Technical Report

Optimistic Parallel Simulation of TCP/IP Over ATM Networks

Ming H. Chong
Victor S. Frost

ITTC-FY2001-TR-18836-06

November 2000

Project Sponsor:
Sprint Corporation
Technical Planning & Integration

Abstract

DARPA's Next Generation Internet Implementation Plan calls for a capability of simulations of multiprotocol networks with millions of nodes. Traditional sequential simulators are no longer adequate to perform the task. Parallel simulators and new modeling frameworks are the better approach to simulate large-scale networks. This thesis demonstrates the feasibility of optimistic parallel simulation on TCP/IP over ATM networks, and compares the performance of a parallel simulator to *ProTEuS* (a distributed emulation system targeted for network simulation) in terms of speedup and scalability. The parallel simulator used in this study is Georgia Tech. Time Warp (GTW) which performs parallel discrete event simulation on shared-memory multiprocessor machine. Network models have been developed to construct large-scale TCP/IP over ATM scenarios to evaluate the performance of GTW. Results indicate that large-scale network simulation can greatly benefit from optimistic parallel simulator due to GTW's capability in exploiting parallelism. However, GTW lacks scalability, in terms of network size, when compared to *ProTEuS*.

Contents

1 Introduction **1**

1.1 Parallel Discrete Event Simulation . . . 2

1.1.1 Conservative approach . . . 3

1.1.2 Optimistic approach: Time Warp . . . 4

1.2 Motivation . . . 5

1.3 Thesis organization . . . 6

2 Related Work **7**

2.1 Proportional Time Emulation and Simulation (ProTEuS) . . . 7

3 Georgia Tech Time Warp (GTW) **9**

3.1 Overview . . . 9

3.2 GTW kernel . . . 10

3.2.1 Logical processes . . . 10

3.2.2 States and checkpointing . . . 11

3.2.3 The event queue data structure . . . 11

3.2.4 The main scheduler loop . . . 13

3.2.5 Computing GVT . . . 13

4 Implementation of simulation models with GTW **15**

4.1 Overview . . . 15

4.2 Application layer . . . 17

4.2.1 ABR source . . . 17

4.2.2 VBR source . . . 17

4.2.3 TCP source 18
4.3 TCP layer 18
4.4 ATM Segmentation and Reassembly (SAR) 19
4.5 ATM network layer 20
4.5.1 Overview of ABR traffic management 20
4.5.2 ABR end-host 23
4.5.2.1 ABR source behavior 23
4.5.2.2 ABR destination behavior 23
4.5.2.3 ABR end-host queueing discipline and traffic shaping 24
4.5.3 Switch Behavior 25
4.5.3.1 Switch routing and queueing discipline 25
4.5.3.2 Switch congestion control 27
4.5.3.3 EPRCA 28

5 Evaluation **30**
5.1 Overview 30
5.2 Validation of GTW models for ATM Simulation 31
5.2.1 Link Utilization 32
5.2.2 Mean Queueing Delay 32
5.2.3 ABR Queue Length 33
5.2.4 ABR Source Rate 33
5.2.5 Execution Time 34
5.3 GTW performance on ATM network simulation 35
5.3.1 Scenario A: 2 cores, 4 edges, 40 hosts 35
5.3.2 Scenario B: 4 cores, 12 edges, 120 hosts 40
5.4 Effect of network characteristics on GTW performance 44
5.4.1 Effect of feedback loop in network: Rollbacks degrade GTW performance 44
5.4.2 Effect of round trip time (RTT) 47
5.4.3 Effect of Network Size 51
5.5 Effect of simulation parameters on GTW performance 54

5.5.1 Event memory and state memory allocation 54

6 Conclusions and Future Work 56

List of Tables

4.1 List of parameters in ABR service 21
4.2 New ACR Upon arrival of BRM 22
4.3 Example routing table 26

5.1 Clipper system information 30
5.2 System parameters 31
5.3 Mean Normalized Link Utilization 32
5.4 Mean ABR Cell Queuing Delay 32
5.5 Execution Time 35
5.6 Scenario A: Source parameters 35
5.7 Scenario A: Execution Time 37
5.8 Scenario B: Source parameters 41
5.9 Scenario B: Execution Time 42
5.10 Source traffic policy 47
5.11 Execution time vs. Round Trip Time (RTT) 49
5.12 Execution Time vs network size 52

List of Figures

4.1 Implementation: Protocol layers 17

4.2 ABR Traffic Management 20

4.3 Structure of RM cell 22

4.4 End-host queueing discipline 24

4.5 Per-Class Queuing in the ATM switch 26

5.1 3-switch, 5-host topology 31

5.2 ABR Queue Length (zoom) 33

5.3 ABR Source Rate (zoom) 34

5.4 6 Switch 40 Host Topology 36

5.5 Scenario A: Execution time vs Number CPU 38

5.6 Scenario A: GTW Speedup (relative to 1 processor) 39

5.7 16 Switch 120 Host Topology 40

5.8 Scenario B: Execution time vs Number CPU 43

5.9 Scenario B: GTW speedup (relative to 1 processor) 43

5.10 Rollback percentage vs Event memory allocated 45

5.11 Number of aborted event vs Event memory allocated 46

5.12 Number of fossil collection vs Event memory allocated 46

5.13 Scenario A: Execution time vs. RTT (fixed load) 48

5.14 Scenario A: Execution time vs. RTT 50

5.15 Execution time vs. network size 53

5.16 Execution time vs. event memory allocated 54

List of Programs

4.1 Pseudo code for source reaction to backward RM cells 22
4.2 EPRCA upon reception of a backward RM cell 28

Chapter 1

Introduction

Computer simulation is the machine evaluation and observation of a system. It provides a practical methodology for analyzing system bahaviors that are either too complex for mathematical analysis, or too costly to study by prototyping. Simulation has become an essential phase of design processes in many areas including aeronautics, automobile, weather forecasting, electronics, and telecommunications to evaluate and improve systems before physical implementation.

The global communications architectures of the new millenium will emerge from internetworking of diversified wireline, wireless, and satellite networks interconnected at hundreds of millions of nodes. One major concern related to network engineering is the behavior of this very large hierarchical multihop network carrying millions of multiprotocol flows. With the rapid growth of global infrastructure, network configurations of very large size have to be simulated to study issues such as scalable routing. Also, to avoid misinterpreting transient bahaviors, models often have to be simulated for a very long timescales. DARPA's Next Generation Internet Implementation Plan call for a capability of simulations of 100,000 node, multiprotocol networks [10], and simulations of 10,000,000 node networks. Conventional sequential simulators such as BONeS [1], OPNET [2] lack the capabilities to model such large systems while keeping the simulation time to a tolerable level. New modeling frameworks and parallel simulator attempt to address such simulation requirements. The DaSSF [10] modeling framework developed at Dartmouth College has shown significant speedup in an attempt to simulate global internet with over 16,000 nodes and routers on a 14-processor Sun Enterprise server [10]. Parallel simulation reduces execution time by distributing the total workload onto multiple processing entities (i.e., processor or workstation).

However, parallel simulation has not not yet been widely used in network research and industrial practice due to the lack of a convenient modeling methodology suitable for parallel execution, and the absence of mature software environment. However, parallel simulations can be performed on commodity multiprocessor servers, and on a network of workstations, all of which are now commonly available in academic research labs and industry facilities.

Parallel discrete event simulation (PDES) techniques have been actively researched and improved over the past decade. Promising results of PDES's feasibility to ATM networks simulation have been demonstrated by Georgia Tech PADS group [4] and the Telesim project [31]. Experiments with large ATM network scenarios with 100 switches show that execution time speedups of 3-5 are possible on a 16-processor shared-memory SGI PowerChallenge [31]. However, there are significant technical challenges to overcome regarding event granularity, simulation partitioning, and load balancing. Other parallel simulators includes UCLA's *ParSec* [24] and Purdue's *ParaSol* [23]. Most parallel simulators run on shared-memory multiprocessor workstations where processors and memory are closely coupled via high-speed bus. The synchronization can be very fast and reliable. Nevertheless, the low cost and scalability of a PC and ethernet-based network of workstations (NOW) makes NOW another attractive platform for parallel discrete event simulation. Although such loosely-coupled systems require a more complex inter-processor communication mechanism to ensure correct and efficient message delivery, techniques have been proposed in Georgia Tech Time Warp (GTW) [6] and *Parasol* [26] to yield competing performance.

1.1 Parallel Discrete Event Simulation

Discrete event simulation is a method used to model real world systems by decomposing a system into a set of logical processes (LPs) autonomously progressing through time. All interprocessor and intraprocessor communications are handled by message passing (i.e., event scheduling). Each event is assigned a timestamp, and must take place on a destination LP. The result of this event can change the state of an LP, and/or generate new events to one or more other LPs.

A sequential discrete event simulation engine assigns all LPs to a single processor and always process events in strict logical time order. Since time order is strictly enforced and only one event can be processed at a time, interprocessor events can never arrive at LPs either out of sequence or after an event with a later scheduled timestamp has been processed.

Many simulations are sufficiently large that it is impractical to run them on a single processor within a realistic time-frame. The execution time of a simulation experiment can be reduced by distributing the experiment over multiple processing units. As a matter fact, using a multiprocessor machine is much more cost effective than building a single processor with equivalent computation power. Theoretically, large-scale simulation with huge number of LPs inherits more parallelism. Therefore the simulation should be performed in parallel fashion on multiprocessor computers.

Several practical approaches to parallel simulation includes replication, conservative technique, and optimistic technique [20, 9, 13, 29]. A simple way for distributing a simulation experiment over many processors is to run the same replication with different parameter values on separate processors. This replication technique is relatively efficient but it cannot extract all the parallelism available in a problem, especially when searching through a problem's parameter space.

In practice, several processors can be used in a coordinated fashion to execute the same simulation replication. A simulation model is partitioned into LPs. These LPs are distributed across multiple processors for simulation and they communicate only by message passing. The difficulty is to ensure that all messages and events in the PDES are processed in the same order and at the same simulation time as in a single processor simulation. Out of order processing results in causality errors. Two popular synchronization techniques to ensure accurate parallel discrete event simulation are the conservative and the optimistic approaches.

1.1.1 Conservative approach

Conservative approach represents an earliest work in PDES first desicribed by Chandy and Misra [9]. Simulation synchronized by conservative approach consists of a network of LPs communicating with timestamped messages. Each LP advances its local clock and processes incoming events only if it could determine that no causality errors would result from this event. Causality error is detected when an LP receives a message with a timestamp earlier than the

LP's local clock. Deadlock is possible, depending on the configuration of the LPs, when no LP can obtain the necessary information to advance its clock.

Early research has shown that the conservative approach cannot exploit the parallelism in many simulation models. The amount of parallelism is determined by how much the LP can lookahead based on local knowledge. And the amount of lookahead depends on the struture of a simulation model. The conservative approach was known to be less suitable for general purpose network simulation since the lookahead available is often too little to exploit significant parallelism.

1.1.2 Optimistic approach: Time Warp

In contrast to conservative approach, optimistic approach is able to exploit parallelism when causality errors are allowed but infrequent. It allows clock at each LP to be advanced without regard to the states of other LPs. In other words, each LP executes without waiting on other LPs for messages. However, a mechanism is required to detect and correct causality errors by restoring the simulation state from a previously saved state and resimulating from that point. The most common optimistic protocol is the Time Warp protocol [13] originally proposed by Jefferson [19].

The Time Warp protocol corrects causality errors by saving state for each LP periodically, then rolling the simulation back to the point when causality errors are detected. Variations of Time Warp have been reported [5, 31] to perform excellent on realistic simulation problems. In general, optimistic approach are more useful for general purpose applications compared to conservative approach because the previous requires less knowledge of a system.

A few improvements can be done in order to improve Time Warp's performance. An inherent problem with optimistic simulation is the overhead of state-saving to permit rollback. Fujimoto [13] reported that the performance of an optimistic simulation was cut in half when LP state size increased from 100 bytes to 2000 bytes. Incremental state-saving[27] is one method to reduce state-saving overhead where only changes to the state are saved.

Rollback overhead is another major factor that affects performance. One way to reduce the cost associated with rollback is to use lazy cancellation. Time Warp using lazy cancellation do not immediately cancel the messages previously sent when a causality error is detected. Indeed, the messages are first compared to those that would be generated by the corrected computation [13]. Only those messages that were not generated by the corrected computation are canceled.

Achieving load balancing may be another improvement. For parallel simulation to work effeciently, the LPs must be distributed properly over the available physical processors. The optimal distribution cannot be known without performance profiling of the simulation. Dynamic partitioning can be a good solution though it is very complicated to implement in the simulator. During an optimistic simulation, processors that do not have enough LPs assigned will be frequently rolled back by old messages sent from heavily loaded processors and thus produce little useful work. Dynamic partitioning attempts to assign more LPs to those processors which are not doing much useful work. This technique has been shown to decrease local time gap among LPs, and reduce rollback frequency.

1.2 Motivation

Most parallel simulation works done on large-scale ATM network have been focusing their effort on two distinct areas. First, they exploit inherent parallelism of ATM network scenario to achieve maximum possible speedup. The second task is to study ATM network performance associated with various queuing disciplines and different traffic flows generated from video, audio, and data sources. However, the issue of how parallel simulator reacts to certain network, particularly the TCP over ATM networks with multiple control loops, have not been addressed.

Little work has been done to study how the performance of parallel simulators are sensitive to certain properties of the network being simulated such as the network delay (i.e. round trip time of traffic flow), how does a network protocol with multiple control loops (e.g. TCP/IP over ATM, ABR's EPRCA) affect the rollback frequency, and how well parallel simulators scale with growing network size on certain multiprocessor machines. Furthermore, the performance of parallel simulators varies on different hardware platforms. GTW has been shown to perform poorly on the popular SGI Origin2000 due to the newly adopted Cache-Coherent Non-Uniform Memory Architecture (CC-NUMA) in [8]. The proposed solution is discussed in [7].

