Conservative Simulation of Load-Balanced Routing in a Large ATM Network Model

C. D. Pham H. Brunst* S. Fdida

Laboratoire LIP6
Université Pierre et Marie Curie
4 place Jussieu 75252 Paris Cedex 05
{Cong-Duc.Pham, Serge.Fdida}@lip6.fr

Abstract

With the emerging of broadband networks based on ATM technology, performance evaluation tools that allow the study of large systems are desperately needed. In this article we present our experiments in distributed simulation of large and complex ATM network models with a conservative simulator. The goal here was not to achieve the maximum speedup with well-shaped topologies but rather to see what speedup can be obtained with a realistic model on a "state-of-the art" parallel computer. A network model with 78 switches is simulated on a Cray T3E using 3 different traffic loads. The performance results show that good speedups can be achieved but also highlight partitioning problems and bottlenecks in the simulation model that can seriously limit the speedup of realistic model simulations.

1 Introduction

With the emerging of Asynchronous Transfer Mode (ATM) technology the performance evaluation of future large scale networks based on ATM is a great challenge. It is mandatory to evaluate carefully these networks because they typically introduce a new environment where both speed and throughput are increased. Making these new networks operational is not an easy task because problems such as congestion avoidance, traffic control and optimal routing algorithms should be addressed first. It is not only necessary to evaluate new algorithms but also the existing ones that perform well in actual networks but may fail in a high-speed environment. Performance evaluation with analytical methods is limited because restrictive hypotheses reduce the usefulness of the results. Sequential simulation has been widely used but the computation load required to simulate large models is now beyond the scope of such an approach. Since simulation remains the best way to study complex systems in detail, distributed simulation techniques have been proposed during the last 2 decades. These methods fall in two categories: the conservative approach [3] and the optimistic approach [11] depending on how the synchronization is performed.

Although many issues on simulation of complex communication network mechanisms have already successfully been achieved, most of them were done with sequential simulators. In a number of previous studies using distributed simulation techniques [6, 13, 4, 5, 2], relatively few really deal with general topologies, realistic complexity and low granularity. In this paper, we describe our experiments in distributed simulation of a large and complex ATM network model at the cell-level with a conservative distributed simulator. The goal here was not to achieve the maximum speedup with well-shaped topologies but rather to see what speedup can be obtained with a realistic model. The network model implements the routing algorithm described in [10] with dynamic state-dependent routing functions that provide load-balanced routing features. The simulation of a routing algorithm implies to simulate (i) the mechanism that consists in constructing and updating the routing tables (link costs can change due to link failures for example) and (ii) the flow of cells transported by the network.

*This work was realized while the author visited the LIP6 laboratory for an 8 months period as part of his Master degree from Aachen University, Germany.

This paper is organized as follows: the next section presents an overview of the ATM network model and the routing algorithm. Section 3 presents the distributed simulator. Section 4 discusses the experimental results and Section 5 finally gives conclusions.

2 Description of the model

The ATM technology is intended to provide large throughput and high-speed transfer between nodes by using small cells and switching components. Nowadays, ATM can be found in Local Area Networks and starts to appear in Metropolitan Area Networks. The next step is to develop it for large scale configurations. We propose to study the topology illustrated by Fig. 1. This 78-switch network model has been obtained by linking three 26-switch networks (bold links in figure 1) used by the ATM Forum for the specification of the PNNI (Private Network-to-Network Interface) routing algorithm [1]. In each sub-network, we added 10 sources (S_0 to S_6 and S_4^1 to S_6^1) and 7 sinks (D_0 to D_6) to model the traffic of cells at the network interfaces.

