Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

A PARALLEL SIMULATOR FOR PERFORMANCE MODELLING OF BROADBAND TELECOMMUNICATION NETWORKS

Richard W. Earnshaw

Computing Laboratory
University of Cambridge
Cambridge, CB2 3QG, U.K.

Alan Hind

School of Engineering and Computer Science
University of Durham
Durham, DH1 3LE, U.K.

ABSTRACT

This paper describes the structure of a parallel simulator developed to investigate the performance of broadband telecommunication networks. The simulator hardware is based on a reconfigurable array of Inmos transputers. The software has a layered architecture and issues of efficient communication, deadlock avoidance and message routeing have been addressed. The synchronization mechanism used is conservative, based on the Chandy-Misra model, and exploits lookahead. The speed-up results are almost linear when compared with the same parallel simulation run on a single transputer and are still impressive when compared with an optimized single processor version.

1 INTRODUCTION

The performance evaluation of telecommunication networks rapidly becomes analytically intractable as the complexity of the network increases. In addition to this, the behaviour of interest to the performance engineer is often that which occurs under transient conditions, such as traffic fluctuations or component failures which are known to be difficult to express mathematically.

Under such conditions the use of simulation techniques to determine relevant performance parameters becomes necessary. Conventional sequential simulations running on sequential computer architectures suffer from limitations imposed by the excessive processing time required to achieve the required depth of information and the intrinsic statistical nature of the results. These problems increase as functions of the traffic intensity and the size and complexity of the network. This leads to detailed simulations, of traffic intensive and very large networks, often being economically and even physically impossible to implement. Such problems have led to growing interest in parallel simulation using multiprocessor hardware.

The argument becomes most pointed when considering the performance evaluation of broadband networks. The complexity, traffic intensity and the potential size of the networks are all large. Simulation studies of systems of this nature have thus far largely centred on the behaviour of single traffic sources, multiplexers or switching nodes. When complete networks have been studied it has usually been at the call- or burst-level since lower-level simulation involves levels of complexity (and hence processing time) which are orders of magnitude greater.

Nevertheless, for many investigations of network behaviour, lower-level simulation is unavoidable and has motivated the development of a parallel multiprocessor simulator by the University of Durham Telecommunication Networks Research Group. The simulator is being used for the study of network behaviour and, particularly, for studying the integration of mobile communication protocols into the broadband environment (Earnshaw and Mars 1991).

The use of parallel simulation introduces many additional issues into the simulation design process. These include the hardware architecture, the decomposition approach used to produce the parallel software processes, mapping these processes onto the processors and the synchronization of the resulting parallel simulation. Reviews of the application of parallel simulation techniques to the performance evaluation of communication networks have been written by Mouftah and Sturgeon (1991) and by Hind (1991). An excellent general review of parallel simulation has been written by Fujimoto (1990).

2 BROADBAND NETWORKS

The CCITT define an integrated services digital network (ISDN) as one " ... that provides end-to-end digital connectivity to support a wide range of services, including voice and non-voice services, to which users have access by a limited set of standard multi-purpose user-network interfaces" (CCITT 1984). Initially, *basic access* was centred around two 64 kbits/s B channels and one 16 kbits/s signalling D channel. The requirement for supporting more advanced multi-media services within ISDN has led to the development of broadband ISDN (B-ISDN).

The asynchronous transfer mode (ATM) is the target solution for B-ISDN defined by the CCITT. ATM networks use a fixed-size data packet, known as a cell, which consists of 48 octets of data and 5 octets of header. They are typically transmitted, within the network, using multi-megabit-per-second media, such as fibre-optic links; such links will typically be running at data rates in excess of 150 Mbit/s. A good background text on B-ISDN, and

ATM networks in particular, has recently been published by Händel and Huber (1991).

The ATM switch used in the simulation study is based on the Orwell ring protocol which is a slotted ring protocol described by Chauhan, King and Micallef (1990). The ring is divided into slots which circulate around the ring; a node wishing to transmit a message waits until an unfilled slot is found, changes the slot header and transmits the message in the body of the slot. Slotted ring protocols have been unpopular in the past for several reasons. A monitor node is required to ensure that slots which become corrupted can be identified and regenerated, thus correct behaviour of the ring is critically dependent on correct behaviour of the monitor. To get a reasonable number of slots onto the ring delays have to be inserted at each node and one node, normally the monitor, has to be able to adjust its delay so that there are an integral number of slots. Finally, the efficiency of slotted rings is generally poor since the ratio of header to body is normally high. Its greatest advantage over token-based protocols, however, is that more that one node can be transmitting information at a time, using different slots on the ring. Acknowledgement of delivery is normally made by releasing the slot at the source (correct receipt there is taken to imply correct delivery at the destination); the node may not refill a slot that it has just released, ensuring that the slot is passed to the next node and thereby ensures fair access to all nodes on the ring. An earlier implementation of a slotted ring is the Cambridge ring protocol (British Standard BS6531).