This thesis compares the performance of Georgia Tech Time Warp (GTW) (i.e., an optimistic parallel discrete event simulator) to *ProTEuS* (i.e., a distributed proportional time network emulator), focusing on the issues mentioned above. GTW and ProTEuS will be discussed in the chapters 2 and 3.

1.3 Thesis organization

Chapter 2 provides some of the related works regarding ProTEuS. Chapter 3 illustrates the architecture of GTW, and the modeling interface. Chapter 4 explains the design and implementation of ATM network models used in simulation. Chapter 5 verified the simulation models and presents an evaluation of GTW's performance. Finally, chapter 6 concludes my thesis.

Chapter 2

Related Work

Alternative approaches other than parallel discrete event simulation (PDES) have been proposed to simulate ATM networks. One such hardware approach is the FPGA-based ATM Simulation Testbed (FAST) developed in University of California, Santa Cruz [30].The testbed uses field-programmable gate arrays (FPGAs) to allow implementation of important algorithms in an ATM switch in hardware. Each FPGA board enabled with I/O devices can be used to simulate an ATM switch, and multiple boards may be interconnected to simulate network of ATM switches.

ProTEuS [17] is another distributed synchronization technique used to perform cell-level ATM network simulation. The detail of ProTEuS will be discussed in the following section since the performance studies done in this thesis are compared to ProTEuS. It is used as a reference platform to show the pros and cons of GTW.

2.1 Proportional Time Emulation and Simulation (ProTEuS)

Although PDES has successfully demonstrated its feasibility as well as scalability in simulating large-scale network, it is often not known whether any network specific property will affect its performance. For instance, network protocol with feedback loop such as the TCP and ABR's EPRCA has the potential to cause excessive rollbacks that will degrade Time Warp's performance. In terms of cost, a rack of personal computer required to perform distributed simulation costs less than a multiprocessor server such as the SGI Origin2000.

ProTEuS is a general network simulation solution developed to combat the shortcomings of PDES. The focus of ProTEuS is a NOW architecture with commodity PCs where the system uses proportional time distributed system technique, and embedded system technique to synchronize distributed simulations as described in [17]. The system represents an innovative form of parallel simulation intended to speedup simulation by distributing the work across any number of physical machines. Initial experiments shown that ProTEuS outperformed GTW in terms of scalability. Growing the size of a network in a simulation is often limited by the available physical memory in a Time Warp simulator while this problem can easily be solved in ProTEuS by including more distributed hosts. Another merit of ProTEuS is that it does not incur state-saving and rollback synchronization overheads as required in Time Warp simulator.

ProTEuS uses commercial off-the-shelf PCs to configure a NOW simulation platform. The major interest of ProTEuS is to simulate ATM network using real system code such as the real operating system ATM protocol stack (Linux with ATM support) and the ATM signaling support which is the same as an off-board signaling architecture (Q.Port). This feature creates a faithful simulation and avoids the implementation of system code abstractions into a software simulator as is required in GTW. Also, an additional layer of virtual network device is used to support arbitrary mapping of simulated network entities onto a group of physical machines. The additional abstraction enhances the scalability of ProTEuS.

Chapter 3

Georgia Tech Time Warp (GTW)

3.1 Overview

The Georgia Tech Time Warp (GTW) is an optimistic parallel discrete event simulator developed by the PADS group of Georgia Institute of Technology led by Fujimoto [11]. The system is built on Jefferson's Time Warp mechanism [19]. The primary goal in designing GTW is to support efficient execution of small granularity discrete event simulation applications. The time required to process an event is very little in small granularity applications. The dominating performance factors are the send/receive event overhead (i.e., enqueue/dequeue) of network packets, and the synchronization overheads including state-saving and rollback process. Cell-level simulation of ATM network is an example of small granularity application. It is the reason why GTW is chosen among other Time Warp-based simulators in this work.

GTW runs on cache-coherant, shared-memory multiprocessor servers including the Kendall Square Research KSR-1 and KSR-2, Sequent Symmetry, Sun UltraSPARC workstation, Sun Enterprise server, and SGI Challenge/Origin. The latter three are the major platforms used in this work.

3.2 GTW kernel

3.2.1 Logical processes

A GTW program consists of the definition of a collection of logical processes (LPs) that communicate by exchanging timestamped messages/events. Each LP is identified by a unique integer ID, and the mapping of LPs to physical processors is static. The dynamic partitioning technique mentioned in previous chapter is not enabled in GTW. The execution of each LP is entirely message driven. LP cannot ”spontaneously” begin new computation. Any computation is a direct result of receiving a message. Hence, the behavior of each LP can be defined by three procedures.

1. The *initialize()* procedure: Initialize the LP at the beginning of the simulation. This procedure allocates all memory required by the LP and indicates to the GTW kernel what portion of the memory should be state-saved. Also, state of the LP is usually initialized here and initial messages are sent to get the simulation started. The procedure executes at simulated time zero.

2. The *process-event()* procedure: This is the procedure that will be called whenever a message is received. It forwards events received to the appropriate user-defined event handlers for processing.

3. The *wrapup()* procedure: It is called at the end of the simulation often to output statistics.

The execution of GTW program moves through three phases: initialization, event-driven execution, wrap-up.

The Initialization phase: A global initialization and a local initialization of each LP will be performed at this phase. The global initialization must specify all LPs involved in the simulation, and the static mapping of LPs to processors. Local initialization calls the *initialize()* procedure of each LP. This is done in parallel where each processor calls *initialize()* of the LPs that are mapped to it.

The Simulation phase: The simulation phase is driven by calls to event handlers, one for each LP receiving a new message. Events scheduled for LPs on a processor are processed one at a time in a time-ordered fashion and may modify state variables as well as schedule new events to other LPs including itself in the simulated future.

The Wrap-up phase: The wrapup phase consists of calls to *wrapup()* procedures of all LPs in the simulation. Simulation terminates when all *wrapup()* procedures are finished.

3.2.2 States and checkpointing

Each LP must define a state vector which contains all of the variables used by the LP. These variables should never be accessed by other LPs. A state vector may include three separate types of state variables that are distinguished by the type of checkpointing scheme that is performed on them.

1. **Read-only:** Read-only state variables that will never be modified after the initialization phase do not need to be checkpointed.

2. **Full-copy:** State variables that are automatically checkpointed. The GTW kernel will automatically make a copy of all of these variables (full-copy checkpointing) prior to processing each event for the LP. This is done transparent to the user application.

3. **Incremental:** State variables that are checkpointed when the application explicitly requests a variable be checkpointed. The GTW kernel uses incremental checkpointing where variables are state-saved before modifications are made on them.

An LP may choose to have some variables automatically checkpointed, and others incrementally checkpointed. These designations are static and cannot be changed during simulation.

The overhead of checkpointing can be large for some applications, and can easily cripple GTW's performance. The various state variable types are designed to reduce checkpointing overhead through careful selection of an appropriate type for each state variable. If the state vector is small, the state-saving overhead is small relative to the amount of computation in an event, then automatic checkpointing is recommended for all variables in state vector. Yet if state vector is large, and only a few variables are modified by each event, then incremental checkpointing should be used.

3.2.3 The event queue data structure

The original Time Warp proposed by Jefferson uses three distinct data structures: 1.) Input queue that holds processed and unprocessed events; 2.) Output queue that holds anti-messages; and 3.) State queue that holds history of LP state. However, GTW uses a single event queue data structure that combines the functions of these three queues. Direct cancellation is used.

Whenever an event computation schedules a new event, a pointer to the new event is kept in the sending event's data structure This eliminates the need for explicit anti-messages and the output queue.

The event queue data structure **(EvQ)** actually consists of two queues.

The processed event queue: Each individual LP on a processor maintains a processed event queue sorted by receive timestamp. Each processed event contains pointers to state vector history to allow rollback, and pointers to messages scheduled by the computation associated with this event. Note that all events and saved-state that are older than global virtual time (GVT) can be discarded to reclaim memory storage, as it is impossible to roll back to a virtual time before GVT. This process of destroying information older than GVT is referred as *fossil collection.* During *fossil collection*, the portion of this queue that is older than GVT is located by scanning the queue from high to low timestamp. Then the events to be collected are moved to the processor's free list. In this case, the *fossil collection* procedure need not scan through the list of events that are reclaimed.

The unprocessed event queue: Unlike the processed event queue, unprocessed events for all LPs mapped to a processor are stored in a single priority queue. Using a single priority queue for all LPs eliminates the need for a scheduling algorithm to enumerate the next executable LPs. It allows the selection of the next LP to execute, and location of the smallest timestamped unprocessed event in that LP to be implemented with a single dequeue operation. This reduces the event processing overhead and simplifies the GVT computation. However, if dynamic load management mechanism is enabled in the future, migration of an LP to another processor becomes a difficulty.

The event queue may only be exclusively accessed by the processor maintaining the queue. No locks are required to access it. In addition to an event queue, each processor also maintains two additional queues to hold incoming messages from other processors.

Message queue (MsgQ): Hold incoming positive messages that are sent to an LP residing on this processor. Events are placed into this queue by placing ”send” commands in user applications. Since access to this queue is shared among other processors, locks are used for synchronization.

The message cancellation queue (CanQ): Hold messages that have been cancelled. When a processor wishes to cancel a message upon detection of causality errors, it enqueues the messages being cancelled into the CanQ of the processor to which the messages was sent. Messages enqueued in CanQ can be viewed as anti-messages, however, it is the message itself rather than an explicit anti-message that is enqueued. Access to this queue is also synchronized with locks.

3.2.4 The main scheduler loop

After the simulation is initialized, each processor enters a loop that repeatedly performs the following steps:

1.) All incoming messages are removed from the MsgQ and filed into EvQ, one at a time. If a filed message contains timestamp smaller than the last event processed by the LP, causality error is detected and the LP is rolled back. Messages sent by this rolled back event are enqueued into the CanQ of the processor holding the messages.

2.) All incoming cancelled messages are removed from CanQ and processed one at a time. Anti-messages annihilate their complementary positive messages in the unprocessed event queue. However, if the positive messages have been processed, secondary rollbacks may happen and are handled in the same manner as rollbacks caused by straggler positive messages. Memory storage taken by cancelled messages is returned to processor's free memory pool.

3.) Dequeue an unprocessed event with the smallest timestamp from the EvQ, then process it by calling the LP's event handler. A smallest timestamp first scheduling algorithm is used where event with smallest timestamp is always selected as the next one to be processed.

3.2.5 Computing GVT

Global Virtual Time (GVT) is defined as the lower bound on the timestamp of all unprocessed or partially processed messages and anti-messages in the system. GVT computation ensures the simulation progresses and reclaims memory storage. In a shared-memory environment, GVT computation can be greatly simplified. An asynchronous algorithm is used in GTW to compute GVT. The algorithm runs in between normal event processing. It requires no special ”GVT messages”. All interprocessor communication needed for GVT computation is realized using three global variables: 1.) a global flag variable called *GVTFlag*; 2.) a global array to hold each processor's local minimum; and 3.) a variable to hold the new GVT value.

Any processor can initiate a GVT computation by setting *GVTFlag* to the number of processors in the system. *GVTFlag* holds a value of zero before GVT computation. A lock on this variable ensures that at most one processor initiates a GVT computation. Each processor maintains a local variable called *SendMin* which contains the minimum timestamp of any message sent after *GVTFlag* is set. Upon each message or anti-message sent, *GVTFlag* is checked , and *SendMin* is updated if the flag is set. At the beginning of main event processing loop, each processor checks if *GVTFlag* was set. If *GVTFlag* was set, the processor computes the minimum of the *SendMin* and the smallest timestamp on the unprocessed event queue. Then, it writes this minimum value into its entry of the global array, decrements *GVTFlag* to indicate that it has reported its local minimum. The normal event processing resumes and the *GVTFlag* is now ignored until new GVT value is received.

The last processor to compute its local minimum, which is the processor that decrements *GVTFlag* to zero, computes GVT from local minimum array and writes the new value into GVT variable. Each processor realizes the new GVT by observing that the value of GVT global variable has changed. A *fossil collection* is performed after new GVT is received.

The overhead associated with this algorithm is minimal. No synchronization is required. The major overhead in GVT computation is updating *GVTFlag* and *SendMin.* And the GVT calculation is performed by one processor.

Chapter 4

Implementation of simulation models with GTW

4.1 Overview

The implementation of ATM network and TCP time warp simulation models are modularized based on protocol layers. Protocol layers implemented integrate to form network components such as ATM end-host and switch. Three sources: ABR source, VBR source, and TCP source with different traffic characteristics simulate the application layer. A TCP module simulates the transport layer. A segmentation and reassembly (SAR) module simulates ATM AAL5 layer. An ATM network module, which can be viewed as an ATM NIC device, simulates ATM network services (i.e., ABR, VBR) and multiplexing. Finally, there is a link module that functions as the physical link layer.

This approach provides an easy way in constructing network scenario especially for large networks. Network scenario construction can adopt a bottom-up approach where network components can be interconnected to form larger network. Also, network scenarios can be modified, extended, and maintained at ease. However, the performance of this approach on GTW is not known. Each module represents an LP in GTW. The complexity (e.g., state size, amount of computation per event) of the LPs and the pattern of communication among LPs are factors determining GTW's performance. This complex task has been a research issue in the parallel simulation community and it would be left as future work.

The focal points of this implementation are model correctness, compatibility to ProTEuS, and optimum execution on GTW. The development of each module is based on the publicly available *NIST ATM simulator* [16] since the simulator performs cell-level ATM network simulation and is also event-driven. The queuing structure, ATM services, and event scheduling scheme from the NIST simulator are used as a reference in this implementation to ensure correctness of ATM simulations done in this work. Standards such as the ATM forum's traffic management specification are an important source. The modules developed are also tailored to match the design of ProTEuS in terms of queueing discipline, ABR cell rate control mechanism, and other ATM functions. This ensures the performance comparison between ProTEuS and GTW is fair because the complexities of network models in both platforms are matching.

ATM technology is calibrated to support a broad spectrum of applications including loss-sensitive data transmission and the delay-sensitive real time audio and video transmission. Hence, ATM layer provides four service categories: Constant Bit Rate (CBR), Variable Bit Rate (VBR), Available Bit Rate (ABR), and Unspecified Bit Rate (UBR). These service categories relate traffic characteristics and Quality of Service (QoS) requirements to network behavior. Each service category is associated with different functions for routing, Call Admission Control (CAC), and resource allocation.