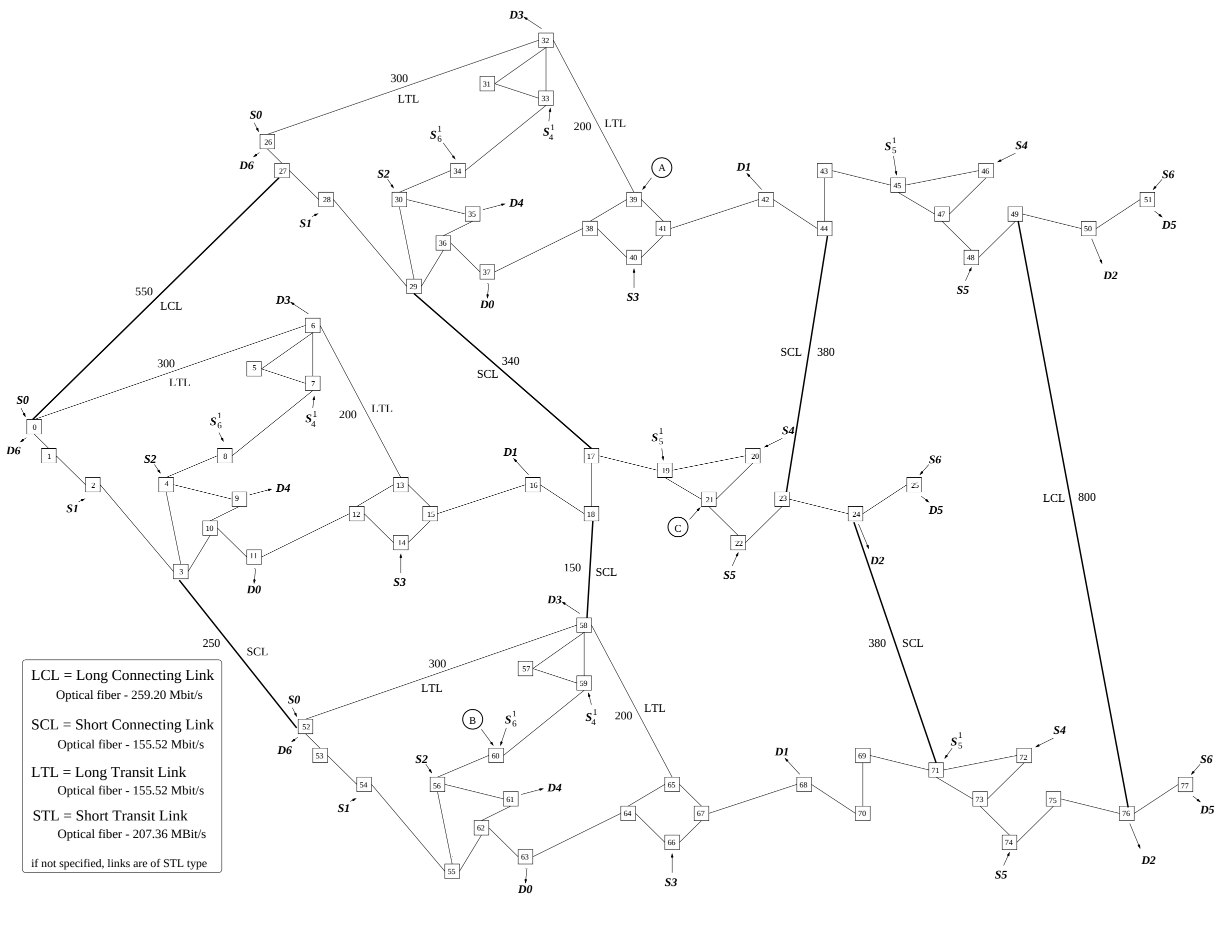

Figure 1: **Network model with 78 switches.**

These sources each set up a permanent Constant Bit Rate (CBR) connection to the corresponding sink (subscript number) until the simulation ends. In addition, 3 traffic sources called A, B and C repeatedly choose one destination among the 21 possible destinations for a temporary connection. Traffic sources execute a `setup` procedure with their corresponding destination prior to any data transfer. We will assume only one connection at a time per traffic source. All along the path of the connection, a new entry is added to the connection table of each intermediate ATM switch to store the connection identifier and the output link for this new connection. During the setup procedure a switch can decide to reject the call if the resources available are not sufficient (`release` message). Upon reception of the `setup` message, the final destination can either send back a `connect` message or a `release` message. At the moment, the admission control works as follows: each CBR connection indicates a *Peak Cell Rate* (PCR) that is reserved in each output link along the way to the sink. If the sum of all the PCRs allocated at a given link has exhausted the link capacity an additional call can not be routed via this link. A call is rejected at a switch if no output link with sufficient resources can be found.

