MODELING ATM NETWORKS IN A PARALLEL SIMULATION ENVIRONMENT:
A CASE STUDY

P. Gburzynski T. Ono-Tesfaye S. Ramaswamy
Department of Computing Science
University of Alberta, Edmonton, Canada T6G 2H1

Keywords: Parallel simulation, Time Warp, ATM networks, Protocol modeling.

Abstract

This paper describes our experience with implementing an Asynchronous Transfer Mode (ATM) network simulator in the high-level sequential programming language SMURPH [Dobosiewicz and Gburzynski 93], and then porting it to SimKit—a parallel simulation tool offering a C++ interface to a shared-memory implementation of the well-known Time Warp [Jefferson 85] concept. The work was undertaken as part of the TeleSim project, which aims to build a set of multi-purpose high-fidelity ATM simulation tools for execution in sequential and parallel environments.

1 Introduction

ATM is a connection-oriented packet-based switching technology designed for high-speed networks. Designing a simulator for ATM networks is not a well-defined task, because ATM is a networking concept rather than a single networking solution. Owing to the lack of a complete specification, our network model is made up of two distinct components with different requirements: a model of the generic ATM network hardware, i.e., the hardware components like buffers, links and switch fabrics, and a model of ATM signaling, which constitutes a well-established part of the ATM standard. [Arlit et. al. 94] have developed a number of realistic traffic sources and sinks that interface with our model. The simulator currently features

- support for arbitrary switch dimensions and network topologies.
- 3 single-stage and 3 multi-stage switch fabrics. Because switches with different switch fabrics implement the same signaling protocol, they can be freely interconnected.
- separate buffering of cells on CBR, VBR, ABR and UBR connections.
- support for point-to-point switched virtual channels (SVCs) and permanent switched virtual channels (PVCs).
- usage parameter control (UPC) at all switch inputs.
- support for switch-to-switch virtual paths (VPs).
- extensive switch-level statistics collection.
- support for the explicit forward congestion indication (EFCI) bit in the cell header.
- The routing table used at switches to route connections can be either computed by a built-in function (based on the topology) or supplied by the user.
- Signaling messages may be several cells in length. Segmentation and reassembly of these messages may have a significant effect on the processing time for a signaling message. Hence, the segmentation and reassembly of signaling messages is modeled. All signaling messages are assumed to be of the same size.
- Signaling messages for a switch are placed in an infinite queue and serviced FIFO. This is useful when the user is interested in the connection setup delay, for example.
- The signaling protocol does not take into account link or node failures. It does not have the ability to set up point-to-multipoint connections.

2 Model Overview

Time Warp simulations require that the computation be broken up into logical processes (LPs) which communicate via time-stamped messages. LPs always execute whichever incoming message has the lowest time stamp, disregarding any need for synchronization. The local virtual time (LVT) of the process is set to the time stamp of the message being handled. Processes regularly save their local state in a *saved state vector*. A causality error occurs when a message with a lower time stamp than the LVT (a *straggler*) is received. This forces the affected process to roll back in virtual time and restore a state saved before the time stamp of the straggler.

We initially implemented a prototype network model in SMURPH. SMURPH is an object-oriented sequential

simulation language that provides high-level structures like stations, links and ports. The SimKit C++ interface to Time Warp provides two simple base classes: an LP class and an event (message) class. The interface is sufficiently general to be identical in sequential and Time Warp-based parallel environments. The mapping of SMURPH objects to SimKit LPs and non-LP objects was an important issue in the process of porting the network model. Design trade-offs were needed to maintain as much of the parallelization potential as possible, while at the same time keeping sequential overheads low: fragmenting the simulation into too many LPs would increase the sequential overhead, whereas having too few LPs would reduce the parallelization potential (consider the extreme case where the entire simulation is executed by a single LP). The peculiar nature of Time Warp memory also had to be considered: all data that can change during the simulation has to be part of an LP's state.

Primary objectives in the hardware model design were extensibility and both good sequential and parallel performance. The number of connections in a typical simulation run is small compared to the number of cell transmissions, however, and performance was therefore not the main goal in designing the ATM signaling model. Instead, we focussed on a reasonably detailed implementation of the ATM Forum signaling specifications.