CBR applications are guaranteed a fixed amount of bandwidth throughout the course of the connection. This service is typically used for real-time applications with strict delay requirements. VBR service is intended for applications with bandwidth requirements that vary with time such as interactive multimedia applications and compressed voice/video transmissions. VBR applications are guaranteed a sustained cell rate in compliance with a maximum burst size. ABR service is aimed for applications whose bandwidth requirements cannot be known precisely, but are known to be within an upper and lower bound. Example ABR applications are FTP sessions, or Internet traffic. These applications uses the instantaneous available bandwidth remaining after all CBR and VBR traffic requirements have been satisfied. The network provides application with information about the amount of available bandwidth through a feedback system. EPRCA is one such feedback system that has been implemented in this work and will be discussed in detail later. Finally, UBR receives the remaining bandwith after the previous traffic classes. UBR is used by best-effort services, such as e-mail, which are insensitive to delay and loss.

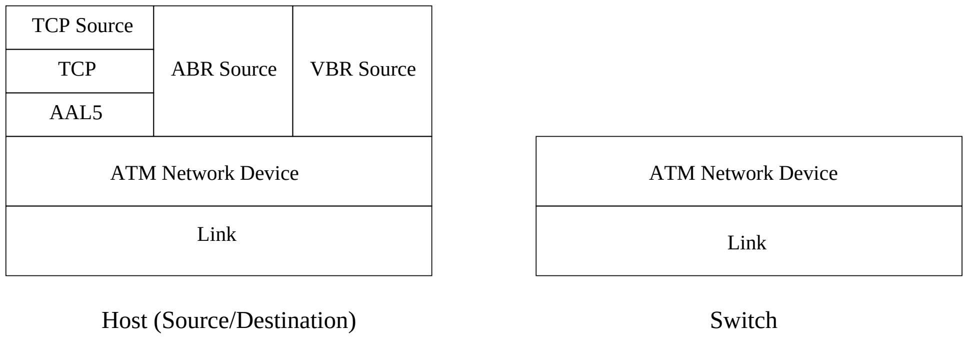

Figure 4.1: Implementation: Protocol layers

Figure 4.1 shows the protocol layers implemented. They will be discussed in detail in the following sections.

4.2 Application layer

4.2.1 ABR source

The ABR source module creates greedy ABR traffic that will always send data cells to the network device layer so that the ABR connection will never become idle. It keeps the ABR queue in the network device non-empty at all time. The source creates ABR data cells, attaches ATM header with necessary information (e.g., VPI, PTI, protocol ID, etc.) to the cells, and then sends the cells to the network layer to be transmitted at allowed cell rate (ACR). The ABR source is simply a cell generator.

4.2.2 VBR source

The VBR source module creates background VBR traffic. The module is basically derived from ABR source. The difference is that VBR source generates VBR data cells at a controllable rate, then transmits the cells to the network device layer. The cell rate can be controlled by explicit timeline schedule generated by capturing the cell traces from the movie *Star War* MPEG clip to simulate transmission of compressed video stream over ATM network. Another simple VBR traffic generator we developed has a square wave transmission pattern with maximum cell rate, minimum cell rate, and a user-defined duty cycle.

4.2.3 TCP source

A TCP source is also a greedy source. Note that sources are designed to be greedy so that the source will always fill up the bandwidth available to the connection. The aggregate traffic from all sources can be predicted and hence it is easier to control the bandwidth utilization of a link in a network.

The TCP source sends buffers to the TCP layer and it will keep the input buffer of the TCP layer busy so that there are always packets waiting in the output buffer of TCP layer to be propagated to the lower protocol. A TCP connection will never become idle with this greedy TCP source.

4.3 TCP layer

Transmission Control Protocol (TCP) offers a connection-oriented, reliable byte-stream service. It is a full-duplex protocol that guarantees in-order delivery of a stream of bytes. Each TCP connection supports a pair of byte streams, one flowing in each direction. It also includes a window-based flow-control mechanism for a connection that allows the receiver to limit how much data the sender can transmit.

In addition to the above features, TCP also implements a highly tuned congestion control mechanism. The idea of this mechanism is to keep the sender from overloading the network. The strategy of congestion control is for each source to determine how much capacity is available in the network, so that it knows how many packets it can safely have in transit in the network. Once a given source has this many packets in transit, it uses the arrival of an *acknowledgement* (ACK) as a signal that one of its packets has reached destination, and that it is therefore safe to insert a new packet into the network without raising the level of congestion.

To determine the available capacity in the network, TCP uses an aggresive approach called *slow start* to increase the *congestion window* rapidly from a cold start. *Congestion window* limits how many packets can be in transit in the network. The higher the *congestion window*, the higher the throughput. *Slow start* increases *congestion window* exponentially until there is a loss, at which time a timeout causes multiplicative decrease. In other words, *slow start* repeatedly increases the load it imposes on the network in an effort to find the point at which congestion occurs, and then it backs off from this point.

The original implementation of TCP timeouts led to long periods of time during which the connection went idle while waiting for a timer to expire. A new mechanism called *fast retransmit* was added to TCP. The mechanism triggers the retransmission of a dropped packet sooner than the regular timeout mechanism. When the *fast retransmit* mechanism signals congestion, rather than drop the *congestion window* all the way back to 1 packet and run *slow start*, it is possible to use the ACKs that are still in the pipe to clock the sending of packets. This mechanism called *fast recovery* removes the *slow start* phase that happens between when *fast retransmit* detects a lost packet and additive increase begins. *Slow start* is only used at the beginning of a connection and whenever a regular long timeout occurs. At all other times, the *congestion window* is following a pure additive increase/multiplicative decrease pattern.

The implemented TCP layer is a derivation of the BSD 4.3 TCP (Reno version) which will behave as described above. The TCP layer also supports *delayed ACKs* - acknowledging every other segment rather than every segment - which is optional.

4.4 ATM Segmentation and Reassembly (SAR)

ATM cell is fixed at a size of 53 bytes. Packets handed down from high-level protocol are often larger than 48 bytes, and thus, will not fit in the payload of an ATM cell. The solution to this problem is to fragment the high-level message into low-level packets at the source, transmit the individual packets over the network, and then reassemble the fragments back together at the destination. The technique is called segmentation and reassembly (SAR).

The ATM Adaptation Layer (AAL) represents a SAR layer. AAL sits between ATM layer and the variable-length packet protocols such as IP. Since ATM is designed to support a number of services, different services would have different AAL needs. Hence, four adaptation layers were originally defined. AAL1/2 were designed to support guaranteed bit rate services like voice, while AAL3/4 were intended to provide support for packet data transfer over ATM. AAL5 emerges as a more efficient layer over AAL3/4 and is the preferred AAL in the IETF for transmitting IP datagrams over ATM. AAL5 is implemented in this SAR module since we are assuming internet traffic on the TCP connections.

4.5 ATM network layer

4.5.1 Overview of ABR traffic management

ABR requires constant interaction between the network and the source end-host. The network consists of switches that use their current load information to compute the allowable cell rates for the sources. The source end-hosts then limit their data transmission to rates allowed by the network. These rates are sent to the sources as feedback via *resource management (RM)* cells. RM cells are generated by the sources and travel across network to the destination end-host. The destinations return the RM cells to the sources.The ABR traffic management scheme is illustrated in Figure 4.2. The ABR traffic management is rate-based since the sources send data at a specified ”rate” which is different from other packet network such as TCP where the control loop is window-based. The scheme is a closed-loop because there is a continuous feedback between the network and end-hosts.

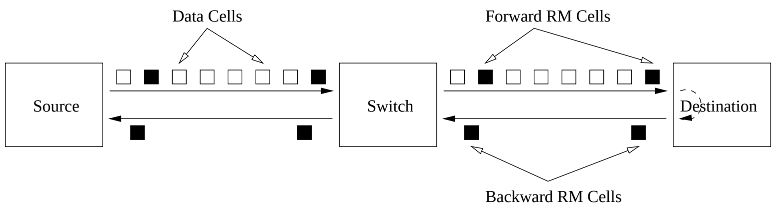

Figure 4.2: ABR Traffic Management

During connection setup, ABR applications negotiate several parameters with the network. Two important parameters are the peak cell rate (PCR), and the minimum cell rate (MCR). PCR is the maximum rate at which the source will be allowed to transmit on this virtual circuit (VC). MCR is the guaranteed minimum rate a network has to reserve for the VC. During data transmission phase, the rate at which a source is allowed to send at any particular instant is called the allowed cell rate (ACR). The ACR changes between PCR (i.e., upper bound) and MCR (i.e., lower bound). It is set to initial cell rate (ICR) at the beginning of the connection or after a long idle period. The list of parameters used in the ABR service is presented in Table 4.1.

Label	Meaning	Default Value
PCR	Peak Cell Rate	-
MCR	Minimum Cell Rate	0
ACR	Allowed Cell Rate	-
ICR	Initial Cell Rate	PCR
NRM	Number of cells between FRM cells	32
MRM	Minimum number of cells between forward RM cells	2
TRM	Upper bound on Inter-FRM Time	100ms
RIF	Rate Increase Factor	$\frac{1}{16}$
RDF	Rate Decrease Factor	$\frac{1}{16}$
ADTF	ACR Decrease Time Factor	500ms
TBE	Transient Buffer Exposure	16777215
CRM	Missing RM-cell Count	$\lceil \frac{TBE}{NRM} \rceil$
CDF	Cutoff Decrease Factor	$\frac{1}{16}$

Table 4.1: List of parameters in ABR service

The structure of the RM cells is shown in Figure 4.3. Every RM cell has the regular five bytes ATM header. The protocol id field is set to 1 indicating ABR connection. The payload type indicator (PTI) field is set to 110 in binary to distinguish RM cells from data cells. The direction (DIR) bit is used to distinguish forward RM (FRM) cell and backward RM (BRM) cell. Congestion indication (CI) and no increase (NI) bits are used in explicit rate switches to indicate congestion in the network. The current cell rate (CCR) field is used by the source to indicate to the network its current rate. The explicit rate (ER) field gives feedback from the network to the sources. It advertises the maximum rate allowed to the source. If either CI or NI is set, the source will adjust its ACR according to Table 4.2. Note that when there are multiple switches along the path, the lowest ER given by the most congested switch is the one that reaches the source in BRM.

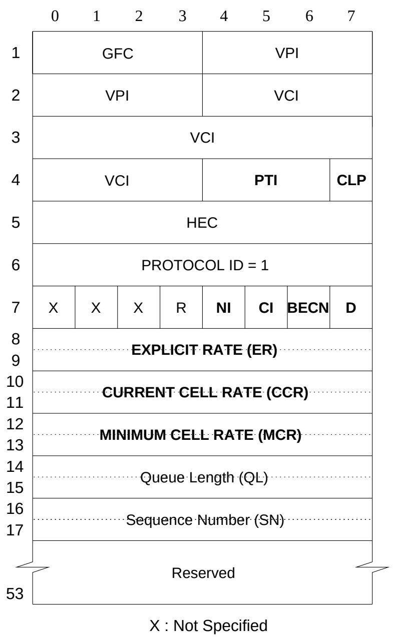

Figure 4.3: Structure of RM cell

CI	NI	New ACR
0	0	MIN(ER, ACR $+$ PCR $\times$ RIF, PCR)
0	1	MIN(ER, ACR)
1	X	MAX(MCR, ACR $-$ ACR $\times$ RDF)

Table 4.2: New ACR Upon arrival of BRM

Program 4.1 Pseudo code for source reaction to backward RM cells

```
if (CI = 1)
    ACR = ACR - ACR x RDF;

else if (CI = 0) and (NI = 0)
    ACR = ACR + ACR x RIF;
    ACR = MIN (ACR, PCR);
endif
if (ACR > ER)
    ACR = ER;

ACR = MAX (ACR, MCR);
```

4.5.2 ABR end-host

4.5.2.1 ABR source behavior

1. The transmission rate of sources should equal to or below ACR. Also, $MCR <= ACR <= PCR$
2. At the beginning of a connection, sources send at ICR. The first cell is always a FRM cell.
3. The sources are required to send an FRM after every 31 data cells. However, if the source rate is low, the time between RM cells would be large and network feedback will be delayed. The problem can be overcome by imposing a source to send a FRM cell if more than 100 ms (TRM) has elapsed since the last FRM. This causes another problem for low rate sources. In some case, at every transmission opportunity the source may find that it has exceeded 100 ms and needs to send an FRM cell. No data cells can be transmitted in this case. Hence, an additional condition was added that there must be at least two other cells (MRM) between FRM cells.
4. If no FRM cells have been sent after a period of 500 ms (ADTF), then the ACR should be reset to the minimum of the ACR and ICR and an FRM cell should be sent.
5. If at least CRM number of FRM cells have been sent since the last backward RM cell was received, then ACR shall be reduced by at least ACR X CDF, unless that reduction would result in a rate below MCR, in which case ACR shall be set to MCR.
6. When a BRM is received, ABR sources should reset the ACR based on Congestion Indication (CI), No Increase (NI), and Explicit Rate (ER) information from the BRM cell. New ACR is computed as shown in Table 4.2.

4.5.2.2 ABR destination behavior

The duty of destinations is to turn around FRM cells they received. This involves changing the direction bit in the FRM cells so that they become BRM cells, then the destinations send them back out on the VC on which they were received. The following items illustrate destination behavior.

1. When a data cell is received , its EFCI bit is stored as the EFCI state of the connection.
2. Upon receiving FRM cell, the destination should turn around the cell to return to the source. The direction bit will be changed from ”forward” to ”backward”. Also, set the CI bit if the EFCI was set in the last data cell received.
3. ABR destinations may set CI and/or NI bits, as well as modifying the ER in the RM cell if it is experiencing internal congestion.
4. ABR destinations may generate a BRM cell without first having received a FRM cell in order to increase the responsiveness of the source. The Backward Explicit Congestion Notification (BECN) and CI bit in the BRM cell should be set.