2.1 The routing algorithm, link cost and routing functions

A routing algorithm consists of calculating the best path for the packets/cells from a source to a sink. This calculation is done by considering either implicit criteria such as the number of hops or explicit ones given by the user. In this paper, we are interested in the routing algorithm proposed by Jaffe and Moss in [10]. This algorithm belongs to the *vector-distance* category and by the authors is said to have a quick recovery from cost changes property (this is the main reason for the choice of this algorithm). Transient loops are avoided by a synchronized exchange of cost increase information and acknowledgments.

The metric we used for routing is the cost of a link represented by an integer ranging from 0 to 100, 100 being the highest cost. Routing functions are used to determine in real time the cost of a given link based on its type and instantaneous load. The type of a link varies with its maximum throughput, its length, its function in the network (connecting or transit) and its access medium (e.g. twisted pair, coaxial, optical fiber, radio...). In our model we will assume that optical fibers are available everywhere in the network. Four types are then defined in the model: Short Transit Link (STL) and Long Transit Link (LTL) for the interconnection of switches in the same sub-network, Short Connecting Link (SCL) and Long Connecting Link (LCL) for the interconnection of sub-networks. A link is said to be long if its length is greater than 150km for a connecting link and 50km for a transit link. For each type of link, a routing function is defined in order to calculate the cost of the link and to balance the connections over a larger number of links. These functions have a generic shape with 4 thresholds as illustrated by Fig. 2.

C_S, C_L and C_N are respectively the cost for saturated, loaded and normal state. $T_{L\to N}, T_{N\to L}$, $T_{S\to L}$ and $T_{L\to S}$ are respectively the thresholds, in percentage of the effective throughput, for going from loaded state to normal state, from normal state to loaded state, from saturated state to loaded state and from loaded state to saturated state.

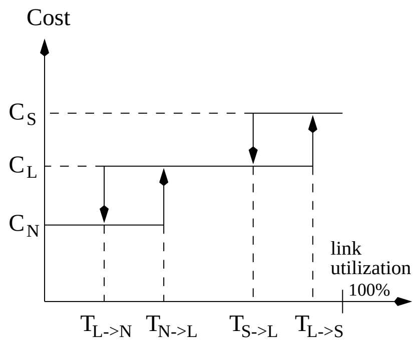

Figure 2: The generic routing function.

At the moment, the calculation of the link cost is based on the PCR given during the connection phase. Each time a new connection is accepted and routed on a given link, the cost (and the state) of the link may change in which case the switch sends cost update information to its neighbors. The routing tables may then change but new cheapest links will only be taken into account for new connections. The temporary connections set up by sources A, B and C will cause many link cost changes and produce very frequent updates of routing tables.

2.2 Model's properties

Even if the topology is fictive, it is general enough to represent most of real topologies with different types of links and non homogeneous throughput. The internal delays of switch and link models can be set accordingly to reproduce the average behavior of a real system. For this study, our switch model is non-blocking with buffers located at the output ports. The average delay for switching a cell from an input port to an output port is computed depending on the throughput of the switch. Each output port contains 2 queues for high and low priority traffic. Scheduling is done in a FIFO manner within each queue. Links are characterized by a throughput and a propagation delay. In the network model, permanent CBR connections model the long term load in the network while temporary connections model the cross-traffic.

From the distributed simulation perspective, this model has several challenging properties to mention. First, the simulation of a shortest-path routing algorithm on a general topology creates disparities between links. Many of them may not be often used (links depicted in the figure are bi-directional even if for sake of simplicity a single link has been drawn). With a conservative method, a number of null-messages are needed to propagate the simulation time. We think that this situation is a highly realistic one. The second property is a consequence of the first: by concentrating a large part of the traffic on a given path bottlenecks can be created in the simulation model even if there are no such bottlenecks in the real system. Bottlenecks in the real system may be avoided by upgrading some network components, but in most cases for the distributed simulation, it will be hardly possible to increase the speed of the processor that simulates the overloaded part of the network. Third, in addition to permanent connections dynamic sources with random destinations introduce frequent changes to the link's load resulting in complex load-balance patterns. Finally, the simulation of high-speed networks at the cell-level requires a large amount of events. 1 second of the real system may need the simulation of millions of event.