Broadly, our model consists of two main kinds of objects: *LP objects* and *switch objects*. Every switch object creates and manages a number of LP objects. In addition to LPs, switches have a number of constant read-only data fields, for instance the switch's LP interconnect structure. Statistics are variable data and are therefore collected by LPs.

The model is made up of links, traffic sources/sinks (TSSs), end nodes and switches. Links are bidirectional and connect switches with switches and switches with end nodes. Each end node is connected to a switch by a single link; and every TSS is associated with at least one end node (one way to understand this is to view the end node as a workstation with an ATM card and the TSS as the application software running on it).

The cell transfer phase is the main simulation phase during which connections are set up/cleared, cells are sent/received, and statistics are collected. The result of a connection setup is that the input port switching tables along the call route are updated and that the network layer of every node along the route keeps track of resources allocated for the call.

There are two basic connection types in ATM-TN: permanent virtual circuits (PVCs) and switched virtual circuits (SVCs). To establish an SVC, a traffic source must request a call setup by sending a message to the network layer of its end node. A traffic source can send cells on an SVC only if the connection setup request is successful. Resources for PVCs are allocated during model initialization. A traffic source can send cells on a PVC at any time.

3 Switch Hardware Model

The LP classes are *input LPs*, *output LPs*, *fabric LPs* and *signaling LPs*. ATM cells are SimKit events sent from LP to LP.

The input process makes the switching decisions. When a cell arrives at an input LP, it looks up the cell's VPI/VCI values in a switching table, changes the cell's VPI/VCI fields if necessary, and *tags* the cell with the output port index and the connection's quality of service (QoS) class. If the cell does not conform to the connection's traffic descriptors, its loss priority bit (CLP) is set. Depending on the cell's tag and the switch's (read-only) interconnect structure, the cell is sent either directly to the output LP, to a fabric LP (for example, a shared buffer), or to a signaling LP (if the cell arrived on a signaling channel). The output LPs buffer cells if necessary and then send them to the input LP on the other end of their link.

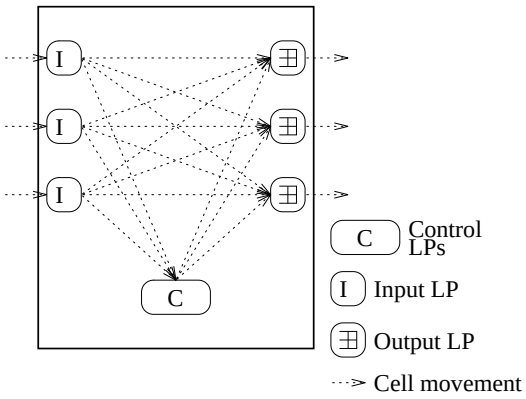

Figure 1: 'Perfect' Switch

While the use of input LPs decreases performance, it adds flexibility to the model. In particular, it allows switches to operate completely independent of each other: a switch need not know anything about the internal structure of the switches it is connected to. Note that in this model, signaling cells are treated just like any other cell by the input, output and fabric LPs. This means that to experiment with different signaling protocols, the researcher only needs to modify the signaling LPs.

All switch models use the same generic buffer architecture. Each buffer consists of four FIFO queues, one FIFO for each of the four traffic classes CBR, VBR, ABR

and UBR. Each FIFO has three parameters that can be specified independently: the *size*, the *CLP=1-threshold* and the *EFCI-threshold*. The CLP=1-threshold specifies the FIFO occupancy level beyond which cells with their CLP bit set (i.e., low-priority cells) are dropped. The EFCI-threshold specifies the occupancy level beyond which the EFCI bit of buffered cells is set. The order in which the four FIFOs of one buffer are served can be set to either round-robin or to exhaustive priority in the order CBR, VBR, ABR, UBR.

A brief description of each switch class follows.