4.5.2.3 ABR end-host queueing discipline and traffic shaping

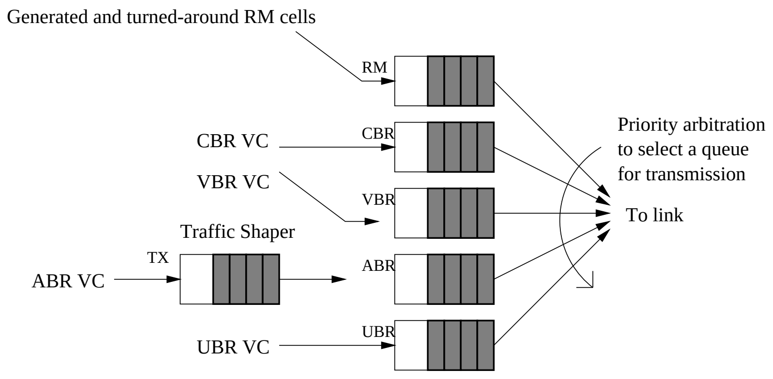
Generated and turned-around RM cells
RM
CBR VC
CBR
VBR VC
VBR
Traffic Shaper
TX
ABR VC
ABR
UBR VC
UBR
Priority arbitration to select a queue for transmission
To link

Figure 4.4: End-host queueing discipline

When sources send data on a virtual circuit, packets from application propagate to the SAR layer where they are chopped into 48 byte payloads. Five bytes of ATM header are added to each payload and a stream of 53 byte ATM cells propagate to the lower protocol layer.

Figure 4.4 shows how ATM cells are queued at the ATM network device. Cells entering ATM network layer are queued into a transmit queue on a per-VC basis where they wait to be enabled by the traffic shaper. Cell-level traffic shaping is performed on every ABR VC in this implementation. The purpose of traffic shaping is to decide whether or not a VC should be allowed to send a cell based on its current cell rate (CCR). The algorithm of traffic shaping is trivial. The shaper spaces cell transmission on the VC as evenly as possible using CCR. Each ATM network device has five queues as shown in Figure 4.4. When ABR cells are enabled by the shaper, they are transferred from their VC queue to the ABR queue. VBR traffics are not shaped and will proceed directly to VBR queue. CBR and UBR queues are unused in this work. At each transmission opportunity, the network device is allowed to send one cell if they are any in the device queues. The scheme used to choose a cell is a strict priority which favors traffic classes in the order: RM, CBR, VBR, ABR, and UBR. Hence, if an RM cell waiting, it will be sent with highest priority, regardless of the occupancy of other queues. The selected cell is then sent to the physical link.

4.5.3 Switch Behavior

4.5.3.1 Switch routing and queueing discipline

The task of an ATM switch is to switch cells from an incoming (Port, VC) onto an outgoing (Port,VC) based on the contents of a routing table. Note that the current implementation does not support virtual path (VPI) switching capabilities. Table 4.3 shows a sample routing table with three connections. Each entry represents routing information for a particular connection. For instance, cells with VCI=1 coming from port number 0 are routed to port number 6 with same VCI number.

The switch employs per-class queueing discipline. In per-class queueing, each port has a FIFO queue for each class of traffic; CBR, VBR, ABR, and UBR. In addition, there is a separate RM queue to segregate RM cells from data cells. As shown in Figure 4.5, incoming cells are routed to the output port of their connection where they are queued on the particular queue for their traffic class.

In		Out	
Port	VCI	Port	VCI
0	1	6	1
0	2	7	2
1	3	6	3

Table 4.3: Example routing table

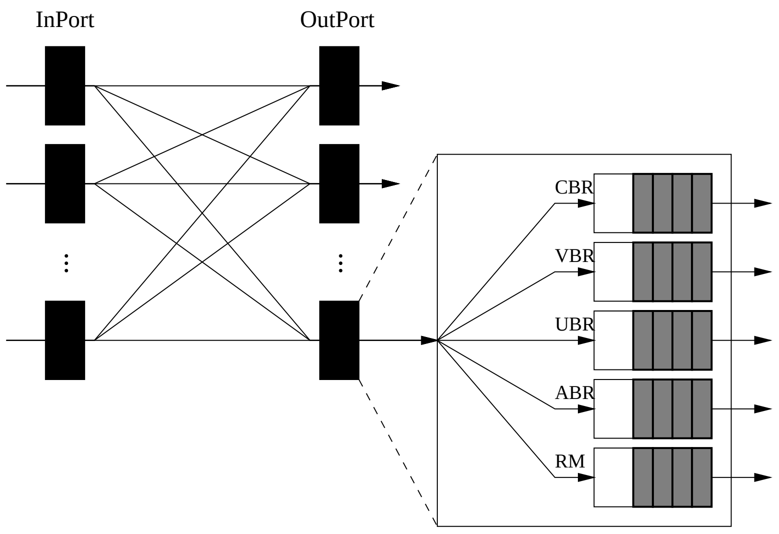

Figure 4.5: Per-Class Queuing in the ATM switch

During each slot time, one cell from each port is transmitted to next destination. A weighted-round robin scheduling mechanism has been implemented to service cells between classes on each port. Since the focus of this work has been ABR service and background VBR traffic, the scheduling support is only available for ABR and VBR. The VBR and ABR queues are serviced with a VBR:ABR weight ratio of 200:1. The RM cells in the RM queue are serviced with the highest priority.

4.5.3.2 Switch congestion control

Despite routing cells to their destination, the principle function of ATM switches is to monitor congestion on the switches and provide feedback to the ABR sources. Due to the bursty nature of VBR traffic, the instantaneous bandwidth available to ABR connections varies with time and may cause congestion and cell loss. The switches in a network shall reduce congestion and keep cell loss to minimum while efficiently allocating the network bandwidth.

Upon the reception of a BRM cell, switches invoke a congestion control algorithm which provides feedback information to the ABR sources using RM cells. Several congestion control schemes have been proposed and the two common forms are binary feedback system and explicit-rate feedback system.

Binary feedback system is feedback based on a binary condition, normally a flag to indicate *congested* or *not congested.* The Explicit Forward Congestion Indication (EFCI) bit in the ATM header is used for this purpose. By setting the EFCI bit, switches on the path from source to destination notify the destination that congestion was experienced. Switches monitor their queue lengths and set EFCI when they exceeded a predefined threshold. Sources realize network congestion by observing the EFCI bit in the returning RM cells, then they adjust their sending rates using an additional increase/multiplicative decrease algorithm. However, binary schemes can be unfair since the penalized source may not be the one causing the congestion.

Explicit-rate schemes provide sources with the actual rates at which to send cells. Explicit-rate schemes converge faster than binary schemes and therefore more suitable for high speed networks. Typical explicit-rate scheme performs the following:

1. Determine load on switch by either monitoring the queue length or the queue growth rate.

2. Compute the fair share of the bandwidth for each VC that can be supported.

3. Determine the explicit rates and send these information to the sources through RM cells.

Examples of explicit rate schemes are the Enhanced Proportional Rate Control Algorithm (EPRCA) and Explicit Rate Indication for Congestion Avoidance (ERICA).

4.5.3.3 EPRCA

EPRCA [3] operates at each output port of the switch. The RM cells sent by the source contain a desired explicit rate (ER), the current cell rate usually set to the ACR and the CI bit set to zero. The switch computes a mean allowed cell rate (MACR) for all VCs using a running exponential weighted average. MACR is computed by

$$MACR = \frac{15}{16}MACR + \frac{1}{16}CCR$$

The switch monitors its load by keeping track of the queue length. When the RM cells return from the destination the switch reduces ER if it experienced congestion. The *fair share* is computed to be a fraction of 7/8 of the MACR

$$fairshare = \frac{7}{8}MACR$$

Program 4.2 EPRCA upon reception of a backward RM cell

```
MACR = 15/16 * MACR + 1/16 * CCR;
fair share = 7/8 * MACR;

if ( queue_length >= HIGH ) {
   CI = 1;
   NI = 0;
} else if (( queue_length < HIGH) & (queue_length > LOW)) {
   CI = 0;
   NI = 1;
} else {
   CI = 0;
   NI = 0;
}

ER = MIN ( ER, fair share );
```

Program 4.2 shows the operation of EPRCA at the reception of a BRM cell. EPRCA sets the CI and NI bits based on the ABR queue length of the port on which the BRM cell was received. The value of LOW and HIGH thresholds are set to 200 cells and 300 cells respectively.

If the calculated *fair share* is greater than ER value in the BRM cell, then the ER is unchanged, otherwise the ER is set to the *fair share*. This ensures that if BRM cell traverses more than one switch, the minimum of the *fair share* reaches the source. Thus avoids congestion at the bottlenect switch.

EPRCA is generally a congestion reaction scheme which is computationally simple. However, its congestion detection mechanism, which is based on queue length, has been shown to be unfair. A better approach is ERICA, a congestion avoidance scheme which used queue growth rate as load indicator. ERICA is not implemented and hence will not be discussed here.

Chapter 5

Evaluation

5.1 Overview

This chapter evaluates and compares the performance of GTW to ProTEuS in terms of speedup, scalability on network size, as well as the impact of network characteristics and simulation parameters. The next section validates the network modules implemented. A fairly complex topology is used to generate detail network results to be compared to those of ProTEuS. This ensures correct simulation results. The rest of the sections focus on large-scale ATM network topologies in performing further performance study.

Experiments in this chapter were performed on an 8-processor Sun Enterprise server known as *Clipper* located at the Lawrence Berkeley National Laboratory. Important information on *Clipper* is listed in Table 5.1

Node name	Clipper
Platform	SUN Ultra-Enterprise
Operating System	SunOS 5.7
Number of CPU	8
CPU clock speed	168 MHz

Table 5.1: Clipper system information

5.2 Validation of GTW models for ATM Simulation

This section verifies the correctness of modules implemented by comparing network results to those of ProTEuS using an ATM network topology with three switches, five hosts as configured in Figure 5.1. There are 2 ABR and 2 VBR streams.

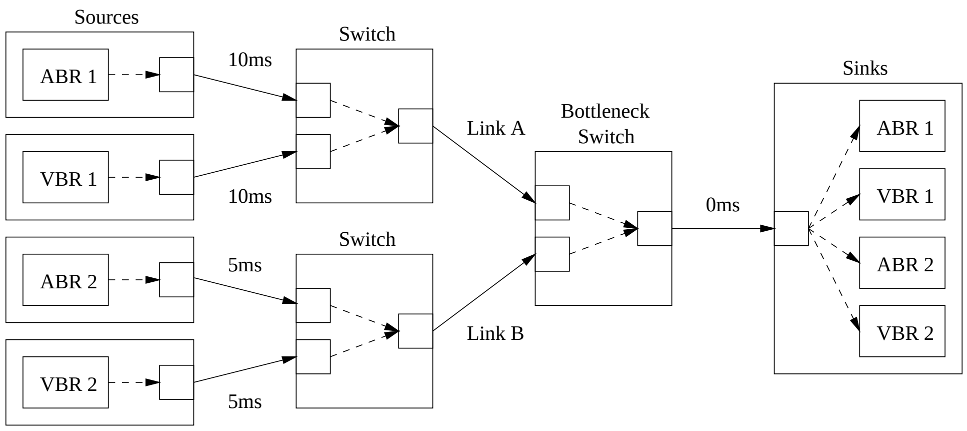

Figure 5.1: 3-switch, 5-host topology

Three sets of experiments have been run with different link delays from the two edge switches to the bottleneck switch. The three sets of delays on link (A,B) are: (5ms, 20ms), (15ms, 15ms), (20ms, 5ms). Important system parameters are explained in Table 5.2. Congestion was experienced at the output port of the bottleneck switch. This exercises the EPRCA algorithm.

Line rate, Switching rate	8000 cps
ABR sources	Greedy PCR=8000, ICR=1000, MCR=0 cps
VBR sources	Bursty MPEG video traffic Average cell rate = 3000 cps

EPRCA queue threshold	Low = 200 cells
	High = 300 cells

Simulated time	50 seconds

Table 5.2: System parameters

5.2.1 Link Utilization

Table 5.3 compares the mean normalized link utilizations on link A and B. In the first set of experiment with link delays (A:5ms, B:20ms), the round trip time (RTT) of ABR 1 is 2x(10ms+5ms) = 30ms while ABR 2 has a RTT of 2x(5ms+20ms)=50ms. Due to the fact that ABR 1 could increase its rate at a faster pace, we expected link A would have a higher link utilization over link B. This was confirmed by the experiment results depicted in Table 5.3. In the other two sets of experiments, the RTT of link B was higher. We observed similar behavior where link B has better link utilization. The results from GTW and ProTEuS were very close and shown similar trend.

	Link A		Link B	
Experiment	GTW	ProTEuS	GTW	ProTEuS
A:5ms B:20ms	0.502	0.503	0.498	0.497
A:15ms B:15ms	0.498	0.499	0.502	0.501
A:20ms B:5ms	0.498	0.499	0.502	0.501

Table 5.3: Mean Normalized Link Utilization

5.2.2 Mean Queueing Delay

Table 5.4 compares the mean queueing delay experienced by both ABR cell streams at the bottleneck switch. The queueing delays were generally increase as the RTT increased. Similar behavior was observed in both GTW and ProTEuS, and their queueing delays were off by less than 2 % .

	ABR 1 queuing delay (sec)		ABR 2 queuing delay (sec)	
Experiment	GTW	ProTEuS	GTW	ProTEuS
A:5ms B:20ms	0.159	0.156	0.164	0.163
A:15ms B:15ms	0.165	0.163	0.161	0.160
A:20ms B:5ms	0.167	0.165	0.159	0.157

Table 5.4: Mean ABR Cell Queuing Delay

5.2.3 ABR Queue Length

We also compared the ABR queue length at the bottleneck port over the first 10 seconds. The delay on links (A,B) were (15ms,15ms) in this case. Figure 5.2 shows the ABR queue length curves for GTW and ProTEuS. The EPRCA was operating on the bottleneck switch with a high and low threshold of 300 and 200 cells respectively. Whenever the ABR queue grew beyond 300 cells, all ABR sources were throttled down until the ABR queue accumulation dropped below the low threshold. Then the sources began to throttle up and the queue grew again. The evidence of this repeated activity can be observed from the two curves in the figure moving up and down between the high and low thresholds. Again, ProTEuS and GTW were getting very close results.