3 The distributed simulator

The first goal was to develop a tool with practical relevance for simulating ATM network models at the cell-level. Application specific simulators are often developed close to the problem. Our approach is general enough to simulate most topologies of ATM networks. However, the internal structure of the switches and the protocols can not be configured from the outside. The simulator offers a unified framework for parallel and sequential implementation so the same source code can either be run sequentially or in a distributed manner. This method has the advantage to allow fair speedup measures, easy debugging and automatic modification of the two versions. Interprocess communication is implemented using the SHMEM library on the Cray.

To hide the complexity of distributed simulation techniques and to achieve the best performance, some optimizations have been integrated in the simulator. Although implementing a conservative simulator is not a new topic and is beyond the scope of this paper, we will shortly describe the main issues we think to be the most relevant in our simulator.

3.1 Management of null-messages

The wish to deal with very general topologies and large scale models introduces the need to handle a large number of Input Channels (ICs) per Logical Process (LP). In general, in communication network models, the larger is the model, the larger is the number of links. All these links are far from being used at the same time. In most cases, the network traffic will be aggregated in only some specific parts of the networks to produce stressed scenario. Therefore, a lot of links can be very lightly used. In order to manage a large number of ICs and to reduce the overhead associated to null-messages, our ICs have some little modifications compared to the original scheme: each IC i has a clock $last_i$ that ticks according to the timestamp of the last received message instead of a top_i clock that reflects the timestamp of the topmost message in the channel. In [12], $last_i$ is called the Earliest Output Time of the sending LP. Null-messages are not stored in the ICs but simply change the value of $last_i$ and therefore do not significantly burden the simulation.

3.2 Unique event queue

To avoid the scanning overhead of empty input channels, an unique event queue is used to store both remote and local events. Actually there are no ICs at all since incoming messages are directly put in the unique queue, no matter where they come from. A time window is defined by taking the minimum of all the $last_i$. This minimum is called the Earliest Input Time (EIT) in [12]. We will use the same terminology. Unsafe and external events are discarded by the time window rather than storing them into separate buffers. In this way, the scanning of empty channels is avoided since the time window is automatically increased when new messages arrive. Message eligible for execution will automatically be discovered as the upper limit of the time window increases.

3.3 Lookahead and generation of null-messages

Only the lookahead represented by the link's propagation delay is exploited in our simulator. It is expected that this delay is much greater than the switch internal delays so the amount of lookahead appears to be relatively high. This means that at the moment the amount of lookahead is not tailored for any particular ATM mechanism because the link's delay can always be used. Performance may be better with an application-specific lookahead computation but the transparency would decrease. Since no assumptions are made on the scheduling strategy of the switches, it is easy to simulate non FCFS scheduling (e.g. complex dynamic rate control). Null-messages are generated when all eligible messages have been processed, i.e. all messages have a timestamp greater than EIT.

4 Performances of the simulator

The simulation time is expressed in time units (tu). The corresponding real time value can be obtained by defining a time unit as follows: it is the time needed to switch a cell from an input port to an output port given a base throughput T_b. For the network model depicted in Fig. 1, T_b is set to the throughput of the most rapid link (LCL links at 259.20 Mbit/s) in order to avoid time increments less than 1 tu. In this case, 1 tu represents $\frac{8*53}{259.2*1024*1024} = 1.56\mu s$ of real time (8 bits per byte and 53 bytes per ATM cells). All switches process cells at the rate of 155.52 Mbit/s per input link. STL links have a throughput of 207.36 Mbit/s, LTL and SCL links have a throughput of 155.52 Mbit/s. The propagation delay is set to 125 tu for all SCL links. For the other links a different value is indicated in figure 1. With 1 tu representing 1.56 μs, 100 tu represents approximately 32km of optical fiber when 2/3 of light-speed is assumed for the propagation. The initial load for all links is set to zero.

