LOOSELY-COORDINATED, DISTRIBUTED, PACKET-LEVEL NETWORK SIMULATION

By

Yu Liu

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

Approved by the
Examining Committee:

Dr. Boleslaw K. Szymanski, Thesis Adviser

Dr. Christopher D. Carothers, Member

Dr. Bülent Yener, Member

Dr. Shivkumar Kalyanaraman, Member

Dr. Kevin Kwiat, Member

Rensselaer Polytechnic Institute
Troy, New York

September 2004
(For Graduation December 2004)

LOOSELY-COORDINATED, DISTRIBUTED, PACKET-LEVEL NETWORK SIMULATION

By

Yu Liu

An Abstract of a Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Major Subject: Computer Science

The original of the complete thesis is on file
in the Rensselaer Polytechnic Institute Library

Examining Committee:

Dr. Boleslaw K. Szymanski, Thesis Adviser

Dr. Christopher D. Carothers, Member

Dr. Bülent Yener, Member

Dr. Shivkumar Kalyanaraman, Member

Dr. Kevin Kwiat, Member

Rensselaer Polytechnic Institute
Troy, New York

September 2004
(For Graduation December 2004)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I am deeply indebted to my thesis adviser Professor Boleslaw Szymanski, without whom this thesis would not have been possible. He has devoted so much time and effort to direct me in this project and guided me through all the related research work. Also thanks my Doctoral Committee members, Professor Christopher Carothers, Professor Shivkumar Kalyanaraman, Professor Bülent Yener and Dr. Kwiat Kevin, who gave me valuable advice and feedbacks on my research.

This work was partially supported by the DARPA Contract F30602- 00-2-0537 and by the grant from the University Research Program of CISCO Systems Inc. I am grateful for these supports which made this project possible.

I would also like to acknowledge the contributions of Anand Sastry, Adnan Saifee, Kiran Madnani, Qiuju Gu and Rashim Gupta to their efforts in the development and implementation of the ideas presented herein.

Above all, thank you my parents and sister for their love and support to my study and research. Last, special thanks for my lovely wife. Her encouragement and intelligent support have been great incentive to my thesis writing.

ABSTRACT

The complexity and dynamics of the Internet is driving the demand for scalable and efficient network simulation. In this thesis, we describe a novel approach to scalability and efficiency of parallel network simulation that partitions the networks into domains and simulation time into intervals. Each domain is simulated independently of and concurrently with the others over the same simulation time interval. At the end of each interval, traffic statistics data, including per flow average packet delays and packet drop rates, are exchanged between domain simulators. The simulators iterate over the same time interval until the exchanged information converges to the value within a prescribed precision before progress to the next time interval. This approach allows the parallelization with infrequent synchronization, and achieves significant simulation speedups.

Large memory size required by simulation software hinders the simulation of large-scale networks. To overcome this problem, our system supports distributed simulations in such a way that each participating simulator possesses only data related to the part of the network it simulates. This solution supports simulations of large-scale networks on machines with modest memory size.

CHAPTER 1
INTRODUCTION

1.1 Motivations

In simulating large scale networks at the packet level, one major difficulty is the enormous computational power needed to execute all events that packets undergo in the network [38]. The needed computational resources can only be provided by the parallel computation involving a large number of processors. Over the last two decades, Parallel Discrete-Event Simulation (PDES) technology has been studied and applied to a variety of network simulation applications. In order to synchronize parallel simulators, conservative and optimistic synchronization techniques were invented and different protocols in both categories were proposed [25]. In conservative parallel simulation, each logical process must be blocked from processing an event until it can make sure that it will not receive any event with a smaller time stamp than the current event in the future. In optimistic parallel simulation, each logical process is allowed to execute events on its event-list without the guarantee that later arrival events have no smaller time stamps, and it uses additional mechanisms to detect out-of-order event execution and to roll the logical process back to a state before the execution of out-of-order events. All the conventional parallel simulations require precise execution and tight synchronization for each individual event. However, the inherent characteristics of network simulations are the small granularity of events and high frequency of events that cross the boundaries of parallel simulations. Tight synchronization on the event-level lowers parallel efficiency of the execution, and therefore do not scale to large number of processors. This difficulty motivated us to invent a new synchronization mechanism for network simulations to achieve better efficiency and scalability, and develop our novel Genesis network simulation system, which integrates a set of parallel simulators loosely coordinated by a coarse granularity synchronization framework [75, 76, 77].

