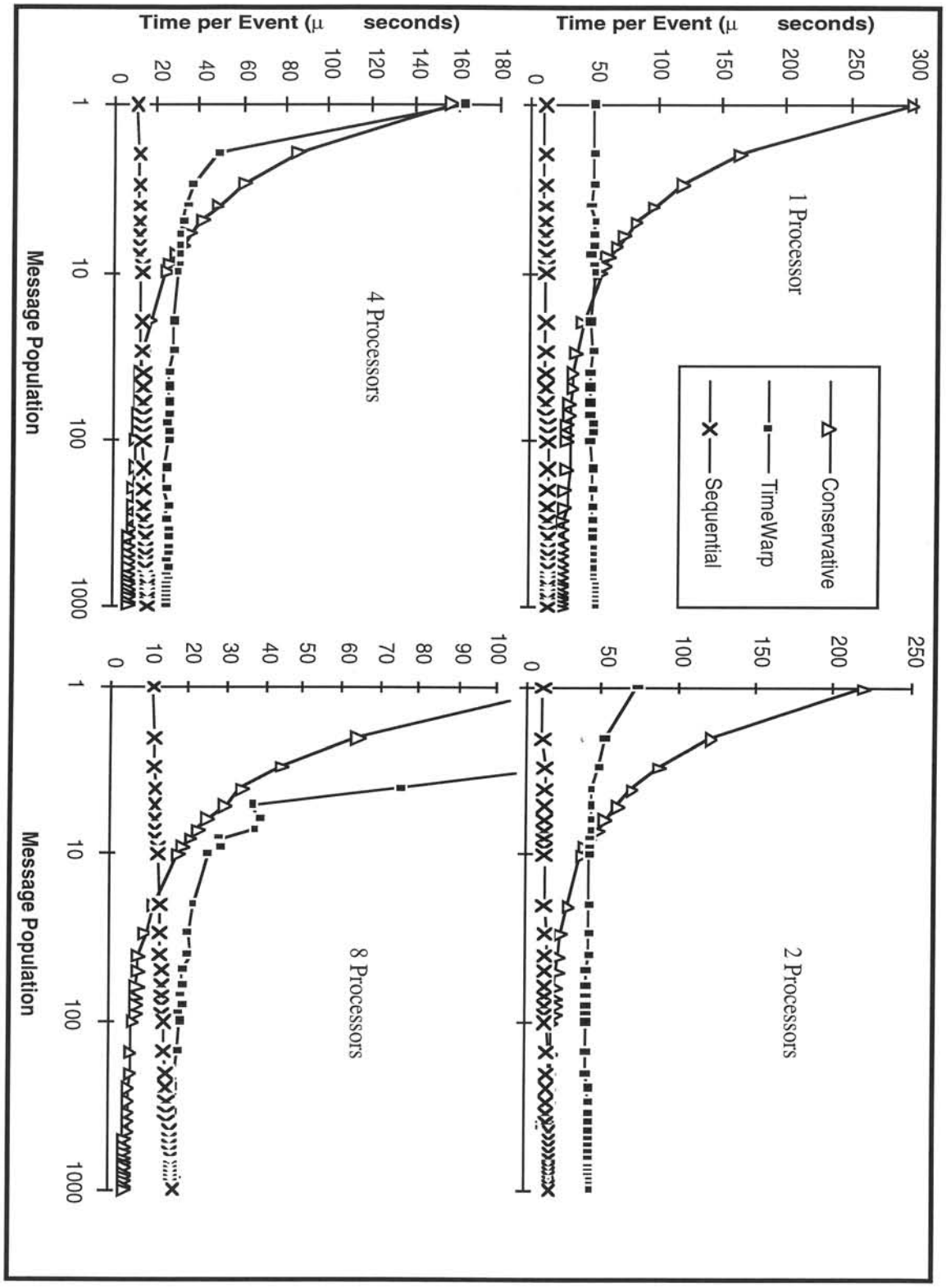


Figure 4. Event Overhead