Perfect Switch. This is a simple single-stage output-buffered switch with per-port buffer memory (fig. 1). The fabric is perfect in the sense that every cell arriving at an input port makes it to an output buffer without getting dropped and in constant time. Various realizations are possible for small switch dimensions (say ≤ 16); many of the switches currently on the market fit in this category.

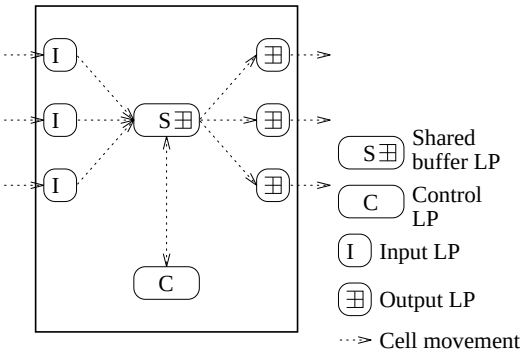

Figure 2: Shared Buffer Switch

Shared Buffer Switch. This is a single-stage switch with a shared buffer fabric (fig. 2). The shared buffer is modeled by a single LP. Like all our switch models, the switch also has buffers at the output ports. Every cell arriving at an input port is first sent to the shared buffer and then to its final destination (an output port or the control module). Cells are buffered in the shared buffer if they can't be transferred immediately. The rate at which cells can be transferred from the shared buffer to a specific output port is given by the switch's *internal transmission rate*; our model assumes that cells can be transferred to several different output ports in parallel.

Crossbar Switch with buffers at the crosspoints (fig. 3, [Ahmadi and Denzel 89]). Every cell arriving at an input port is first sent to the buffer at the crosspoint of the input port and the cell's destination output port. Crosspoint buffers associated with one output link are managed by one crossbar LP. Cells are buffered at the crosspoint buffer if they can't be transferred immediately. The rate at which cells can be transferred from

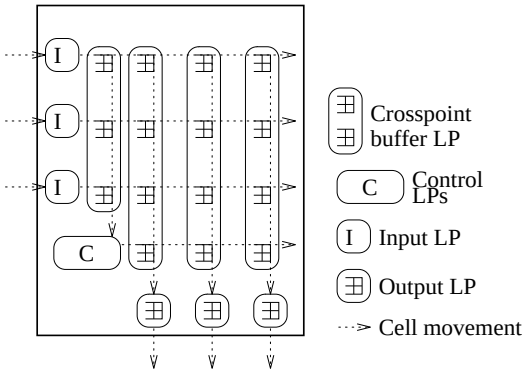

Figure 3: Crossbar Buffer Switch

crosspoint buffers to a specific output port is given by the switch's *internal transmission rate*; the crosspoint buffers leading to the same output port are served in round-robin fashion. Because there are dedicated output lines for each output port, our model assumes that cells can be transferred to several different output ports in parallel.

Multi-stage switches. Because simple switch fabric architectures are only scalable within limits, most larger switches contain *banyan* fabrics made up of stages of *switching elements*. We implemented multi-stage versions of the switch models by re-using the LP objects of the single-stage switches and only changing the LP interconnect structure. We chose an asynchronous self-routing, perfect shuffle *delta* interconnect structure for the multistage switches. Delta fabrics are banyan fabrics with the attractive property that the bits of the output port index (the *tag*) are used for internal routing [Patel 81]. In these switches, internal blocking and cell loss can be avoided or reduced by placing buffers in the switching elements, or by increasing the internal transmission rates relative to the external rates.

Fig. 4 shows a multi-stage switch with output buffered switching elements. The output buffer LPs of switching elements are identical to the output LPs at the switch output ports. A multi-stage shared buffer switch class and a multi-stage crossbar switch class were implemented similarly: the switching elements use the same shared buffer LPs and crossbar LPs as the respective single-stage switches.