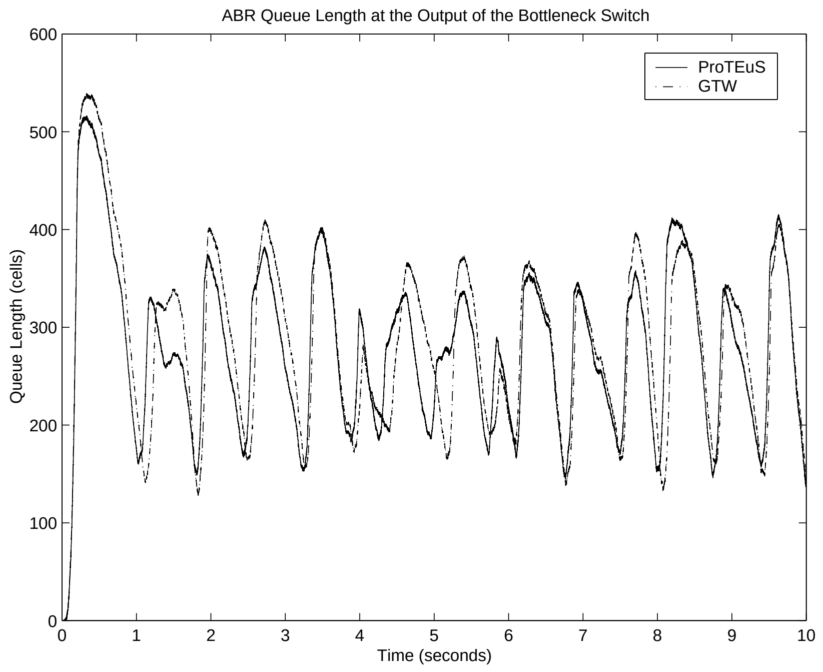

Figure 5.2: ABR Queue Length (zoom)

5.2.4 ABR Source Rate

Figure 5.3 compares the ABR source rates of ABR source-1 over the first 10 seconds. The delays on links (A,B) were (15ms,15ms). Since the switches were allowed to process at 8000 cells/s, the total bandwidth available to all sources was 8000 cells/s. The average cell rate of VBR source was 3000 cells/s. Hence, the two VBR sources occupied 6000 cells/s and left 2000 cells/s for the two ABR sources. Each ABR source would receive an average bandwidth of 1000 cells/s. Figure 5.3 shows the ABR source rate fluctuating on the average rate of approximately 1000 cells/s.

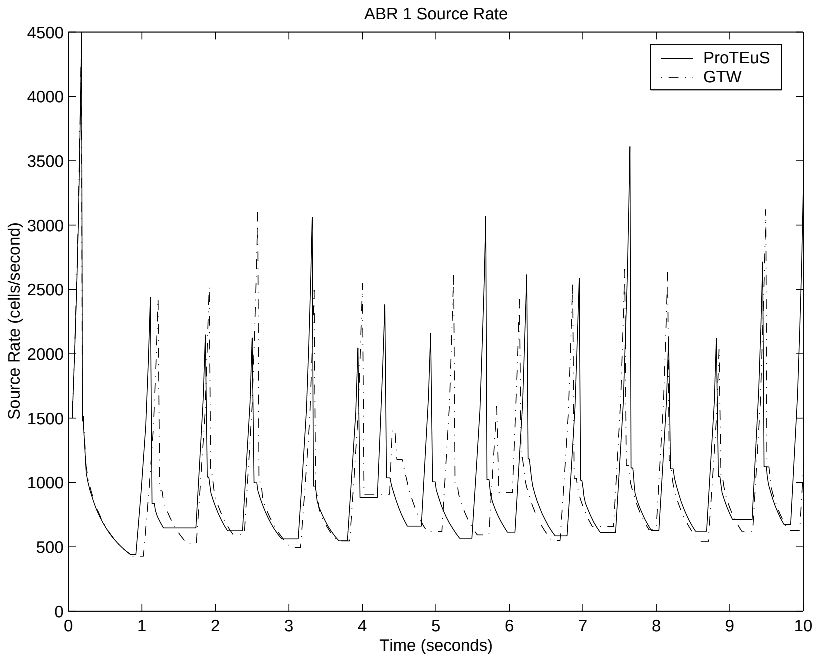

Figure 5.3: ABR Source Rate (zoom)

5.2.5 Execution Time

Table 5.5 lists the execution times of all simulations performed. The performance of GTW was slightly better than ProTEuS. However, the scale of the network topology considered here was small and the line rate was low. The situation does change as the network size grows.

Similar experiment has been done using BONeS, a sequential discrete-event simulator, and the execution times are included in the table to contrast the advantage of GTW and ProTEuS.

Experiment	Execution Time (HH:MM:SS)		
	BONeS	GTW	ProTEuS
A:5ms B:20ms	01:40:36	00:01:24	00:01:28
A:15ms B:15ms	01:40:36	00:01:23	00:01:28
A:20ms B:5ms	01:40:36	00:01:22	00:01:28

Table 5.5: Execution Time

5.3 GTW performance on ATM network simulation

This section focuses on the speedup achievable on ATM network simulation using multiple processors. Two large-scale network scenarios are set up to study GTW's performance.

5.3.1 Scenario A: 2 cores, 4 edges, 40 hosts

The first scenario comprised of 2 core ATM switches, 4 edge ATM switches, and 40 hosts was configured as shown in Figure 5.4. The configuration of the topology is symmetric, as are the traffic flows. Each edge switch was connected to ten hosts: 5 VBR connections and 5 ABR connections. The line rate for all links was OC-3 rate (i.e., 155 Mbps). Switch was allowed to switch at OC-3 rate. All links have a delay of 5ms and thus the RTT is 50ms plus queuing and transmission overheads.

The behaviors of the different sources are described in Table 5.6.

ABR sources	Greedy
	PCR = 21000 cps
	ICR = 25% PCR
	MCR = 0 cps

VBR sources	50% square wave
	period = 100 ms
	MAX = 15000 cps
	MIN = 10000 cps

TCP source	Greedy
TCP layer	Window size = 512 KBytes TCP Processing time = 1 ms

Table 5.6: Scenario A: Source parameters

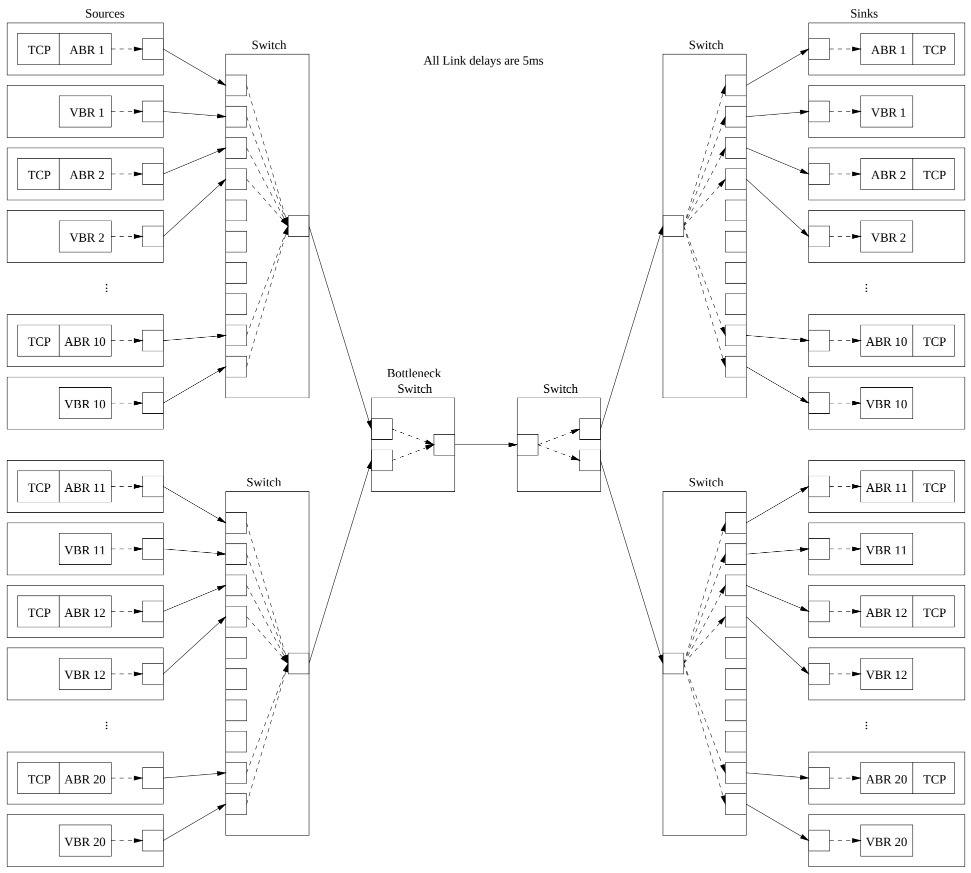

Figure 5.4: 6 Switch 40 Host Topology

The primary performance metric to be studied here is the execution times of simulations. The scenario was simulated with four different loads. For each load, three simulations were run utilizing 2, 4, and 6 processors respectively. Running simulation on all 8 processors available on *Clipper* is not recommended since one processor is always dedicated to operating system's kernel processes.

The four loads are distinguished by traffic types and traffic directions. Two traffic types are raw ABR traffic and TCP over ABR traffic. And the two traffic directions are: 1.) uni-directional traffic where data cells flow from source to destination in one direction; 2.) bi-directional traffic where data cells flow in both forward and backward directions. Bi-directional traffic roughly doubled the load on the network, and hence the number of simulated events and execution time would be doubled. The simulated time for each simulation is fixed at 10 seconds.

Table 5.7 tabulate the numerical results from the simulations. The histograms in Figure 5.5 shown the results visually.

Experiment	# Processors	Execution Time (seconds)	
		GTW	ProTEuS
Uni-directional	2	1551.75	1191.32
Traffic	4	1228.38	1213.28
ABR only	6	610.33	1055.88
Bi-directional	2	2622.97	1548.12
Traffic	4	2177.81	1540.79
ABR only	6	1134.29	1221.99
Uni-directional	2	1600.48	1234.22
Traffic	4	1649.88	1243.12
TCP over ABR	6	663.93	1070.77
Bi-directional	2	3016.11	1540.10
Traffic	4	2730.70	1502.08
TCP over ABR	6	1488.28	1200.08

Table 5.7: Scenario A: Execution Time

The numbers shown in Table 5.7 do not intend to indicate that one is faster than the other since GTW and ProTEuS operate on fundamentally different hardware architectures with unmatched CPU processing power, different inter-processor communication overheads, etc. A more powerful multiprocessor machine for GTW or PCs for ProTEuS can dramatically improve the outcomes of the experiment. Indeed, we focus on the trends on the result.

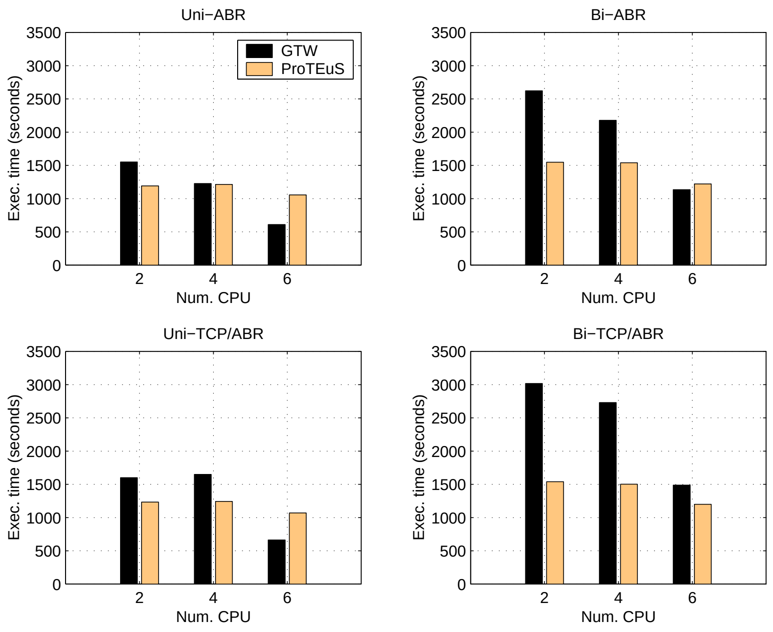

Figure 5.5: Scenario A: Execution time vs Number CPU

One major trend we observed is that ProTEuS scales better with traffic load compared to GTW. The traffic load on the network was doubled in the bi-directional cases. GTW required nearly twice the execution time to complete the bi-directional cases compared to uni-directional cases. Whereas ProTEuS required only 20% longer execution time. Another observed trend is that GTW achieved more speedup with more number of processor. This indicates that GTW can exploit more parallelism inherited in large-scale networks.