Examination of existing protocols has indicated that those based on a slotted ring are probably the best suited for carrying delay-sensitive traffic such as speech, but simulation studies of high-bandwidth Cambridge Rings have indicated that there are still significant limitations when operated under high load (Falconer, Adams and Walley 1985). Further, load control is difficult since there is no relevant parameter that can easily be extracted from the ring. The Orwell protocol was developed after making a detailed study of the limitations of the Cambridge Ring protocol: it was found that by introducing destination release of slots, and by adding a novel, distributed, load control mechanism to bound access delays, a viable level of performance could be obtained. These developments are discussed by Adams and Falconer (1984) and Falconer and Adams (1985). For higher capacity networks multiple, synchronized, rings can be used and such a network is known as an Orwell Torus.

Whilst detailed simulations of a single Orwell ring have been made, under a variety of load and traffic services, there has, as yet, been very little investigation made into the behaviour of an Orwell torus, or ring behaviour in multi-ring systems. The reason for this, at least in part, is because of the large amount of simulation time required to investigate networks of Orwell rings: a single simulation run of one ring takes, typically, a couple of hours on a VAX 11/750, or three times as long on a Sun 3/50 workstation for just a couple of seconds of simulated time.

3 SIMULATOR ARCHITECTURE

3.1 The Multiprocessor Testbed

The multiprocessor testbed used for the ATM simulator is based on a network of Inmos transputers. This was originally designed for use as a high-speed circuit-switched network simulator, with code written in occam; subsequently, a traditional packet-switched network simulator was also developed using the same language (Nichols 1990 and Clarke, Nichols and Mars 1989). The transputer network consists of up to 31 simulation transputers, each with up to 16 Mbytes of memory (the current implementation consists of 13 T800 processors each with 1Mbyte of memory). The transputers are connected with a double layer of cross-point link switches which enables any link on each of the simulation processors to be connected to a link on any of the other processors; this flexibility enables the network to be configured in arbitrary topologies so that the system being simulated can be mapped closely onto the processor network, and enables the path length required when passing messages between processors to be kept to a minimum. Finally, a layer of control processors are used to connect between the host transputer and the link switches; one is connected to the link-switch programming interface, while both can be connected, via the switches, to any of the simulation transputers. An optional transputer-based graphics card can also be connected at this layer.

3.2 The Software Architecture

To isolate the simulation model, as far as possible, from the implementation details of the hardware, the simulator was structured in a hierarchical manner; each layer building on the abstraction of the layer below in a similar approach to that of the ISO seven layer model. At the lowest layer lie the transputer processors in a dynamically reconfigurable array. On top of this a multiplexer task on each processor provides the abstraction of virtual channels between each task in the simulation, regardless of where the tasks are mapped in the processor network. A simple packetizer layer hides the fact that the channels in the multiplexer (and, indeed, the physical channels of the transputer itself) work most efficiently when presented with large packets as opposed to a series of very small ones. A synchronization layer uses the packet layer processes; it ensures that each message is correctly marked with a time-stamp on dispatch and uses this at the receiver to maintain synchronization: the layer is optional, if there is no definable synchronization between two tasks (for example, diagnostic messages destined for the console) then the channel can be declared asynchronous and the packet layer accessed direct. Finally, in parallel with the simulation model and the synchronization layer, an event manager is responsible for scheduling components of the simulation model in the correct sequence. The overall hierarchy is summarised in figure 1. The implementation is described by the authors in more detail else-

where (Earnshaw and Mars 1990, Hind 1990 and Earnshaw 1992).

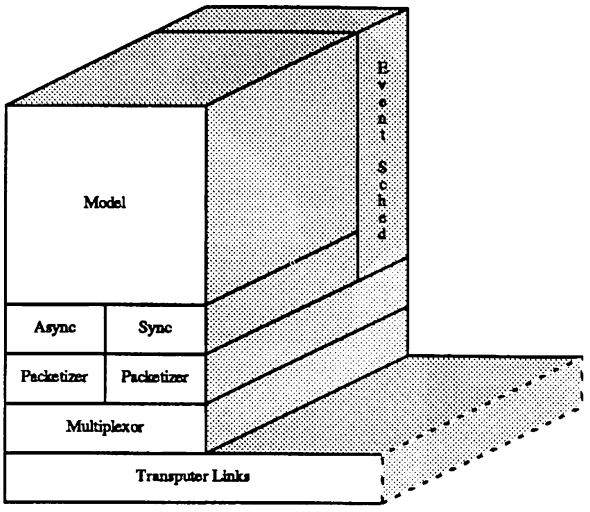

Figure 1: The overall hierarchy of the simulation model. The Event scheduler is a control-plane for the upper layers.

3.2.1 The Multiplexer

The multiplexer is the lowest layer of the simulator kernel; it is responsible for the delivery of messages from one task in the simulator to another, regardless of the topological mapping of either the tasks or the processors upon which they are running. Each transputer in the network is allocated exactly one multiplexer task; all other tasks desiring to communicate with tasks on another processor do so by communicating indirectly via the multiplexer (figure 2). If two tasks that communicate are on the same processor then it is, of course, possible for them to be directly connected. The result is that for large simulations a simulation task may have many of its channels connected to the multiplexer; the routeing decisions that the multiplexer makes are based solely upon the channel from which each message is received.