4 SVC Signaling

4.1 Signaling Layers

Figure 5 shows the layered structure recommended by the ATM Forum [ATM Forum 93] for signaling. We term the layer associated with the processing of signaling

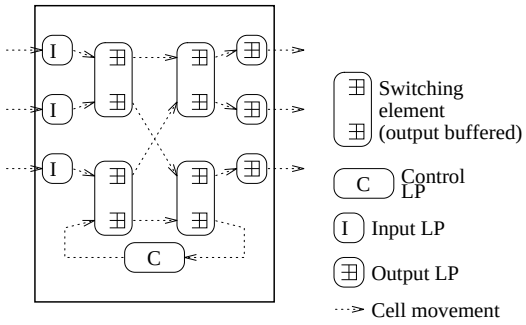

Figure 4: Multi-stage Switch

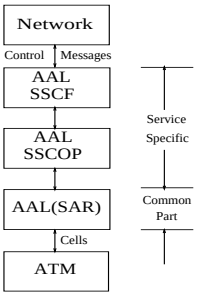

Figure 5: ATM Forum UNI Signaling Model

messages the Network layer because it performs many of the functions of Layer 3 of the OSI model. The Service-Specific Connection-Oriented Protocol (SSCOP) ensures reliable delivery of signaling messages to the signaling peer on the other end of the link. The Service-Specific Coordination Function (SSCF) maps the services provided by the SSCOP into services required by the Network Layer. The Segmentation and Reassembly (SAR) layer converts signaling messages into a cell stream and vice-versa. The ATM layer deals with switching and scheduling of cells.

In order to reduce the complexity due to additional layers, we combine the Network Layer, the SSCOP and SSCF. This means that each connection has the responsibility of ensuring that its signaling messages are reliably delivered (using timers and retransmissions). *Since the simulator does not model node or link failure, we only need to ensure reliability in the face of message loss due to buffer overflow.* However, the stream of cells to and from the ATM layer is unchanged as a result of this simplification. This simplification has the result of introducing signaling messages that are not specified in the ATM Forum signaling protocol. Also, the signaling protocol used at the User-Network Interface (UNI) is the same as that used at the Network-Network Interface (NNI). Figures 6, 7 and 8 explain the signaling protocol used at the the UNI and between ATM switches.

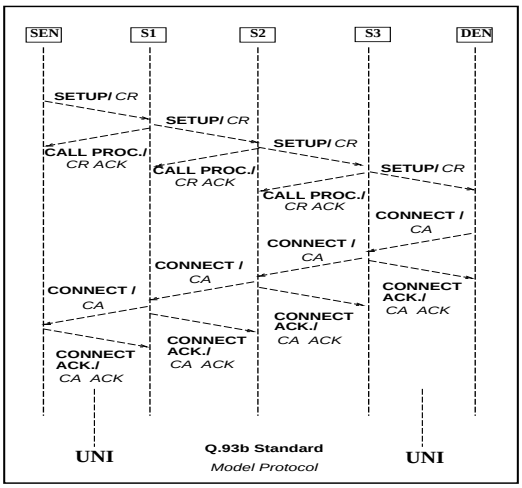

Figure 6: Connection Accepted

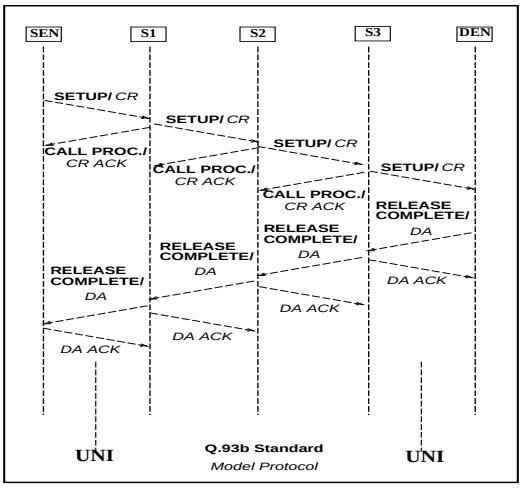

Figure 7: Connection Refused

4.2 Modeling a Layer

Each of the layers in the signaling model is implemented as a SimKit logical process (LP).

4.2.1 Modeling the Busy Period of an LP

In order to model the busy period of an LP when a message is received by it, a *busy* flag is associated with the LP. If the flag is set when a message is received, the message is placed in a queue.