In order to evaluate the ability of the conservative simulator to handle general topologies and the null-messages due to unused links, we define 3 test cases using a low, a medium and a high traffic load. The low load uses only the three A, B and C sources. The medium load adds the CBR sources from S_0 to S_3 in each sub-network with 40Mbit/s connections. In the high load, the remaining CBR sources are also sending cells at the rate of 48Mbit/s. Sources A, B and C are defined with a connection time of 25000 tu, an idle time of 500 tu and a PCR of 50Mbit/s. By considering the average number of cells simulated, the medium load produces 3.91 times more cells than the low load. The high load is 2.19 times heavier than the medium. With these parameters and the ones we chose for the routing functions, an STL link can only carry 2 connections before switching to a loaded state. 2 more connections and the STL link becomes saturated.

4.1 Experimental results

The simulations are run on a Cray T3E provided by the *Institut du Développement et des Ressources en Informatique Scientifique* at Orsay, France. The Cray T3E consists of 256 processing nodes (PNs) interconnected by a very low latency 3D torus network. Each node has 128 MB of local RAM memory and runs in dedicated mode (one job) at 300MHz. Sources and sinks connected to a switch are never separated and are simulated by the same LP. With a mapping file, several switches can be grouped in one LP and scheduled with a sequential algorithm.

The simulations run for 500,000 time units (0.78s of the real system). More than 53 millions of real events are simulated for the high load (more than 17 millions of cell exchanges between the switches). Table 1 shows the null-message ratio (NMR), the remote message ratio (RR), the speedups and the efficiency (eff) computed as speedup/np. The speedup values are obtained by comparing the overall execution time of the sequential version running on one node of the parallel machine to the execution time of the parallel version. In addition to the simple one switch to one processor mapping, 5 mappings using 64, 32, 16, 8 and 4 processors are defined with a simple neighbor aggregation method: we group several adjacent switches and simulate them on the same processor. A sequential-like simulation is realized for all the switches in the same physical processor and conservative synchronization is used between switches on different processors.

	np.	*NMR*	*RR*	*time(s)*	*speedup*	*eff*
LOW	1	-	-	37	1.0	1.0
	4	0.075	0.157	25	1.48	**0.37**
	8	0.244	0.221	20	1.85	0.23
	16	0.402	0.330	16	2.31	0.14
	32	0.370	0.484	12	3.08	0.09
	64	0.451	0.789	11	3.36	0.05
	78	0.511	0.892	10	**3.7**	0.04
MEDIUM	1	-	-	204	1.0	1.0
	4	0.072	0.159	62	3.29	**0.82**
	8	0.157	0.275	40	5.1	0.63
	16	0.254	0.374	26	7.84	0.49
	32	0.489	0.534	21	9.71	0.303
	64	0.528	0.768	18	11.33	0.177
	78	0.507	0.869	16	**12.75**	0.163
HIGH	1	-	-	488	1.0	1.0
	4	0.038	0.135	131	3.72	**0.93**
	8	0.064	0.2	72	6.77	0.84
	16	0.159	0.31	46	10.60	0.66
	32	0.405	0.466	33	14.78	0.46
	64	0.467	0.768	31	15.74	0.24
	78	0.394	0.861	25	**19.52**	0.25

Table 1: Experimental results

The null-message ratio is defined to be the ratio between the number of null-messages sent and the number of real messages sent. In our model, the connection-oriented nature of the traffic can introduce unused links so the null-message ratio is different for each LP. The values indicated in table 1 have been obtained by dividing the total number of null-messages sent by the total number of real messages sent in the simulation. The NMR decreases as the number of processors decreases. As the traffic load increases, less null-messages are produced since more real events are available. For a small number of processors where several switches are grouped in one LP, the NMR is kept low because many events are available in the same time window. The rate of growth of the time window varies from one load to another and typically depends on what is simulated. What is really important is to verify that even with a large number of processors, i.e. few switches per LP, the NMR does not increase dramatically in which case it would have meant a too small lookahead value. As can be seen, a maximum value of 0.551 for the NMR is obtained for the low load with 78 processors. This is still reasonable as regard to the very low number of used links. This result confirms the efficient use of the link's delay for lookahead computation. Usually, the link's delay is expected to be a hundred times greater than the internal delays of the switches.