3.2.2 The Flow Control Mechanism

The flow control mechanism has to ensure two things: firstly that the multiplexer routeing, as a whole, can operate within a fixed amount of memory, i.e. a finite number of buffers (deadlock free); and secondly that all messages will be eventually delivered, regardless of the other traffic in the multiplexer (livelock free). The algorithm adopted for the implementation of the deadlock- and livelock-free routeing is based on that of Toueg and Ullman (1979), using a forward state controller.

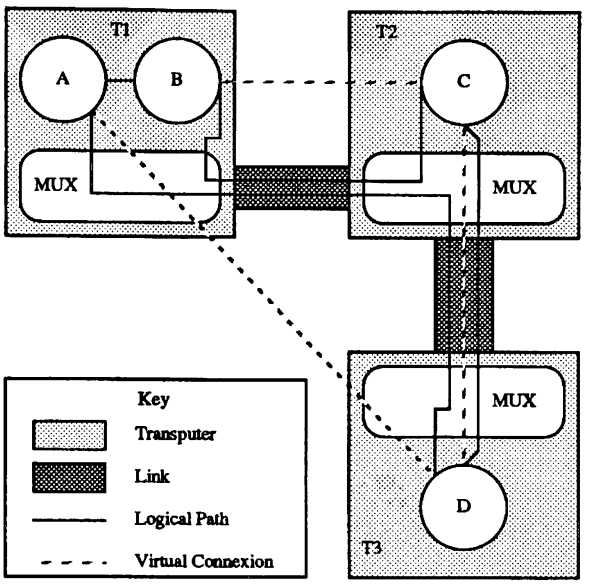

Figure 2: Multiplexer processes run on each node to provide virtual links between each task in the simulator.

3.2.3 The Packetizer

Messages between the simulation tasks commonly consist of several small pieces of information: for example, a cell has associated with it not only the time of transmission and the data and header fields but also the time of creation, the size of the data field in use (for efficiency) and an optional series of trace information packets that can be used when debugging the simulator. If each item were to be transmitted individually across the processor network then the efficiency of the multiplexer would be extremely poor; each packet in the multiplexer would contain perhaps as little as four bytes of information and an overhead of eight bytes (four bytes for each of the packet-header and the packet-size fields). To overcome this inefficiency, each simulation message (e.g. a cell) is concatenated into a single packet (or a series of packets if this would exceed the maximum size of a single multiplexer packet) which is then transmitted to the receiving process.

In addition to the inefficiency associated with using the multiplexer with small units of data there would also be an overhead due to the establishment of the occam channel for passing data between one task and the next. Each communication requires that both ends (the sender and the receiver) are ready to proceed before any data can be sent: if one end is not ready the other task blocks whilst waiting. Because of the way in which the transputer's process scheduler works this can mean a large number of process switches, each switch having an overhead in terms of processor time; in addition, each time a process is descheduled it is placed at the back of the relevant queue (either high or low priority) and has to wait its

turn for further access to the processor. It is clearly more efficient if the number of times a channel communication has to be initiated is kept to a minimum; work by Gould, Bowler and Purvis (1989) has shown that the throughput of the channels increases dramatically as the size of the data block is increased.

3.2.4 The Event Manager

The event list is normally maintained using the twin-list method described by Blackstone, Hogg and Phillips (1981), but it is possible to convert the procedures to be functionally the same as a single list manager by setting the initial length of the first list to infinity. It was found that for the T4 series of transputers (which do not support floating-point arithmetic in hardware), using the twin list method approximately halved the amount of time spent maintaining the event list, but for the T800 transputer (which does support floating-point arithmetic) the change was negligible; indeed, for some configurations, the twin list procedure was slower by about 0.5%.

3.2.5 Configuring the Simulator

For any simulation tool to be useful it must be capable of being run with a series of different configurations, the extent of which has to be borne in mind when the simulator is designed. For a truly flexible system it is not normally sufficient for these to be parameters that are 'hard coded' into the simulator itself; instead, they should be made available from a separate file (or by interactive prompting) at the time the simulator is invoked. In the ultimate case, not only parameters such as load and various delays should be configurable, but also the entire topology of the network itself: this can require substantial effort being expended on making the simulator easier to use, but, consequently, significantly more powerful.

The method employed here is a parse-able grammar that describes the simulation parameters (and some of their dependencies) in a human comprehensible format: in such environments it is rarely necessary for the information to be in a totally fixed order since each parameter will have a tag associated with it that describes it uniquely. Comments are normally easily supported. An example entry might contain:

```
link 5:
% Link between nodes 1 and 6
     prop_delay = 100u S
     speed = 100M bitps
     ;
```

Parsers for grammars of this type are easily produced using tools such as `yacc` and `lex` and would, probably, be implemented using a preprocessor for the simulator that produces the configuration tables that the simulator itself reads. Another advantage of this approach is that default values can now be used: a special entry (for example '`link default:`') might contain a series of fields that should be used when a real definition omits a parameter.