When an LP begins processing a message, it sets the *busy* flag and sends an event to itself. This event will be delivered after a time corresponding to the duration

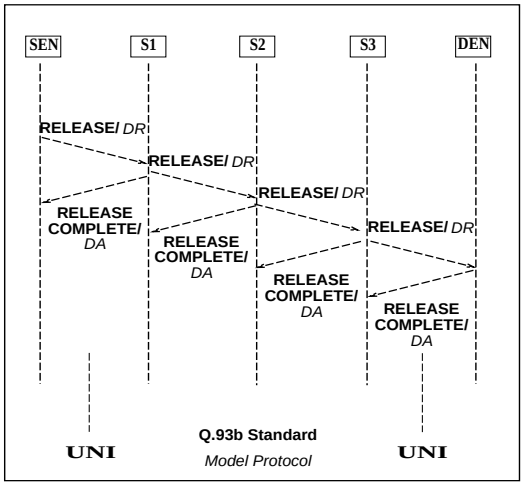

Figure 8: Disconnection Request

of the busy period. Until it is received, any arriving messages will find the LP busy and will be placed in a queue. At the end of the busy period, the *busy* flag is reset and the message at the head of the queue is processed as before.

4.3 Modeling the Network layer

The network layer maintains state per connection. Signaling messages and timer messages are the events that move a connection from one state to another. Thus, when an event is received, the connection to which it refers is found. Given the state of the connection and the event, a function is called to process the event. This makes it easy to for the user to create additional signaling messages with very little knowledge of the working of the model.

4.3.1 Connection Entries

The Network layer at every switch has a table of connection entries that store the information relating to each connection. Each connection has a connection entry with an index in the connection table and a connection id.

Once a connection is torn down, the free entry in the connection table may be given to another connection. However, pending messages for the connection just torn down may be delivered to the new connection. To avoid this, the messages between NL peers specify *connection identifiers* and not connection indexes. The NL at each switch generates its own stream of monotonically increasing connection IDs.

A mapping between connection id and connection index is maintained, so that once the connection id of a message at the local NL is known, the connection entry to which it should be delivered can be found. A mapping is also maintained between the connection id of the connection at the peer NL and the corresponding local connection id.

4.3.2 Reducing the Complexity of the Signaling Protocol

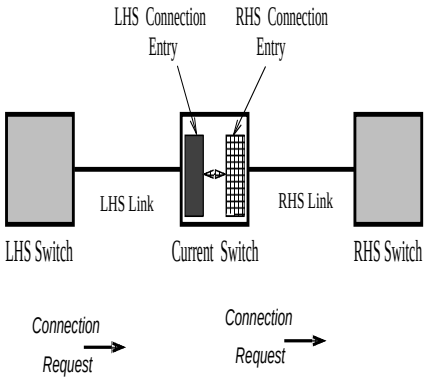

Figure 9: Sibling Connection Entries

With reference to figure 9, consider a switch that has just received a connection request (CR) from another switch (Left Hand Side or LHS Node) on a link or VP (LHS Link). A connection entry is created for this connection at the switch. Since it is not the destination end node, it looks at each of the outgoing links and VPs that leads to the destination end node and chooses one with sufficient bandwidth (Right Hand Side or RHS Link). The node on the other side of the RHS Link is the RHS Node.

The number of states for the connection entry would be large if the same connection entry had to manage packets arriving on both the RHS and LHS links. Therefore, the connection entry (LHS entry) created when the CR arrives on the LHS link makes a duplicate of itself (RHS entry). The RHS entry manages the RHS Link and has its own connection id and state information.

The connection id of one connection entry is known to the other. So they can communicate with each other. These connection entries are called siblings in distinction to *peers*. The latter are communicating connection entries at neighboring nodes.

This scheme has the effect of simplifying the signaling protocol. Moreover, as a result of the symmetry, the protocol used between the LHS node and the LHS entry is the same as that between the RHS entry and the RHS node, further simplifying the protocol.