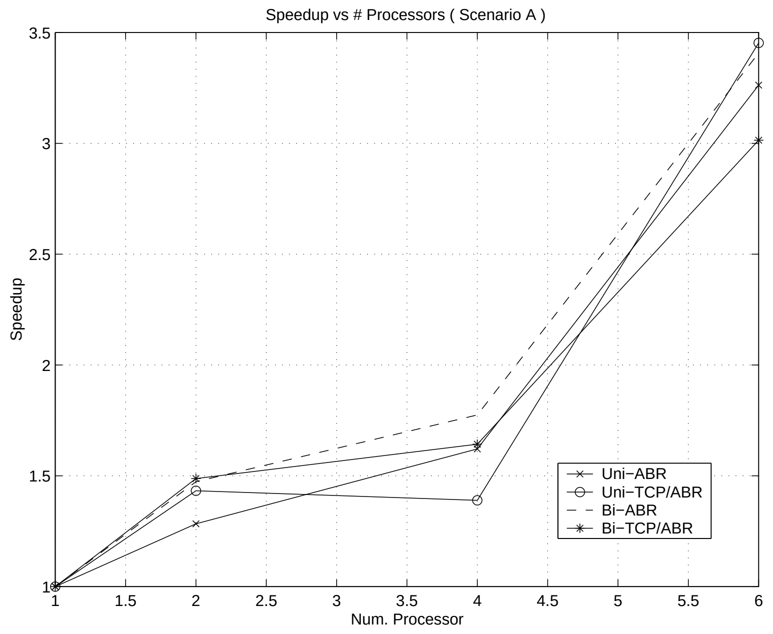

Figure 5.6: Scenario A: GTW Speedup (relative to 1 processor)

The GTW's achieved speedup for all four loads are plotted in Figure 5.6. We were able to gain a speedup close to 3.5 on the simulations run on 6 processors. The reported speedup of parallel simulation of ATM network in Telesim project described in [31] is below 3.0 on simulation using 8 processors. In general, the speedup depends on the size and configuration of a network topology, as well as the partitioning of LPs. Results from Telesim described in [31] were performed using realistic scenarios which were asymmetric in topology and hard to be partitioned. Hence, lower speedup gain was expected. Another important observation from the Telesim results shown that the parallel simulation speedup improves with the relative size of the ATM network scenario executed, as well as with the relative traffic load on each scenario.

In Figure 5.6, the observed speedup when using 4 processors was low as expected. The reason is that the partitioning of LPs onto 4 processors was difficult to achieve good load balancing.

5.3.2 Scenario B: 4 cores, 12 edges, 120 hosts

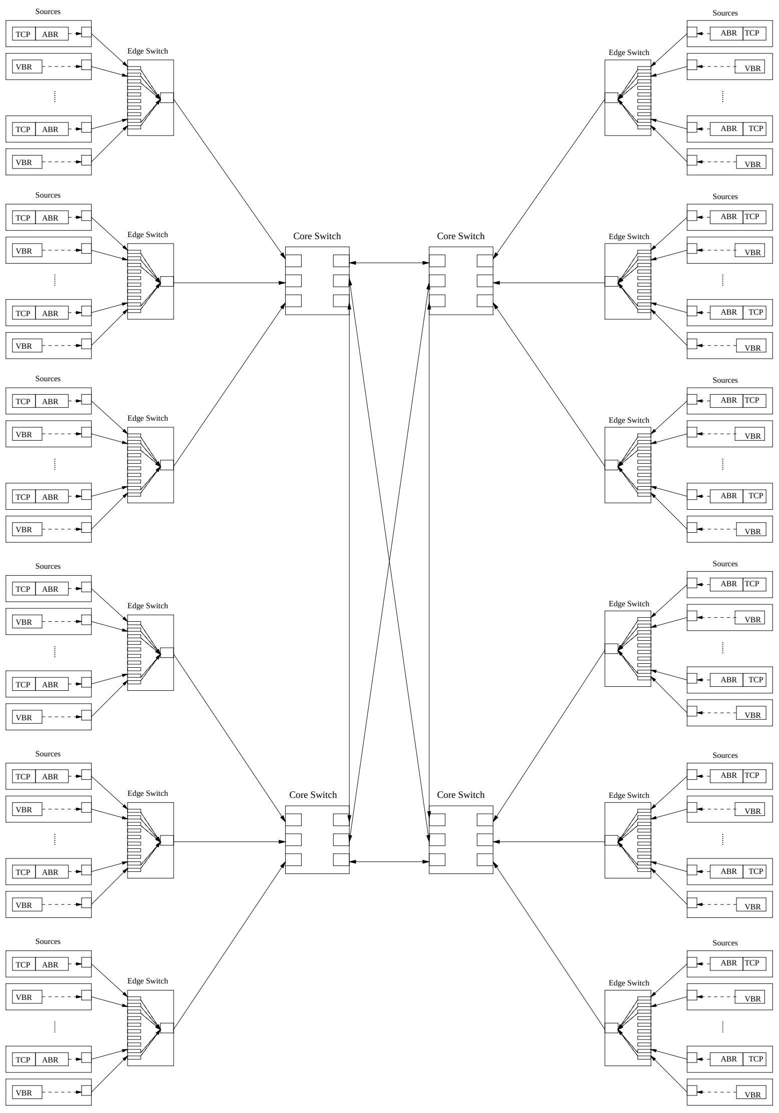

Figure 5.7: 16 Switch 120 Host Topology

This larger scenario comprises four core switches and twelve edge switches. The network topology is shown in Figure 5.7. The four core switches were interconnected and each core switch was attached to three edge switches. Every edge switch was connected to 10 end-hosts consisting 5 ABR sources and 5 VBR sources. Again, the network topology was designed to be symmetric. All links were OC-3 rate. The link delay of every link was 5 ms and hence the RTT of all connections was 50 ms.

The behaviors of the different sources are described in Table 5.8.

ABR sources	Greedy
	PCR = 36000 cps
	ICR = 25% PCR
	MCR = 0 cps

VBR sources	50% square wave
	period = 100ms
	MAX = 36000 cps
	MIN = 10000 cps

TCP source	Greedy
TCP layer	Window size = 128 KBytes TCP Processing time = 1 ms

Table 5.8: Scenario B: Source parameters

Simulations with different loads that have been performed on the previous scenario were repeated on this scenario. The only change made was that we reduced the simulated from 10 seconds to 1 second. The reduced simulated time would still yield stable results since this larger scenario would execute approximately six times more events than the previous scenario. Table 5.9 tabulates the numerical results from the simulations. The histograms in Figure 5.8 compared the results visually.

Experiment	# Processors	Execution Time (seconds)	
		GTW	ProTEuS
Uni-directional Traffic *ABR only*	2	762.88	327.14
	4	385.36	239.43
	6	298.47	178.87
Bi-directional Traffic *ABR only*	2	1569.48	527.29
	4	851.44	335.64
	6	662.42	257.88
Uni-directional Traffic *TCP over ABR*	2	784.74	349.75
	4	425.72	241.39
	6	331.21	178.66
Bi-directional Traffic *TCP over ABR*	2	1535.20	549.22
	4	871.57	327.24
	6	668.90	251.07

Table 5.9: Scenario B: Execution Time

Unlike the previous scenario, ProTEuS outperformed GTW with a larger margin in this bigger scenario. Again, ProTEuS scales better when the traffic load was doubled. GTW is better than ProTEuS in exploiting parallelism (i.e., gained more speedup with more processors).

The GTW's speedup curves for all four loads shown in Figure 5.9 are near-linear. The speedup gained are better compared to the smaller scale scenario A. Speedup using 6 processors could reach over 4.0 on two of the cases. This better speedup matched the observation from the Telesim project mentioned in previous section: the parallel simulation speedup improves with the relative size of the ATM network scenario executed, as well as with the relative traffic load on each scenario.

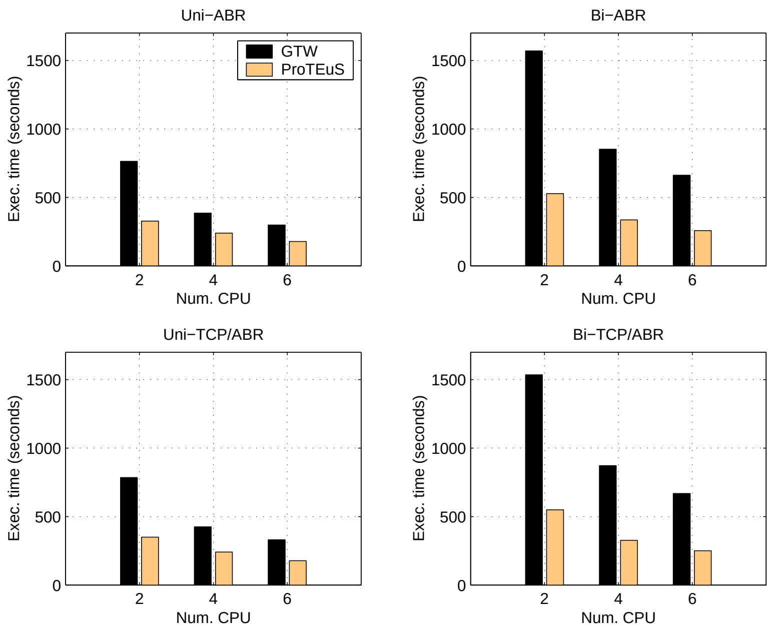

Figure 5.8: Scenario B: Execution time vs Number CPU

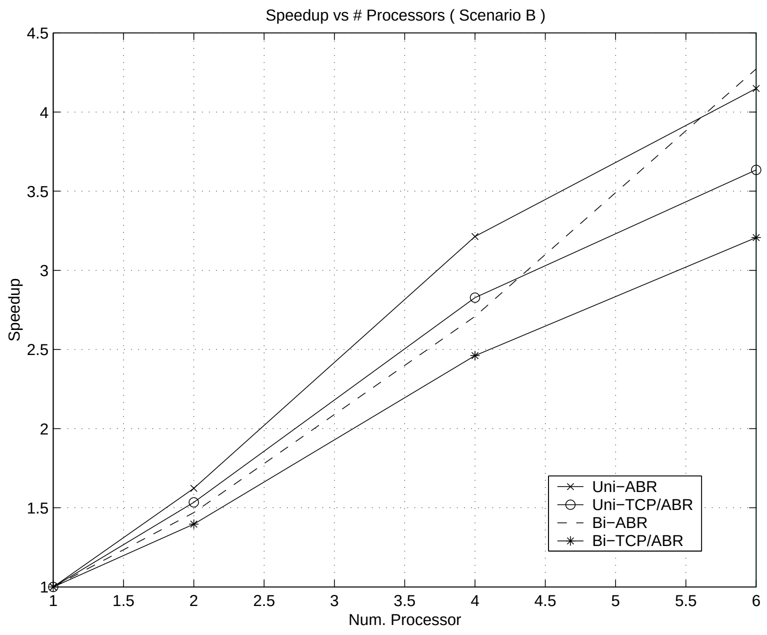

Figure 5.9: Scenario B: GTW speedup (relative to 1 processor)

5.4 Effect of network characteristics on GTW performance

5.4.1 Effect of feedback loop in network: Rollbacks degrade GTW performance

This section studies how network feedback control loops increase rollback activity and impact GTW's performance. Further experiments have shown that the rollback percentage (i.e., percentage of processed events that are rolled back) not only depends on the interaction among network components (i.e., communication pattern among LPs), but several simulation parameters have impacts on the performance too. The most critical factor is the event memory allocation.

All simulations in this experiment were using the 6-switch model with link delay set to 5 ms. Simulated time was 10 seconds in each simulation.

Again, the four different loads experiment mentioned before was used in studying rollback activity due to network control loops. We suspect the *uni-ABR* load would have less rollback activities since there was only one feedback loop (i.e., EPRCA closed loop control) operating. *Uni-TCP* load was subjected to an additional TCP control loop. *ACK* packets in the opposite direction have the potential to cause more rollbacks. Both *bi-ABR* and *bi-TCP* loads have bi-directional traffic and hence would trigger the most rollbacks.

Figure 5.10 shows how rollback percentage depended on the event memory allocation. In general, excessive event memory allocation would degrade GTW's performance. Despite the dependency of rollback percentage on event memory, the curves in Figure 5.10 confirmed our hypothesis. *Uni-ABR* load with little backward traffic was causing less rollback activity. *Uni-TCP* load has twice as much rollback activity than the *Uni-ABR* load. *Bi-ABR* and *Bi-TCP* loads with heavy backward traffic have the most rollback percentage.

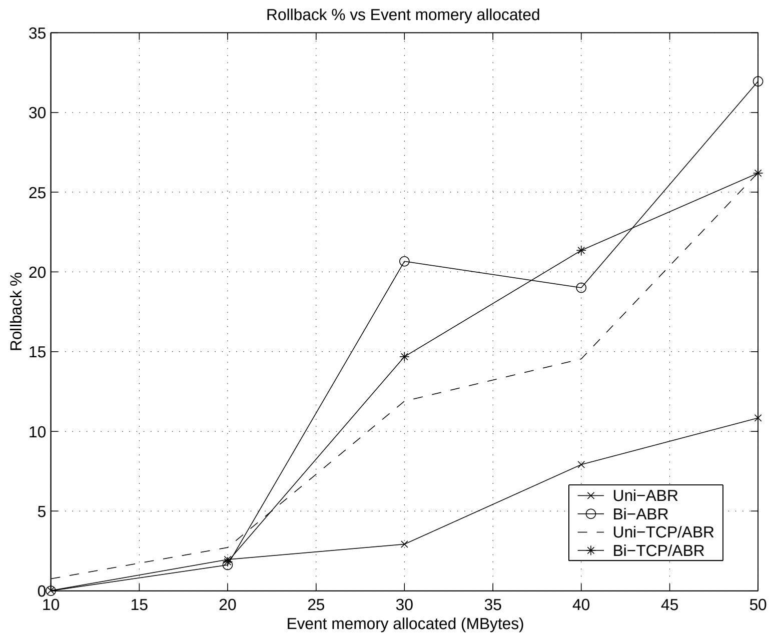

Figure 5.10: Rollback percentage vs Event memory allocated

As mentioned before, excessive event memory degrades GTW's performance. 10 MB was the optimum event memory allocation for most of the simulations except for the cases with *bi-TCP* load. Event memory less than 10 MB would led to memory starvation and simulation halted as a result. This observation can be explained by studying the number of aborted events (i.e., event aborted due to memory starvation, and a fossil collection will be called to reclaim memory) and the number of fossil collection shown in Figure 5.11 and Figure 5.12 respectively. As the event memory allocated increased, there was less number of aborted event since the allocated memory was sufficiently large. Consequently, less aborted event reduced the number of *fossil collection* as we can observe the drop in Figure 5.12. The process to abort an event actually slowed down the moving pace of an LP. In other words, LPs advanced slower when the event memory allocated was small amount, causing more aborted events. This slow down in fact helped to decrease the number of potential rollback. On the other hand, as the event memory allocated was large, individual LP could advance its local virtual time in a faster pace since it was not experiencing much delay from aborted events. The larger gap in local clocks among LPs could potentially lead to more rollback activity.