The ATM Network Simulator currently parses two files when it starts to run: the first describes the topology of the network being simulated and how the individual processes should be mapped onto the processors of the transputer network; the second contains the various parameters required by each individual process. Both files are of the 'table of values format'. A parser is available for generating the first file that understands a superset of the *3L configurer* language (Parallel C User Guide 1989); the extensions are mainly aimed at supporting the reconfigurability of the transputer array used. The second file has to be generated by hand, but a built-in preprocessor parses the special symbols '`%date`' and '`%seed`', replacing them with the current date and an unique seed respectively. The seeds are generated using a different random number generator from the one used during simulation in order to avoid, as far as possible, correlations between the random number streams.

The compiler package supplied by 3L Ltd is described in the Parallel C User Guide (1989) and consists of three main components for use with multi-transputer networks: the compiler, which produces object modules from the source files; a linker, which links object modules and libraries to create tasks; and a configurer, which binds several tasks together to form an executable application. A task is a program in its own right: it is allocated a stack and an area of memory, and has its own global variables; it must always run on one processor, but can spawn *threads* which execute part of the code of the task in parallel and share the memory (they each, however, have their own stack); a task can only communicate with other tasks by using the occam channels implemented in the processor hardware: the collection of program threads in a task are collectively referred to as a process. The configurer is responsible for allocating tasks to processors, creating initial stacks and heap areas, and for mapping the connexions between tasks onto occam channels (both internal and external).

Unfortunately the configurer supplied with the compiler does not support the link-switch mechanism in the transputer network used and, therefore, cannot be used in the traditional sense to boot the entire network. The approach used in the simulator, is to have a small main application, which runs on the fixed topology part of the network, and a series of un-configured tasks. The main application does on-the-fly configuration of the remainder of the application using a single file that describes the simulation run. To do this it uses the low-level configurer execution primitives to load the tasks directly into each processor.

Once each task has been loaded and has started to run, the simulation parameter files have to be loaded. Unlike traditional simulators this poses a large problem: part of the information contained in the parameter file is used by the multiplexers to control the switching of messages; until this is loaded they cannot operate properly. Similarly, none of the other tasks knows any information about where it lies in the overall topology, since to provide this information would require 'hard coding'. Indeed, the only

information that each process has is its own array of channels for use in communicating, but even this has little meaning unless some conventions are used. Fortunately, 'false' channels can be created during the configuration process and their values set to represent something other than a genuine channel. With this information, known as a 'tag', each task in the simulator can be uniquely identified, enabling it to extract the relevant information from the parameters file.

At this stage a task still does not know on which input channel it will receive the configuration information; further, it does not know on which output channels, if any, it must forward the information so that it can reach its neighbours. To obtain this information a boot-tree is built which starts at the task connected to the fixed topology part of the network (there is exactly one such task) and extends outwards until all the tasks know a parent and any children they might have. The protocol for doing this in the presence of loops is quite complex if the use of timeouts are to be avoided; the petri-net in figure 3 represents the code running on just one channel pair of one task (all of the channels in the simulator are paired, one input and one output, to the same remote task), the same code runs on each channel pair throughout the simulator.

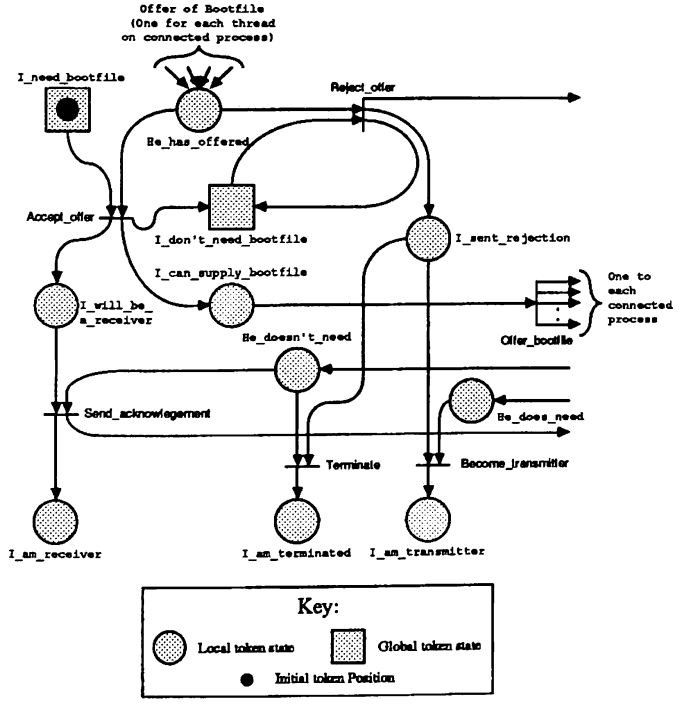

Figure 3: Petri-net showing the state transitions for a single channel while determining the download path for the simulator. The 'square' states are shared by all of the channels, which have been omitted for clarity, making it impossible for more than one channel to be activated as a receiver.