4.3.3 Making the Network layer Reliable

The Network layer is made reliable by using timers and retransmitting signaling messages on expiration of the timer. However, the timer value must be chosen so that the generated cell stream is as close as possible to that when the SSCOP is present.

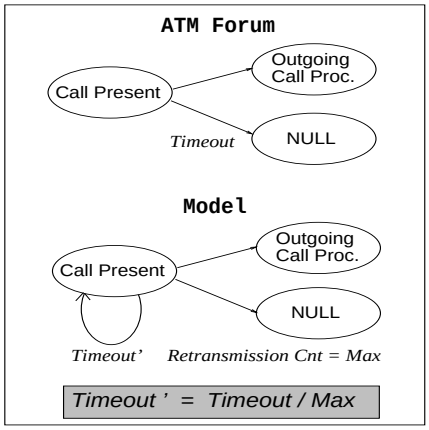

Figure 10: Making a Protocol Reliable

Figure 10 shows how the state diagram for an unreliable protocol can be modified to make it reliable. It also suggests a simple way of deriving the timer values in the modified protocol based on the timeout period in the original protocol and a retransmission count. The higher the retransmission count, the more reliable the protocol will be in the face of message loss. However, there is a lower bound on the setting of the timer; if it is less than twice the link delay, signaling messages will be retransmitted unnecessarily.

Whenever a connection request (CR), disconnection request (DR), connection accept (CA) or Disconnection accept (DA) packet is transmitted a timer is started. *A given connection can have only one pending timeout event.* Starting a timer consists of sending a timer event to itself (the Network LP) with the id of the connection which must receive the timeout event and placing an entry in the timer queue. This entry consists of a mapping between the connection id and the time that the timeout event should be received. The reason for the timer queue is to make it possible to kill the timer. It is not possible to delete a SimKit event. So when the timeout occurs, the timer queue is searched to find an entry with the same connection id as in the timeout event. If such an entry is found, the timeout message is processed. However, if the timer had been killed by deleting the entry in the timer queue, the timeout message will not be processed.

4.3.4 Connection Admission Control (CAC)

CAC determines whether a call can be admitted locally by the switch based on its current configuration of available resources.

As a first step, the bandwidth required by the connection is calculated using the traffic descriptor and required QoS (this may be different in the forward–source-to-destination–and reverse directions). The function that performs this calculation can be modified at will by the researcher. At present, this function exists in the form of a stub. Indeed, much ATM research is focussed on a way of satisfactorily deriving the bandwidth requirements of arbitrary traffic patterns on the basis of a supplied descriptor.

The next step is to check that bandwidth is available in the reverse direction on the link or VP that the connection arrived on. If it is not, the connection is rejected.

The switch then tries to find an outgoing link leading to the destination with sufficient forward bandwidth. The list of outgoing links for the destination switch is obtained using the local routing table. If the bandwidth is available, forward and reverse bandwidths are reserved. Otherwise, the connection is rejected.

References

[Ahmadi and Denzel 89] H. Ahmadi and W. E. Denzel. *A Survey of Modern High-Performance Switching Techniques.* IEEE JSAC, vol. 7 no. 7 (1989), pp. 1091-1103.

[Arlit et. al. 94] M. Arlitt, Y. Chen, R. Gurski, and C. Williamson. *ATM-TN Traffic Modelling.* TeleSim Project Internal Report, August 1994, 18 pages.

[Dobosiewicz and Gburzynski 93] W. Dobosiewicz and P. Gburzynski. *SMURPH: An Object Oriented Simulator for Communication Networks and Protocols.* MASCOTS'93, San Diego, CA (1993), pp. 351-352.

[ATM Forum 93] The ATM Forum *ATM User-network Interface Specification, Version 3.0.* Prentice Hall, New Jersey, September 1993.

[Jefferson 85] D. R. Jefferson. *Virtual Time.* ACM Transactions on Programming Languages and Systems, vol. 7 no. 3 (1985), pp. 404-425.

[Patel 81] J. H. Patel. *Performance of Processor-Memory Interconnections for Multiprocessors.* IEEE Transactions on Computers, vol. 30 no. 10 (1981), pp. 771-780.