Many parallel simulation systems achieved speedup in execution time, however, they also required that every machine involved had big enough memory to hold the full network, the requirement most easily achievable through the systems with shared memory. In distributed memory parallel systems, this leads to the large memory size required by simulation software on each processor. With the emerging requirements of simulating larger and more complicated networks, the memory size becomes a bottleneck. Simulating a large network requires large memory to construct the network when network configuration and routing information is centralized. In addition, memory size used by simulation software also depends on the intensity of traffic loads (that impact the average size of the future event list), hence the memory used might further increase during the simulation. Memory requirements are also impacted by the design and implementation of different simulation software packages, a good design can achieve better scalability [51]. However, we believe that to simulate truly large networks, the comprehensive, distributed memory approach needs to be developed. This requires that our new simulation approach has to incorporate distributed memory techniques to achieve scalability on memory usage in addition to high execution speed.

Recently, Coffman and Odlyzko reported that Internet data traffic is doubling each year [13]. With the fast growth of Internet data traffic, the demands placed on Internet routing protocols, such as the Border Gateway Protocol (BGP), are tremendous. Correct and efficient performance of these routing protocols is important for keeping the Internet connected. These require the development of simulation tools which can efficiently simulate routing protocols in large scale networks. Because of its efficiency in parallel execution and scalability in both execution and memory usage, we believe that Genesis will be such a powerful tool for BGP simulations in large scale networks. Genesis achieves performance improvement thanks to its novel high-granularity, low-frequency synchronization mechanism. No individual packet is synchronized between two parallel simulations; instead, traffic data are "summarized" on some flow metrics and only these statistic data are exchanged between domains at the end of each time interval. Support for distributed BGP simulation was also incorporated into the framework of Genesis [78], which makes it a powerful tool for analyzing the performance of routing protocols.

1.2 Contributions

Our first contribution was the design and implementation of a scalable distributed network simulation system, Genesis, using coarse granularity synchronization techniques. Coarse granularity synchronization provides a novel approach of simulating large scale networks. Compared with conventional simulation systems which use fine granularity (event-level) synchronization techniques, coarse granularity synchronization reduces the synchronization frequency and the amount of data exchanged among processors. Genesis achieved better execution speed and scalability for large scale network simulations.

Our second contribution was the design and implementation of distributed BGP simulation. This was the first distributed version of detailed BGP simulation in major network simulation systems. BGP simulation in large scale networks requires large memory size which has become a scalability limitation. Distributed BGP simulation creates a new approach of simulating large scale BGP networks using clusters of machines with smaller dedicated memory.

Our third contribution was the development of a general approach to porting existing sequential simulations to distributed platforms. To practice and validate this technique, we have successfully created distributed versions of some widely used network simulation systems, including ns2 and SSFNet.

Our fourth contribution was the development of a general simulation system integration framework, which facilitated the inter-operation among heterogeneous simulation systems.

1.3 Thesis Outline

The rest of this thesis is organized as follows:

Chapter 2 introduces the background of our research endeavor. Chapter 3 gives an overview of the Genesis approach and system. Chapter 4 discusses distributed simulation of UDP and TCP in Genesis with full network topologies. The design of distributed BGP simulation is studied in Chapter 5. Chapter 6 introduces memory distributed simulation in Genesis. Chapter 7 discusses simulation parameters in Genesis and their effects on performance. Chapter 8 gives the conclusions.

CHAPTER 2
BACKGROUND

2.1 Parallel Discrete-Event Simulation

2.1.1 Introduction

Parallel/distributed simulation technology enables a simulation program to be executed on parallel/distributed computer systems which have multiple processors interconnected by a communication network. Parallel discrete-event simulation has been successfully applied in several application areas, for example, aviation control [80], Markov chain simulation [53], architectural simulation [63, 19, 20] and telecommunications [14]. The biggest benefit that can be gained from this technology is that it reduces the execution time of the simulation. For example, by subdividing a large simulation computation into ten different sub-computations and executing them on ten processors, we can reduce the execution time up to a factor of ten (ideally). We can also gain other benefits from this technology, such as integration of heterogeneous simulations and better fault tolerance. But execution time is the biggest concern in most of the large-scale simulations, and it is also one of the major concerns in our research.