Normally, the NMR increases as the number of processors increases. This is true for the low load but not for the one-switch-per processor simulation of the medium and high load. In order to explain why the NMR decreases for the 78-processor execution, we plotted in figure 3 the number of events processed in each processor for the 32, 64 and 78-processor mappings. In the 32-processor mapping processor 24 simulates switch 57, 58 and 59. In the 64-processor mapping, only switch 58 and 59 are grouped and simulated by processor 49. This processor has the highest load. In the one-switch-per processor execution, processor 58 that now simulates only switch 58 has the highest load. The explanation we have for the decrease of the NMR is that as a larger number of processors is used, some previously overloaded processors become less loaded and therefore can process and send real events more quickly, reducing the need to send null-messages.

In our experiments, we first ran the simulations with the highest possible number of processors and then reduced the number of processors. At the beginning, we thought that the aggregation of switches would help to reduce the communication cost by avoiding remote messages between switches on the same LP. We found that using less processors in this simple way (neighbor aggregation) provides no better speedup but only better efficiency (the efficiency is maximized with 4 processors for all loads). Actually, when aggregating adjacent switches on the same physical processor, the number of outgoing communication links in an LP often remains the same either for the case where an LP simulates one switch or for the case where the LP simulates several switches. If the partitioning cuts can be done on lightly loaded links, it may save some communication overheads. However, if in order to cut on lightly loaded links, one has to put a lot of switches together then the amount of local computation may also saturates the processor creating bottlenecks in the simulation.

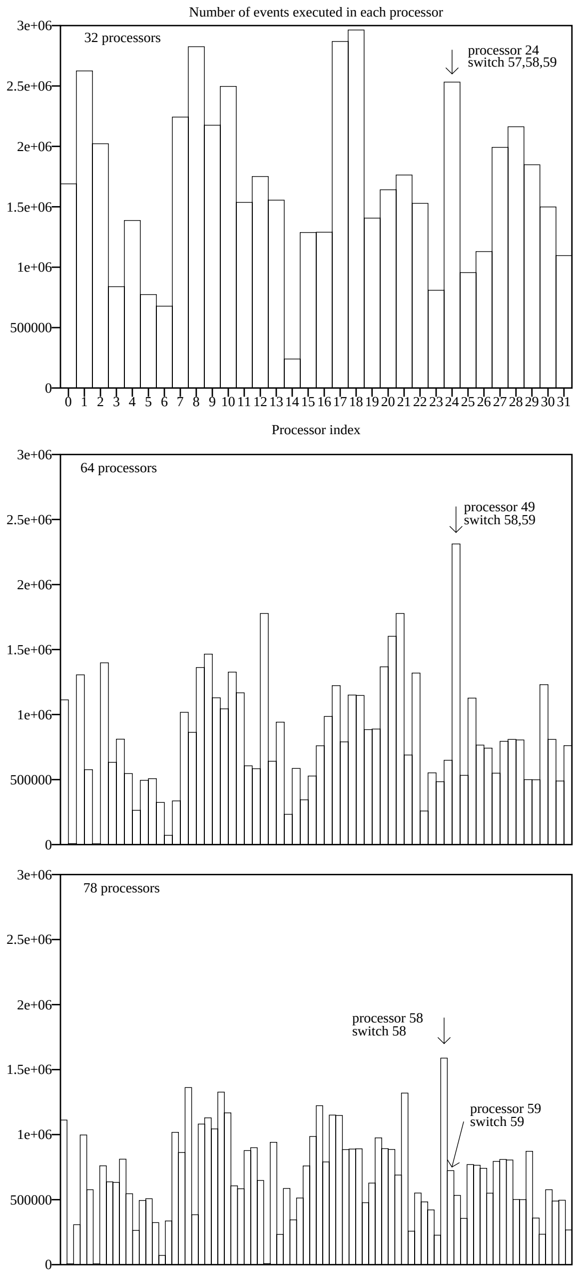

Figure 3: Number of events per processors.