The parameters file contains a few lines of global information, such as the title of the simulation run, the size of the network, and for how long the run must last, followed by a series of entries, one for each task in the simulator. To avoid the need for each task to have to be able to interpret information for other tasks (which may well be of a different class), each task scans the parameters file looking for a string of the form '`class xxx:`', where the `class` is the type of task ('SRCE' for a traffic generator, 'MUX' for a multiplexer, etc.) and `xxx` is the tag-value that was bound to the false link. On finding this string, the task then interprets the following parameters as its personal configuration file. Special routines are used to parse the file while ensuring that at the same time the entire file is passed on to its children in the boot-tree without modification or loss. Once the entire file has been read and interpreted, the configuration process is complete and the simulation can begin.

4 THE SYNCHRONIZATION MECHANISM

In a sequential discrete event simulation, the synchronization of the simulation is maintained by manipulation of a data structure called the event list. This contains the pending events in the system in time-stamped order. The simulation progresses by removing the event with the earliest time-stamp from the list and processing it. If another event is generated, it is inserted into the event list at its time-stamp position. Thus the simulator processes the events in synchronized chronological order. If we now distribute the simulation over several processors, each having a local event list, it becomes possible for a processor to process an event which is not the earliest. Also, in processing this event we may affect conditions for earlier, as yet unsimulated events. Thus the future is affecting the past, which is clearly unacceptable, and is known as a *causality error*.

Thus, synchronization schemes can fall into one of two categories; conservative approaches and optimistic approaches, see Fujimoto (1990) for a fuller explanation of these terms. Conservative approaches avoid causality errors ever occurring by relying on some strategy of determining events which are "safe" to process. That is, they must determine when all events that could affect the event in question have been processed. An added problem which categorises various conservative approaches is that of deadlock. If processes do not have a "safe" event which they can process then they are blocked and cannot progress. If a cycle of blocked processes occurs then we have deadlock and the simulation will grind to a halt unless the deadlock can be broken. In the Chandy-Misra conservative approach used here (Chandy, Holmes and Misra 1979, Chandy and Misra 1979 and Chandy and Misra 1981), NULL-messages are used to avoid deadlock situations occurring. NULL-messages are only used for synchronization purposes and do not correspond to any activity in the physical system being simulated and, hence, have no message content only a time-stamp t_{Null}.

Thus, it is essentially a promise that the sending process will not send a real message to the destination process with a time-stamp less than t_{Null}. NULL-messages are sent on each outgoing port whenever a process finishes processing an event; t_{Null} being a lower bound on the time-stamp of the next outgoing message on each outgoing port calculated from the time-stamp value associated with each incoming port and knowledge of the simulation performed by the process. Generally conservative synchronization approaches can achieve good performance with sparsely-connected systems which have less opportunity for deadlock and/or an application which contains good lookahead properties. Lookahead refers to the ability to predict what will, or will not, happen in the simulated time future based on application specific knowledge.

For an ATM link it is possible to derive a simple formula that describes the number of cells that will be in transit across a link of given length at any one time (the link can be considered as a delay line):

$$N = \frac{LSn}{lc}$$

where L is the length of the link, S is its speed (adjusted to account for overheads such as framing), n is the refractive index of the transmission media (typically, about 1.5 for a glass fibre), l is the cell size and c is the speed of light. Considering, for example, a 15 km link running at 150 Mbit/s, then there may be up to twenty-six cells in transmission across the link at any time; longer, or faster, links would have correspondingly larger numbers of cells in transit. This "pipeline" is used to advantage as a method of lookahead within the simulator. Effectively, a destination task can see a small amount of future behaviour for the link: this can then be exploited for two ends; the avoidance of deadlock with fewer NULL-messages and the improvement of concurrency between the processes.

In the Chandy-Misra simulation, there is not normally an event processor in the classical sense. Instead, events are replaced exclusively by messages and the order of processing is determined by selecting the message with the oldest time-stamp: there must be a message available from each incoming link in order to be able to do this; the absence of a message causes the node to block. In the ATM Network Simulator an event manager is used; consequently, in addition to adding dependence on the link mechanisms to the code of the event manager, monitoring for messages would be inefficient. To overcome this, the synchronization routines are implemented as normal events that run in the same manner as all other events in the simulator: two events are required for each link to a remote process; these are a NULL-message generator and a process blocker.

The NULL-message generator runs on the output of a link: it compares the current simulation time with the time when a message was last sent to the remote process; if this is less than a propagation delay it simply reschedules itself to a time one propagation delay later than the time at which the last message was sent; otherwise, it must be exactly one propagation delay since a message was last sent, so a NULL-message is generated to the remote process and the generator reschedules itself one propagation delay later. The process blocker compares the simulation time against the time when a message was last received across a link from the remote process; if this is less than a propagation delay then it simply reschedules itself for one propagation delay after the time the last message was received; otherwise it blocks the current process until a message is received and then reschedules itself accordingly. The process blocker appears to the rest of the simulation as a routine that takes just sufficiently long to execute that the process remains in synchronization with its neighbours; however, while blocking, it consumes no processing time.