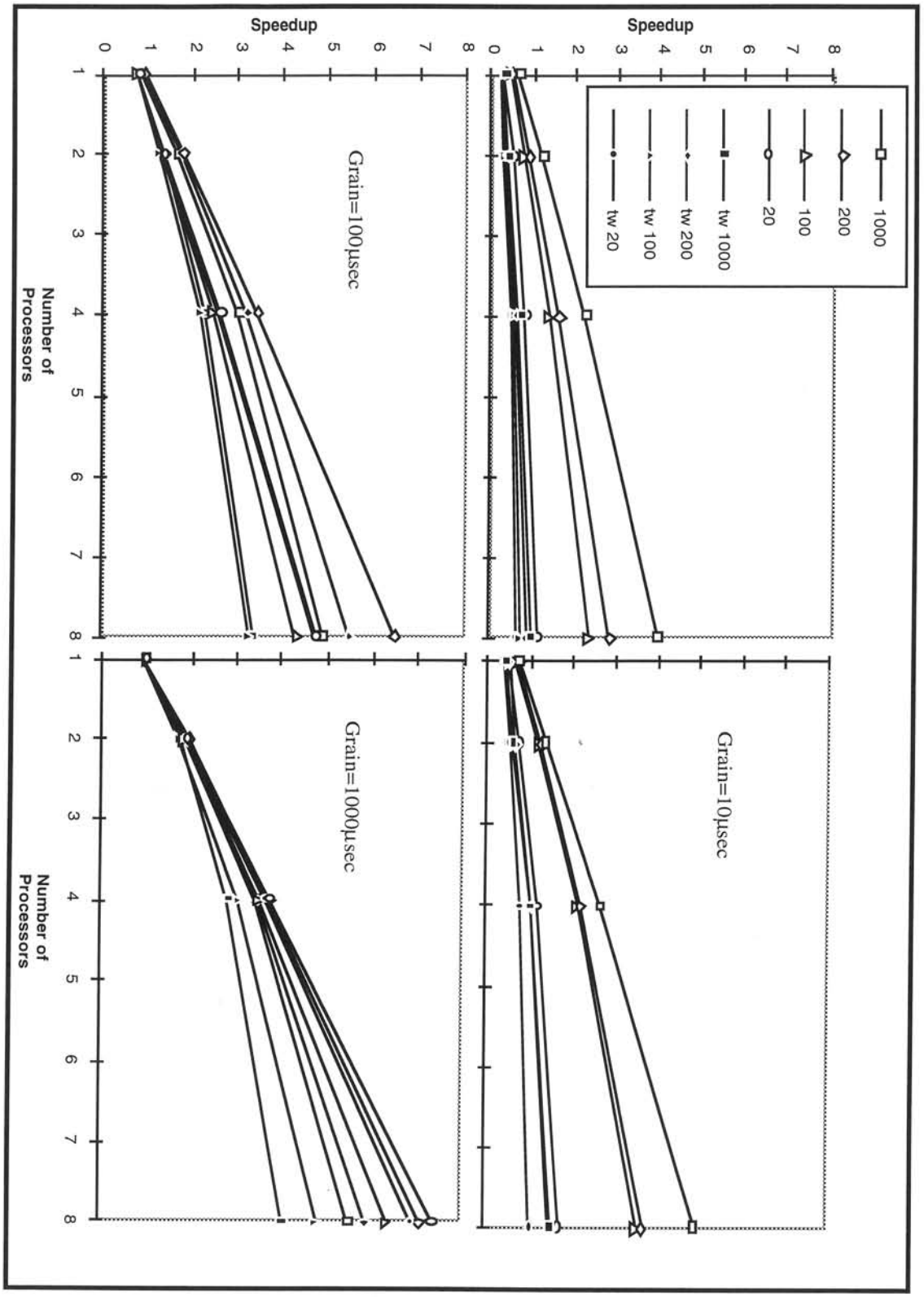


Figure 5. Speedup