When many links are heavily loaded, it is most likely that an LP will keep the same number of remote messages whatever the partition is. This is because all incoming cells are supposed to be sent out in the network model. As the number of cells sent on a given communication link is not affected by the logical mapping, the aggregation of switches increases the local computation load of the processor and also keeps the same overhead for sending external messages. Although the routing algorithm we simulate aims at balancing the connections over the network, the load among the processing nodes is very badly balanced. The reasons for this load imbalance can directly be related to similar problems encountered in real communication network systems such as bottlenecks and congestion. Based on the results depicted in figure 3, we tried to improve the load balance by changing the initial mapping but unfortunately, we obtained no better speedups. The big problem we can foresee is when more connections are routed through a single switch. In this case, even the one-switch-per processor mapping may create a bottleneck and the overloaded processor can slow down the entire simulation. Going beyond the one-switch per processor to n-output-queues per processor may be a solution as discussed in [9] but the performance of very fine-grained LPs on a large number of processors still need to be investigated.

The remote message ratio shown in table 1 has been obtained by dividing the total number of real messages sent off processor by the total number of real messages exchanged between switch objects. As the number of processors increases, the RR increases significantly. With 78 processors, the RR is maximized. Normally, all incoming messages should be forwarded since they typically represent the reception of data cells. The reason why we do not obtained a RR of 1 for the 78-processor mapping is because some special cells for the routing algorithm do not need to be forwarded. As the RR increases the synchronization overhead increases as well. Our considerations are based on the average time t_a the simulator needs to process an event. We calculate an approximation of this value by simply dividing the execution time of a sequential simulation run by the overall number of processed events. As there is no synchronization in the sequential version of the simulator, this value represents the average computation time needed to process one event, including the overhead for context switching and list management. For the parallel version, we calculate for each processor the theoretical amount of processing time by multiplying the number of events simulated by the processor by t_a. Actually, the average processing time for an event in the parallel version may be smaller than t_a because there are less overheads for context switching and list management. However, the difference to the execution time of the parallel simulator can still be regarded as the overhead for synchronization purposes. Figure 4 depicts the overhead for the simulation of the 78-switch model with the medium load on 4, 8, 16, 32, 64 and 78 processors. The boxes graphically represent the number of events processed by the processor with the highest and lowest load (best and worse case). The square-dashed curve illustrates the overall execution time while the diamond-dashed and plus-sign-dotted curves stand for the theoretical execution time needed for event processing for the most loaded and the least loaded processor respectively.

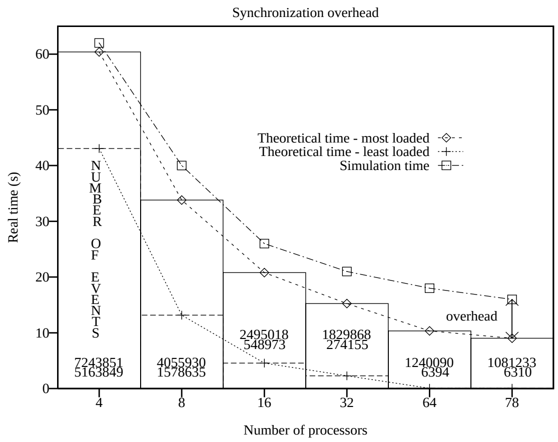

Figure 4: Synchronization overhead.

The vertical distance between those two curves and the square-dashed curve can be regarded as the synchronization overhead (waiting and sending protocol messages). For 4 processing nodes, the overhead for the most loaded processor stays below 5% while using 78 processors cause an overhead of 43% for the best case. The least loaded processor has an overhead of more than 99%! For the least loaded processor, the synchronization overhead starts to be significant since 4 processors.

As expected, the speedup increases as the load increases. More load means more events at the same simulation time and therefore more parallelism. A higher load also forces the routing algorithm to better distribute the network load on a larger number of links. For the medium and high load, almost linear speedups can be achieved with 4 and 8 processors resulting in a high efficiency. For the low traffic, using 32, 64 or 78 processors leads to very similar execution time. This result shows that a kind of limit for the speedup has been reached before the maximum number of processors is used. Actually, the processors do not have enough real work to do in the low load. However, the most interesting point to observe is that even with a large synchronization overhead, the speedup does not decrease. It is quite surprising to see that the maximum speedup has always been obtained when both NMR and RR are high. We are also positively surprised to see that even for the low load the distributed execution can perform better than the sequential execution. The answer resides in the very low communication cost on the parallel computer we use and this behavior can not be generalized. In slower communication architecture, using so many processors would certainly decrease the speedups dramatically. Actually, the very low latency provided by the SHMEM library on the Cray T3E makes the sending of null-messages cheaper than trying to increase the local computation. With very low communication costs, the simulation time can be artificially advanced without penalizing too much the simulator. These experiments and previous tests on smaller networks show that the one-switch per processor mapping has always given the maximum speedup. We are not sure yet that using less processors than switches can provide better speedups than the simple one to one mapping on the Cray. At the moment the only advantage we see is the possibility to work in an environment where the number of processors is a limitation.