5 THE SIMULATOR RESULTS

The results produced by the simulator consists of sets of statistics for the simulator performance, the traffic patterns and the switching-node activity. The simulator performance can be assessed from the run time, processor usage and link usages. The performance of the synchronization mechanism is also monitored, along with several other aspects, by an event profiling process. This gives the number of instances and and percentage processing time spent on various simulation events. Such profiling is made easier as the transputer has hardware timers which allows the profiler to be run at fixed time intervals. Traffic patterns are reported as a set of histograms of the voice delay statistics for each source in the network. Switching-node activities are also reported as histograms of the input queue lengths to the Orwell rings, the ring reset and cell delay statistics.

6 PERFORMANCE ANALYSIS OF THE SIMULATOR

The ultimate goal with parallel simulation is to obtain a simulator that runs as quickly as possible; if the speed of the parallel simulator is less than that of a conventional simulator then there is no reason for using it (and many good reasons for not doing so). However, it is normally impossible to directly compare parallel and sequential simulators since the two are written in an entirely different manner and the programmer rarely wants to write both. A good indication of the possible behaviour of the conventional simulator can sometimes be obtained, though, by running an optimized version of the parallel simulator on a single processor. The time taken for the single processor version to run can be compared with that for the multiprocessor version and the *speed-up* of the simulator is then the ratio of the time for the multiprocessor version to that for the single processor: normally this should lie in the range between one and n, when the multiprocessor version is run on n processors; a speed-up of n is said to be *linear*, as defined by Helmbold and McDowell (1990). If the speed-up is greater than n we have

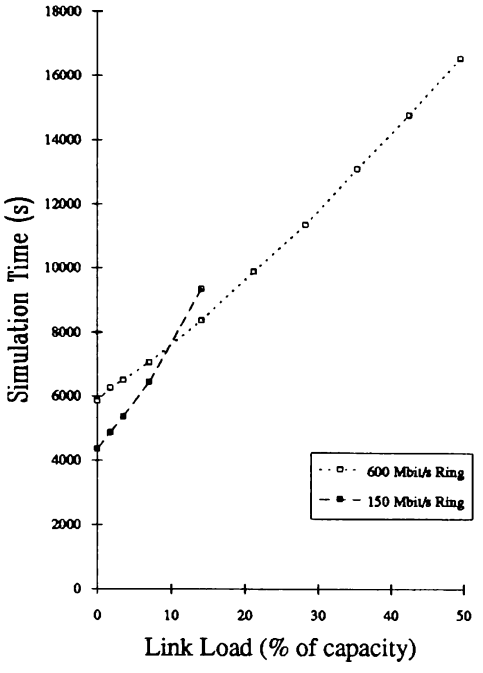

Figure 4: Simulation times for the twelve-node networks of 150 and 600 Mbit/s rings on twelve transputers.

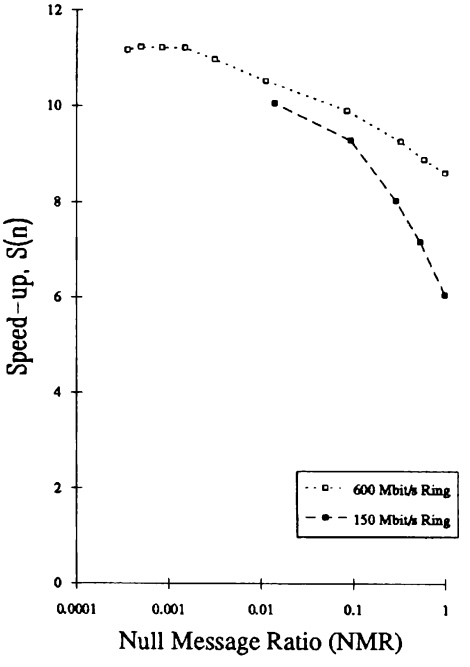

Figure 6: Speed-up as a function of NULL-message ratio. The difference between the two curves represents the extra parallelism that can be extracted from the higher speed rings.

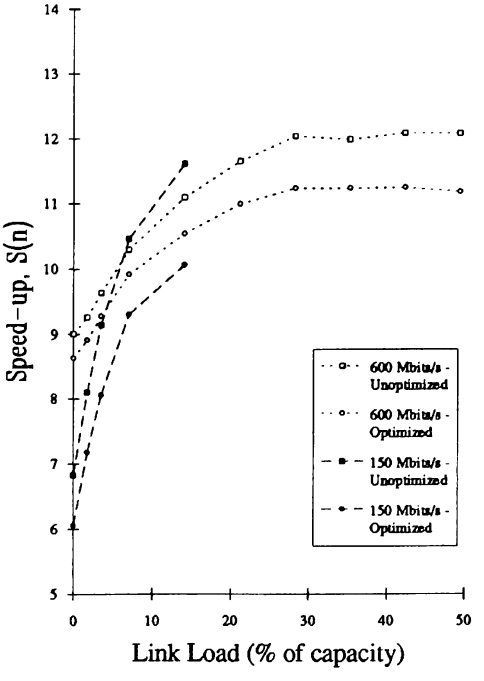

Figure 5: Speed-up for the 150 Mbit/s rings carrying mixed mobile and voice traffic and the 600 Mbit/s rings carrying voice traffic only.