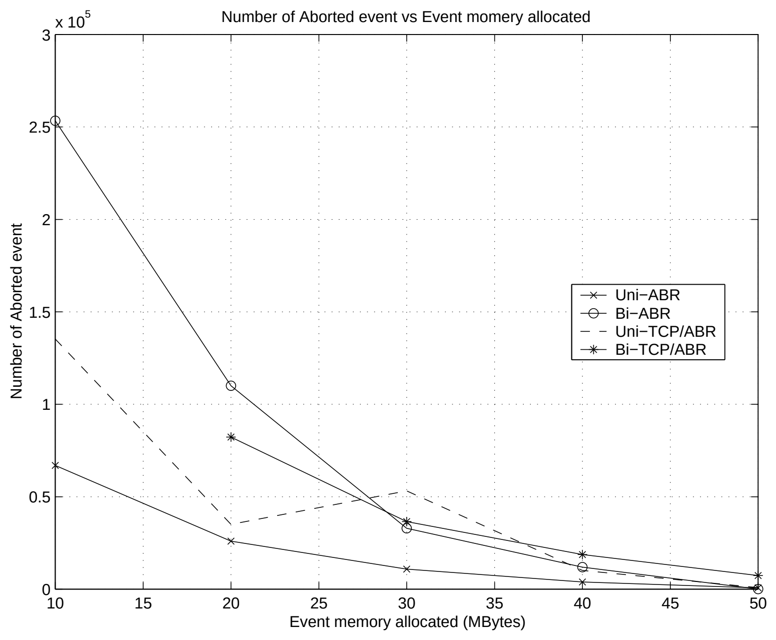

Figure 5.11: Number of aborted event vs Event memory allocated

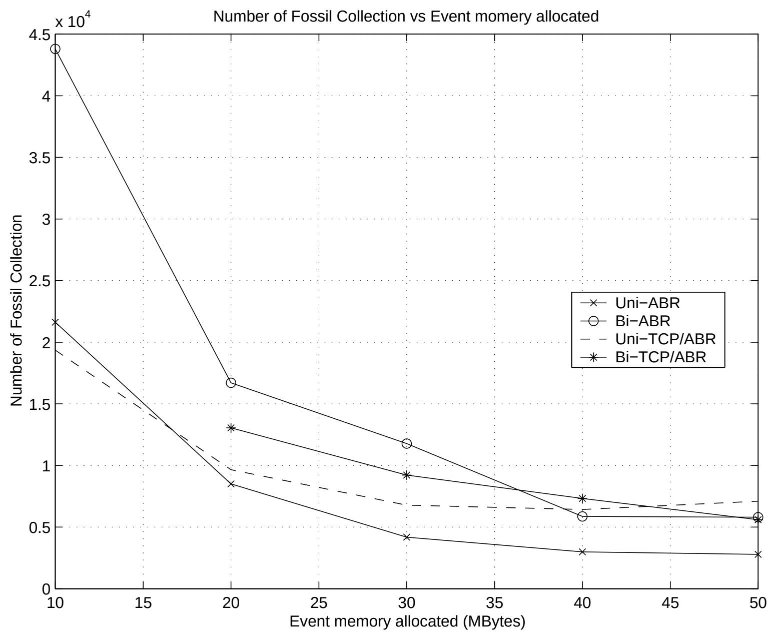

Figure 5.12: Number of fossil collection vs Event memory allocated

The cost of rollback activity is high. The performance of GTW worses as the rollback percentage increases when the event memory allocation becomes excessively high.

5.4.2 Effect of round trip time (RTT)

This section studies how network with long RTT affects GTW's performance. All simulations have been performed using the 6-switch scenario A. Six processors were used for all simulation runs. The delay on each link was varied from: 1, 5, 10, 20, to 40 ms. Hence, the RTT was varied from: 10, 50, 100, 200, to 400 ms. The 400 ms RTT is realistic considering satellite links.

Variation on the RTT will vary the traffic load on the network with a fixed simulated time, especially in the TCP cases. To ensure a constant load on the network across all simulation runs with different RTTs, we fixed the load by requiring all sources to transfer a fixed amount of traffic as defined in Table 5.10. All simulations with varying RTT simulated equal number of events. The processing load on each processor was almost equivalent and hence we expected the execution times for all simulations would be similar despite the effect of RTT.

VBR source	Send exactly 100,000 cells
ABR source	Send exactly 100,000 cells
TCP source	Send exactly 5,120 KBytes

Table 5.10: Source traffic policy

However, results shown in Figure 5.13 give evidence of the effect of changing RTT on GTW's performance. The performance degraded as the RTT was increased. This behavior was amplified in the TCP cases. The RTT is obviously a factor impacting GTW's performance, yet the exact reason for this behavior is not known at this stage. One observation on the simulations was that the memory consumption increased tremendously as the RTT was increased. Larger memory requirement degrades performance. The real culprit behind this large memory consumption will be left as future work.

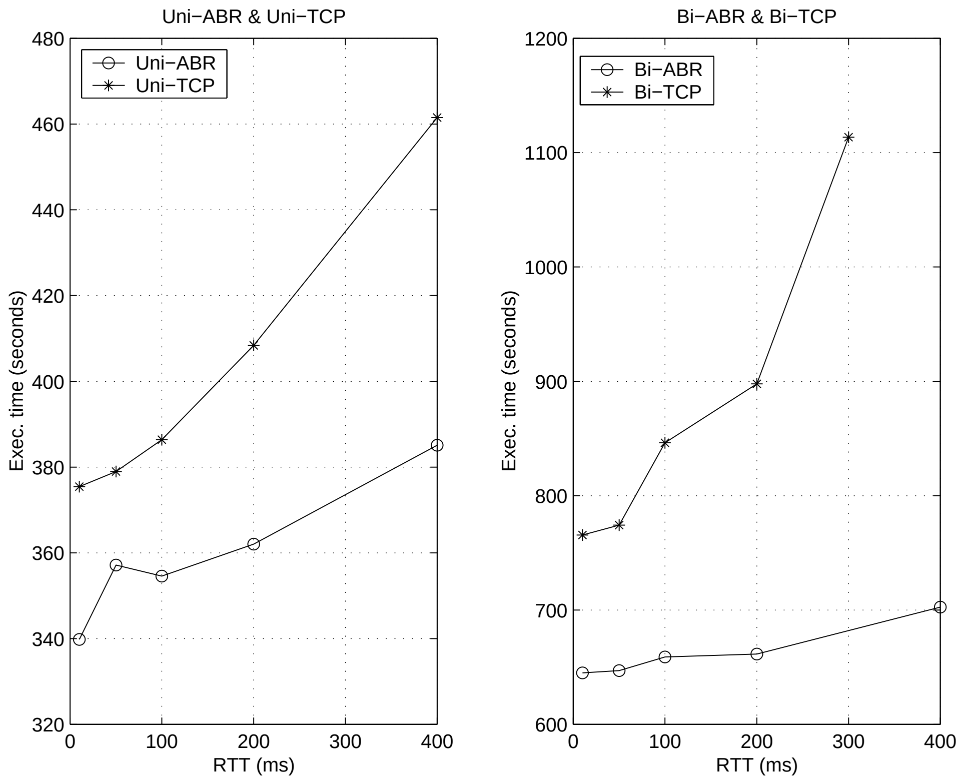

Figure 5.13: Scenario A: Execution time vs. RTT (fixed load)

Experiment	RTT (ms)	Execution Time (seconds)	
		GTW	ProTEuS
Uni-directional Traffic *ABR only*	10	330.44	496.00
	50	333.67	513.50
	100	341.49	513.46
	200	366.40	518.33
	400	398.47	533.03
Bi-directional Traffic *ABR only*	10	609.16	585.95
	50	649.27	603.76
	100	651.15	602.05
	200	708.41	617.11
	400	857.33	643.14
Uni-directional Traffic *TCP over ABR*	10	307.28	565.12
	50	330.40	595.94
	100	349.24	633.44
	200	389.14	702.97
	400	470.61	858.66
Bi-directional Traffic *TCP over ABR*	10	574.69	656.51
	50	648.79	684.89
	100	703.38	754.35
	200	835.71	802.07
	400	1017.36	1098.67

Table 5.11: Execution time vs. Round Trip Time (RTT)

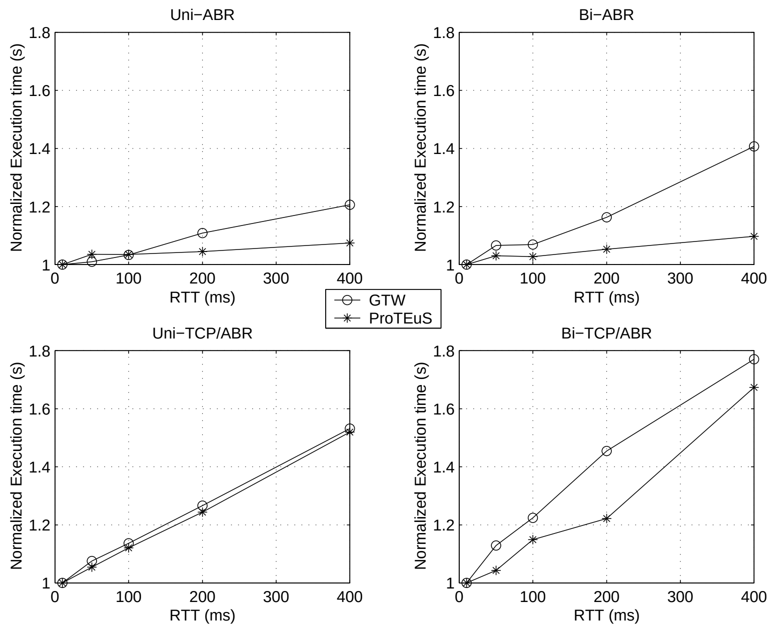

Figure 5.14: Scenario A: Execution time vs. RTT

In order to compare the performance of GTW and ProTEuS in terms of RTT. The approach to fix the load is not fair to ProTEuS in experiment with TCP. The TCP sources and VBR sources finish their data transmissions at times that are far apart (due to different RTT) in the simulation where the network would experience low bandwidth utilization during the gap. This uneven traffic load during the course of simulation favors GTW but hurts ProTEuS performance. Hence, we repeated the experiment where we fixed only the ABR or TCP source, not the VBR source. This change should still yield meaningful results.

The new results are tabulated in Table 5.11. Figure 5.14 shows the execution times normalized to the 10ms case. The impact of RTT on ProTEuS is less obvious in the *ABR-only* cases. However, longer RTT with the TCP layer exists can affect ProTEuS' performance as bad as GTW's.

5.4.3 Effect of Network Size

This section studies how well GTW scales with network size using six processors. Scenario A and B described in previous sections were used. Scenario A has 6 switches and 40 hosts while scenario B has 16 switches and 120 hosts. The network size is roughly tripled. However, the computed traffic loads on both scenarios has a difference of 5.3 times since the sources in scenario B were sending at higher rates. The RTT was 50 ms and the simulated time was fixed at 10 seconds.

The execution times of all simulations are tabulated in Table 5.12. Observe in Figure5.15 that the execution times for the two scenarios differ by a factor of approximately 6 times in the uni-directional cases whereas 7 times in the bi-directional cases. The increment is more than linear since there are more overheads (i.e., more LPs) involved in larger scenario in addition to the increased size and load. On the other hand, ProTEuS required only twice the execution time when we multiplied the network size and traffic load.

Note that attempts to simulate scenario B with *Bi-TCP* load for 10 seconds failed due to event memory starvation (i.e., insufficient physical and virtual memory available on *clipper* to complete the simulation). The error message given by the GTW kernel was ”same GVT in a row for 100000 times”. This failed case can be explained as the following: Consider a successful simulation. A processor has local virtual time (LVT) = x. The event memory allocated is sufficient enough to comfortably accommodate new events (i.e., later become unprocessed events). After processing the next unprocessed event, LVT advanced to (x+increment). Given that all processors in the simulation behave the same, GVT can be advanced and fossil collection can reclaim memory. The event queue shrank and there is more room for future events. Consider a larger network scenario where the #LP per processor increases. Also, the traffic load increases which results in more events per LP (i.e., more processed events needed to be saved tp permit rollback). At some point of a simulation, the processed events kept in event queue can dominate the event memory allocated. There are no longer enough spaces for new events and these new events are likely to be aborted. The unprocessed event queue can become empty when the situation worsen. No new event can be admitted and hence no unprocessed event can be processed to advance LVT. During GVT computation, old GVT = new GVT. Failure to advance GVT results in zero memory reclaimed on this particular processor. The phenomenon persists and the entire system falls into an unproductive stage (deadlock). After failing to advance GVT for so many trials (100000), the simulator gave up and halted the simulation. Note that similar phenomenon has been experienced on a smaller scale simulation when the event memory allocation was too little. The solution is to allocate more event memory. However, when the available 850 MBytes of physical memory and 2.5 GBytes of virtual memory on the machine were fully consumed in this case study, there was no extra memory could be allocated. Hence, the simulation could not be completed.

In general, the execution time increases as the network size and traffic load increases. Yet ProTEuS' scalability on network size is better than GTW.