5 Conclusion

In this paper we presented a conservative simulation of a large and complex ATM network model. With a realistic network topology that consists of 78 switches, a maximum speedup of 19.52 can be achieved with 78 processors. Using less processors provides no better speedup. However this result is very particular to the low communication cost on the Cray T3E. Using as many as processors than switches is a very extreme case and the results are only provided here for a better understanding of the parallel simulation behavior on a machine with a very low communication cost. Practically, as systems with a large number of processors are not accessible to the majority of researchers nowadays, parallel simulations are likely to be executed on a smaller number of processors with higher communication costs. For our model, using more than 32 processors dramatically increases the overhead for synchronization.

From these experiments, the lookahead value provided by the link's delay seems to be enough to achieve good performance. What really limit the speedup in network simulation are rather the partitioning problem and the bottlenecks than the lack of lookahead. In our experiments, we failed to obtain a better partition than the simple one-switch per processor mapping. Once again, we think that this is due to the low communication latency on the Cray T3E. The bottlenecks may actually limits the speedup but in order to avoid them, we must have a simulation granularity lower than that of a switch object, i.e. a switch may be simulated by more than one processor. This solution may be viable with very low communication costs. With the actual granularity, a higher speedup might be difficult to obtain.

Acknowledgments

The authors would like to thank the anonymous referees for their helpful comments and suggestions. Examples of routing functions have been provided by the CELAR. Special thanks to J. Essmeyer who developed the first version of the simulator.

References

[1] ATM Forum, PNNI v1.0 Specification

[2] R. L. Bagrodia et al., "Parallel Simulation of a High-Speed Wormhole Routing Network", *Proc. of PADS'96, pp47-56.*

[3] K. M. Chandy and J. Misra,"Distribution Simulation: A Case Study in Design and Verification of Distributed Programs", *Trans. on Soft. Eng., Vol. 5(5), pp440-452.*

[4] C. D. Carothers et al., "Distributed Simulation of Large Scale PCS Networks", *Proc. of MASCOT'94, pp2-6.*

[5] J. G. Cleary and J. J. Tsai, "Conservative Parallel Simulation of ATM Networks", *Proc. of PADS'96, pp30-38.*

[6] Earnshaw, R. W., Hind, A., "A Parallel Simulator for Performance Modeling of Broadband Telecommunication Networks", *Proc. of the WCS'92, pp1365-1373.*

[7] A. Ferscha and S. K. Tripathi, "Parallel and Distributed Simulation of Discrete Event Systems", *CR-TR-3366, Dep. of Comp. Sci., Univ. of Maryland, August 94.*

[8] R. M. Fujimoto, "Parallel Discrete Event Simulation", *Comm. of the ACM, Vol. 33(10), pp31-53.*

[9] F. Hao et al., "Logical Process Size in Parallel Simulations", *Proc. of WSC'96, pp645-652.*

[10] Jaffe, J. M., Moss, F. H., "A Responsive Distributed Routing Algorithm for Computer Networks", *IEEE Trans. on Comm.,* **30***(7), pp1758–1762.*

[11] D. R. Jefferson, "Virtual Time", *ACM Trans. on Prog. Lang. and Sys., Vol. 7(3), pp405-425.*

[12] V. Jha and R. L. Bagrodia, "A Unified Framework for Conservative and Optimistic Distributed Simulation", *Proc. of PADS'94, pp12-19.*

[13] Mouftah, T., Sturgeon, R. T., "Distributed Discrete Event Simulation for Communications Networks", *IEEE JSAC,* **8***(9), pp1723-1734.*