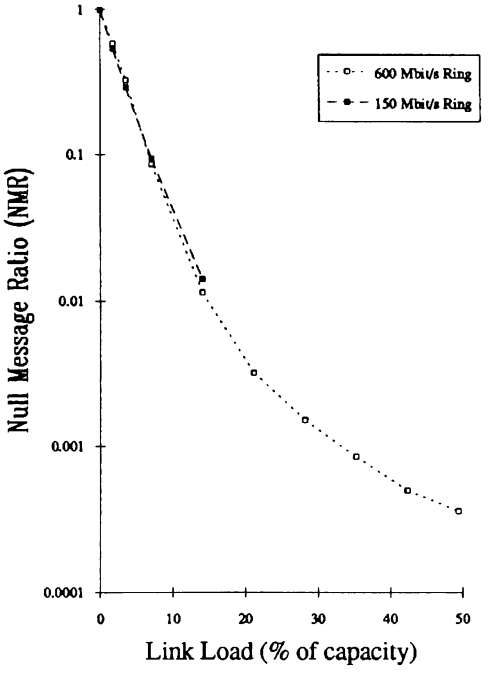

Figure 7: NULL-message ratio (NMR) as a function of load. As might be expected, the ratio is independent of the ring speed.

superlinear speed-up and, if the speed-up is less than n, we have *sub-linear speed-up*.

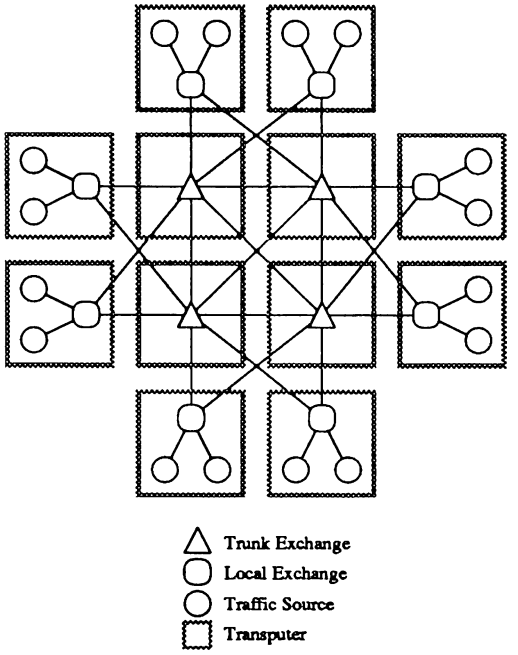

Figure 8: Network topology used for the simulator performance analysis runs. The basic processor assignments are also shown; a traffic source presents a very small load to a processor so it may be safely combined with a local exchange without unduly affecting the load balance.

The performance results given here are for the ATM Network Simulator configured as shown in figure 8: the network consists of four ATM exchanges in a fully-connected trunk network and eight 'local' exchanges each of which is dual-parented onto two trunk exchanges; each local exchange has two traffic generators. The exchanges were all running the Orwell ring protocol (see section 2). Two sets of results were taken with differing switch capacities and traffic mixes. In both cases the links were running at 150 Mbit/s and the propagation delay was set to 1×10^{-4} s (equivalent to about 20 km of glass fibre, or about 35 cells). The results for the lower traffic loads were taken using 150 Mbit/s Orwell rings for the switches and with a mixture of voice and mobile traffic; the results for the higher loads used purely voice traffic and a ring speed of 600 Mbit/s. With the smaller capacity switches the maximum link loading was about 15%, but was increased to about 50% for the high-capacity rings. Two single processor simulations were run for each load: one with identical code to the multiprocessor version, the unoptimized version; the other with the redundant multiplexers removed to speed message transfer, the optimized version. In the following graphs, when the load is shown it is expressed as the average percentage of the capacity of a link.

Figure 4 shows the time taken to simulate the two models on twelve processors. The fact that the two curves do not pass through the origin has two causes: the NULL-message traffic for low loads and the overhead of simulating the ring slot-rotation action for the Orwell protocol. That it is the latter that represents the largest factor can be inferred from the fact that the NULL-message traffic generated for each of the two curves is almost identical for a given link loading (see figure 7); so if this was the cause the two curves would cut the axis at the same point.

Figure 5 shows the speed-up of the simulator as a function of load. It shows, for the 150 Mbit/s rings, that even for a load of just 15% of maximum capacity, the speed-up is approaching the ideal linear value of 12 for the unoptimized version, and is starting to level out at just over 10 when compared with the optimized version. The difference between the two curves represents the proportion of the processing time that is taken up in switching the messages from one processor to another. The speed-up of the simulator relative to the unoptimized version can also be estimated from the processor activity monitoring of each of the transputers in use: the results from doing this agree well with the upper curve shown. Figure 5 shows, for the 600 Mbit/s rings that in comparison with the unoptimized single processor version the speed-up is greater than 9 for all loads simulated, and for link loads greater than 30% it is almost linear.

It can be seen from figure 6 that the speed-up degrades gracefully with increasing NULL-message ratio; but, fortunately, as can be seen from figure 7, the NULL-message ratio remains very low for a large range of the load.