		Execution Time (seconds)	
Experiment	Network size/scenario	GTW	ProTEuS
Uni-ABR	A	610.33	1055.88
	B	3520.28	1754.40
Bi-ABR	A	1134.29	1221.99
	B	7845.38	2528.08
Uni-TCP	A	663.93	1070.77
	B	3834.70	1873.59
Bi-TCP	A	1488.28	1200.08
	B	*N/A*	2579.70

Table 5.12: Execution Time vs network size

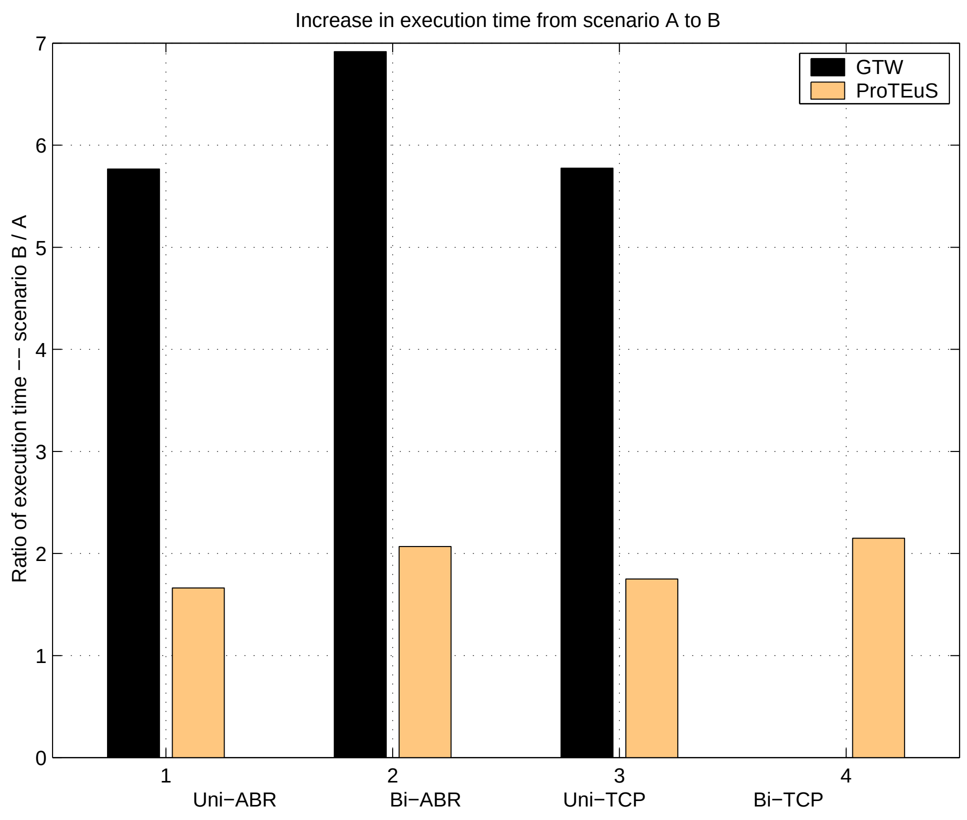

Figure 5.15: Execution time vs. network size

5.5 Effect of simulation parameters on GTW performance

5.5.1 Event memory and state memory allocation

We have shown the negative effect of event memory allocation in previous study. The simulation parameter should be tuned to achieve better performance. This issue will be revisited here to include the effect of state memory allocation. State memory is used in GTW to keep saved state vector that ensures correct rollback.

Again, the 6-switch model was used in this experiment. All simulations were performed on 6 processors, 50 ms RTT, using *uni-TCP* load. The simulated time was 10 seconds.

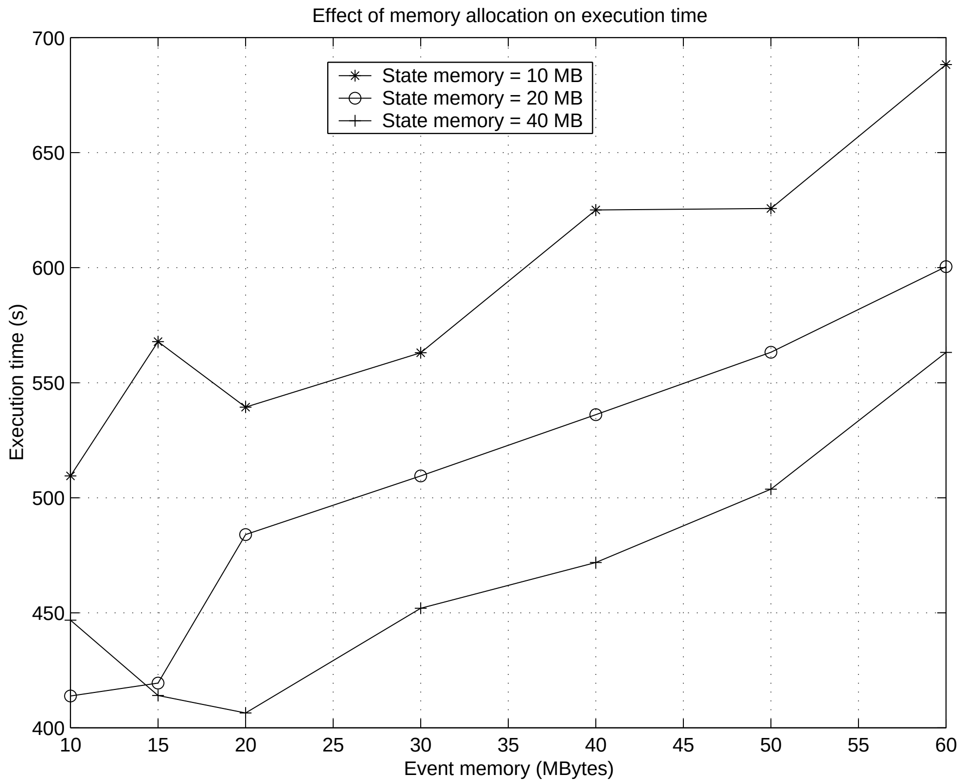

Figure 5.16: Execution time vs. event memory allocated

As shown in Figure 5.16, the event memory was varied from 10 MB to 60 MB. The performance worsen (i.e. execution time increased) as the event memory increased. This behavior was studied and explained before. However, the state memory has positive effect on GTW.

GTW performed better as more state memory was allocated to the simulations. Analysis on simulation statistics shown that less state memory would require the simulator to pause for more memory allocation during the course of simulation. More pre-allocated state memory prevents this extra overhead and thus increases performance.

Chapter 6

Conclusions and Future Work

In this thesis, the performance of an optimistic parallel simulation engine, GTW, is evaluated using large-scale ATM network. Models of ATM switches and end-hosts have been implemented using the modeling interface, provided by GTW, which separates the programmer from the simulator kernel. The modeling framework provides programmers a convenient and structural way to define the state and behavior of an object in sections. The communication among LPs are done trivially by sending message (i.e., event scheduling).

As we have demonstrated, ATM network simulation can be done 4 times faster using 6 processors on a multiprocessor machine. To an extent, general network simulation can truly benefit from time warp's optimistic parallel simulation. However, load balancing is an important issue to be tackled before an optimum speedup can be achieved. Imbalanced load during the course of simulation often leads to poor speedup since idle processors would not constribute to the overall speedup. The current version of GTW requires static mapping of LPs to processors. A careful mapping plan must be done by user to achieve good speedup. Automatic LP-to-processor mapping mechanism has been proposed to make parallel simulator more user-friendly, but such mechanism is not yet mature. Another major difficulty in performing simulation on GTW is that users are required to tune several simulation parameters in order to ensure a simulation executes successfully. Several major parameters required to be defined are the event memory, state memory, local communication percentage, GVT period, etc. Insufficient event memory will break a simulation and there is no systematic way to predict the event memory requirement (i.e., event memory requirement depends on the LPs' state sizes, network characteristics such as

RTT, etc). Experience is often the tool in determining the event memory requirement. Further tuning, which is very time consuming, is required to achieve optimum performance. Similarly, ProTEuS needs to address the issue of mapping to achieve load balancing. The value of *epoch* [17] must be tuned to a sufficient minimum in order to reduce execution time.

In general, our evaluation has shown that GTW can effectively utilize the available processors to achieve great speedup. As the network size grows, more parallelism can be exploited using more processors. However, ProTEuS outperformed GTW in term of scalability in network size. Results also showed that network characteristics have significant impact on the performance of GTW. One characteristic we studied is network with feedback loops. We focused on ABR service as its rate control mechanism with feedback traffic can potentially induce rollbacks. An additional TCP layer on top of ABR can induce more rollbacks and furtherly degrade the performance. Figure 5.10 confirmed the hypothesis made. However, these feedback loops will never impact the performance of ProTEuS as they do to GTW since ProTEuS is free of rollback. Another network characteristic which affects the performance is the round trip time. Simulation with longer RTT requires more execution time. RTT has less impact on ProTEuS.

Interesting future work is discussed in below:

1. Explore new strategies to reduce the memory usage of the existing models since memory consumption limits the size of network scenario.

2. Simulate a realistic scenario (i.e, taken from real world carrier's backbone) which is often asymmetric in topology and loaded with various kinds of traffics. Study how well GTW can exploit parallelism to achieve satisfactory speedup by addressing the issue of LP-mapping.

3. Integrate an IP model and various networking technologies besides ATM to the existing collection of models. Study the feasibility of GTW on a more general network.

4. Experiment GTW on a network of workstation (NOW) platform which is more closely resemble the architecture of ProTEuS.

Bibliography

[1] http://www.cadence.com.

[2] http://www.opnet.com.

[3] ATM Forum Traffic Management Specification Version 4.0. ATM Forum Techincal Committee, Traffic Management Sub-Working Group, April 1996.

[4] I. Akyildiz, I. Joe, R. Fujimoto, and I. Nikolaidis. Parallel Simulation of End-to-End ATM Models. *Computer Networks and ISDN Systems Journal*, 29(6):617–630, May 1997.

[5] S. Bhatt, R. Fujimoto, A. Ogielski, and K. Perumalla. Parallel Simulation Techniques for Large-Scale Networks. *IEEE Communications Magazine*, pages 42–47, August 1998.

[6] C. Carothers and R. Fujimoto. Efficient Execution of Time Warp Programs on Heterogeneous, NOW Platforms. *IEEE Transactions on Parallel and Distributed Systems (pending revision)*.

[7] C. Carothers, K. Perumalla, and R. Fujimoto. Efficient Optimistic Parallel Simulations using Reverse Computation. In *1999 Workshop on Parallel and Distributed Simulations*, May 1999.

[8] C. Carothers, K. Perumalla, and R. Fujimoto. The Effect of State-saving in Optimistic Simulation on a Cache-coherent Non-uniform Memory Access Architecture. In *Proceedings of Winter Simulation Conference (WSC)*, 1999.

[9] K. Chandy and J. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Communications of the ACM*, 24(4):198 – 206, April 1981.

[10] J. H. Cowie, D. M. Nicol, and A. T. Ogielski. Modeling the global internet. *Computing in Science & Engineering*, 1(1):42–50, January/February 1999.

[11] S. R. Das, R. Fujimoto, K. Panesar, D. Allision, and M. Hybinette. GTW: A Time Warp System for Shared Memory Multiprocessors. In *Proceedings of the Winter Simulation Conference*, pages 1332–1339, December 1994.

[12] K. Fall. Network Emulation in the Vint/NS Simulator. In *ISCC 99*, 1999.

[13] R. Fujimoto. Optimistic Approaches to Parallel Discrete Event Simulation. *ACM Transactions of the Society for Computer Simulation*, 7(2):91 – 120, June 1990.

[14] R. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[15] R. Fujimoto, S. R. Das, and K. S. Panesar. *Georgia Tech Time Warp (GTW Version 2.3) Programmer's Manual*. Georgia Institute of Technology, April 1995.

[16] N. Golmie, M. Corner, Y. Saintillan, A. Koenig, and D. Su. *The NIST ATM/HFC Network Simulator (Version 3.0) Operation and Programming*. National Institute of Standards and Technology, March 1998.

[17] S. House. Proportional Time Emulation and Simulation of ATM Networks. Master's thesis, University of Kansas, December 2000.

[18] R. Jain, S. Kalyanaraman, S. Fahmy, R. Goyal, and S. Kim. Source Behavior for ATM ABR Traffic Management: An Explanation. *IEEE Communications Magazine*, 34(11):50 – 57, November 1996.

[19] D. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[20] W. LaRUE Jr. *New Algorithms to improve the efficiency of Parallel Discrete Event Simulation.* PhD thesis, University of Kansas, 1996.

[21] S. Kalyanaraman, R. Jain, S. Fahmy, and R. Goyal. Performance and Buffering Requirements of Internet Protocols over ATM ABR and UBR Services. *IEEE Communications Magazine*, 36(6):152–157, June 1998.

[22] S. Kalyanaraman, R. Jain, S. Fahmy, R. Goyal, J. Jiang, and S. Kim. Performance of TCP over ABR on ATM backbone and with various VBR traffic patterns. *Proc. ICC'97, Montreal*, 2:1035–1041, June 1997.

[23] E. Mascarenhas, F. Knop, and V. Rego. ParaSol: A Multithreaded System for Parallel Simulation based on Mobile Threads. In *Proceedings of the 1995 conference on Winter simulation conference*, pages 690 – 697, 1995.

[24] R. A. Meyer. *PARSEC User Manual (Release 1.1).* UCLA Parallel Computing Laboratory, August 1998.

[25] S. Murthy. A Software Emulation and Evaluation of Available Bit Rate Service. Master's thesis, University of Kansas, October 1998.

[26] R. Pasquini and Vernon Rego. Optimistic Parallel Simulation over a Network of Workstations. In *Proceedings of the winter simulation conference on Winter simulation (Vol. 2)*, pages 1610 – 1617, 1999.

[27] K. S. Perumalla. Transparent Incremental State Saving in GTW++ - Design, Implementation, Verification and Performance Evaluation. Technical report, Georgia Institute of Technology, September 1996.

[28] K. S. Perumalla and R. Fujimoto. *GTW++ - An Object-oriented Interface in C++ to the Georgia Tech Time Warp System.* Georgia Institute of Technology, September 1996.

[29] B. Preiss. Performance of Discrete Event Simulation on a Multiprocessor Using Optimistic and Conservative Synchronization. In *Proc. 1990 International Conference on Parallel Processing*, pages 218 – 222, August 1990.

[30] D. Stiliadis and A. Varma. A Reconfigurable Hardware Approach to Network Simulation. *ACM Transaction on Modeling & Computer Simulation*, 7(1):131 – 156, January 1997.

[31] C. Williamson, B. Unger, and X. Zhonge. Parallel Simulation of ATM Networks: Case Study and Lessons Learned. In *Proceedings of the Second Canadian Conference on Broadband Research (CCBR'98)*, pages 78 – 88, June 1998.