ACKNOWLEDGEMENTS

The authors would like to thank British Telecom Research Laboratories and the Science and Engineering Research Council for both their technical and financial support of this work.

REFERENCES

3L Ltd, *Parallel C User Guide, Version 2.1.* Peel House, Ladywell, Livingston EH54 6AG, Scotland, 1989.

J. L. Adams and R. M. Falconer, "Orwell: A Protocol for Carrying Integrated Services on a Digital Communications Ring," *Electronics Letters*, vol. 20, pp. 970–971, November 1984.

J. H. Blackstone Jr, G. L. Hogg, and D. T. Phillips, "A Two-List Synchronization Procedure for Discrete Event Simulation," *Communications of the ACM*, vol. 24, pp. 825–829, December 1981.

CCITT: COM XVIII, *228-E.* Geneva, March 1984.

K. M. Chandy, V. Holmes, and J. Misra, "Distributed Simulation of Networks," *Computer Networks*, vol. 3, pp. 105–113, 1979.

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

K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Communications of the ACM*, vol. 24, pp. 198–206, April 1981.

J. Chauhan, T. King, and A. C. Micallef, *Specification of the Orwell Protocol.* British Telecom Laboratories, Martlesham Heath, Ipswich, Suffolk, UK. IP5 7RE, May 1990. Revision C.1(05/90).

R. T. Clarke, S. J. Nichols, and P. Mars, "Transputer-based Simulation Tool for Performance Evaluation of Wide Area Telecommunications Networks," *Microprocessors and Microsystems*, vol. 13, pp. 173–178, April 1989.

R. W. Earnshaw and P. Mars, "Simulation of ATM Networks on Transputer Arrays," in *Proceedings of the Seventh IEE UK Teletraffic Symposium*, pp. 1/1–1/5, April 1990.

R. W. Earnshaw and P. Mars, "Footprints for Mobile Communications," in *Proceedings of the Eighth IEE UK Teletraffic Symposium*, pp. 22/1–22/5, April 1991.

R. W. Earnshaw, *Simulation of Packet- and Cell-based Communication Networks.* PhD thesis, University of Durham, UK, May 1992.

R. M. Falconer, J. L. Adams, and G. M. Walley, "A Simulation Study of the Cambridge Ring with Voice Traffic," *British Telecom Technology Journal*, vol. 3, April 1985.

R. M. Falconer and J. L. Adams, "Orwell: A Protocol for an Integrated Services Local Network," *British Telecom Technology Journal*, vol. 3, October 1985.

R. M. Fujimoto, "Parallel Discrete Event Simulation," *Communications of the ACM*, vol. 33, pp. 30–53, October 1990.

L. Gould, I. Bowler, and A. Purvis, "Real-Time, Multi-Channel Digital Filtering on the Transputer," in *Proceedings of the 1989 International Symposium on Computer Architecture and Digital Signal Processing*, 1989.

R. Händel and M. N. Huber, *Integrated Broadband Networks: An Introduction to ATM-based Networks.* Addison-Wesley, 1991.

D. P. Helmbold and C. E. McDowell, "Modeling Speedup (n) Greater than n," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, pp. 250–256, April 1990.

A. Hind, "A Multiprocessor Testbed for Parallel Simulation of Telecommunication Networks," technical report, University of Durham (SEAS), December 1990.

A. Hind, "Parallel Simulation for Performance Modelling of Telecommunication Networks," in *Proceedings of the Eighth IEE UK Teletraffic Symposium*, pp. 9/1–9/6, April 1991.

H. T. Mouftah and R. T. Sturgeon, "Distributed Discrete Event Simulation for Communications Networks," *IEEE Journal on Selected Areas in Communications*, vol. 8, pp. 1723–1734, December 1991.

S. J. Nichols, *Simulation and Analysis of Adaptive Routeing and Flow Control in Wide Area Communication Networks.* PhD thesis, University of Durham, UK, March 1990.

S. Toueg and J. D. Ullman, "Deadlock-Free Packet Switching Networks," in *Proceedings of the ACM Symposium on the Theory of Computing, Atlanta, Georgia*, pp. 89–98, May 1979.

AUTHOR BIOGRAPHIES

Richard W. Earnshaw received the degree of B.Sc. in Applied Physics from the University of Durham, UK, in 1987 and the degree of Ph.D. from the University of Durham, UK, in June 1992. His research interests include performance engineering of telecommunication networks, particularly broadband ISDN and mobile systems, and parallel discrete event simulation. He is currently a Research Assistant in the Computing Laboratory at the University of Cambridge, UK, where he is engaged on the *Fairisle* fast packet switching project.

Alan Hind received the B.Sc. degree in Electronic Communications from the University of Salford, UK, in 1982. He is currently the British Telecom Research Fellow in Parallel Simulation in the School of Engineering and Computer Science at the University of Durham, UK. His research interests include parallel discrete event simulation and the performance engineering of telecommunication networks. He is a member of the Institute of Electrical and Electronic Engineering, the Association of Computing Machinery and the Society for Computer Simulation and an associate member of the Institute of Electrical Engineers in the UK.