However, speedups in execution time in parallel/distributed simulations are limited by the well known synchronization problem: in order to guarantee the correctness of the parallel/distributed simulation, the event execution order should be maintained the same on each of the simulators as the order in which they are executed on an equivalent sequential simulation. For this requirement, once the simulation is divided into concurrent sub-simulations, special synchronization mechanism is needed to control the pace of these concurrent simulations. Different synchronization mechanisms divide parallel/distributed simulation technologies into different categories, as explained below.

2.1.2 Conservative Synchronization Technology

In conservative synchronization technologies, each "Logical Process" (LP) will execute any events strictly in their time-stamp order, no out-of-order events are allowed. Each LP must be sure that no events with smaller time-stamps than the current event will arrive in the future before it can execute the current event. Algorithms were designed to safely advance the simulation time in each LP [12, 4].

The performance of conservative synchronization based simulations is largely determined by a parameter of the simulation model, named lookahead [48, 81]. If a logical process at simulation time T can only schedule new events with time stamp of at least $T+L$, then L is referred to as the lookahead for the logical process. Suppose LP_1 has reached time T and it has a lookahead of L, then any other LPs connected only to LP_1 can safely execute events with time-stamps smaller than $T+L$ without worrying about receiving an out-of-order event from LP_1 in the future. Here is a simple conservative synchronization protocol, where L_i is the lookahead value of LP_i:

```
While   (simulation in progress)
Do
        $T_{min} = MIN(T_i + L_i)$ for all $i$
        $S$ = set of events in the processor with time stamp $\leq T_{min}$
        Process events in $S$
        Barrier synchronization
End
```

Some more advanced conservative synchronization algorithm will take into account the combination of the lookaheads and the relationship among the LPs, in order to advance the local simulation time as much as possible. But the lookahead values is still fundamental to these algorithms.

Intuitively one can expect that the larger the lookahead value, the faster the LPs can advance their simulation execution. But unfortunately, the lookahead values are determined by the design of the simulation model itself, and sometimes also limited by the physical environment to be simulated, so not always big lookahead values are achievable.

Because simulations based on conservative protocols are generally less complex than optimistic ones and easier to be implemented, many simulation applications in the real world fall into this category.

2.1.3 Optimistic Synchronization Technology

Unlike conservative synchronization protocols, optimistic protocols take another approach to maintain the correct event order. The basic idea is that, a LP will advance its event execution without waiting for other LPs. Instead, it will store some additional information for all the "unsafely" executed events. If this LP receives any event with a smaller time-stamp than its current simulation time from other LPs in the future, it will "undo" the execution of all the events with greater time-stamps than that of the received event by using the stored additional information. A special mechanism is used to determined the "safe time point" called Global Virtual Time [31, 32, 23], below which the simulation will not roll back knowing of a simulator can discard the additional information stored before this safe time point. The major concern in this kind of protocols is how to efficiently "undo" event executions and use smaller additional storage.

A parallel simulation system using optimistic synchronization techniques will usually consist of these functionalities:

1. Rolling Back State Variables. Executing an event usually changes the value of some state variables within a logical process. To undo the event, the changed values need to be restored. The major approaches are copy state saving, infrequent state saving [2, 41] and incremental state saving [29, 70]. In any of these methods, the whole or part of the LP's state need to be stored in additional storage. During the roll back, the original value of the state is retrieved from storage or restored by reverse computation. Time Warp programs typically allocate much more memory than sequential simulation [71]. Some rollback-based protocols require no more memory than the corresponding sequential simulation [16, 40], however, performance suffers.

2. Unsending Messages. The execution of an event can also schedule new events and send them to other logical processes. To undo an event, the events sent by this event need to be "un-sent". The Time Warp mechanism [25] introduced

the idea of "anti-message". For each message sent by a Time Warp Logical Process (TWLP), an anti-message, which is simply a copy of the original message with a special flag, is also created. To undo the sending of a previous message, the sending TWLP will send the corresponding anti-message instead. The receiving TWLP will either cancel the original message if it has not been executed, or roll back its state to the point just prior to processing the message to be undone by the anti-message.

3. Fossil Collections. From the discussion above, to support roll backs, additional copies of state variables and messages need to be stored. These additional storage need to be reclaimed regularly during the simulation, to prevent it from running out of memory. The Global Virtual Time (GVT) is computed in Time Warp system. GVT is the lower bound of the time stamp of any future rollback. Once the GVT is computed [26], the copies of state variables and messages earlier than the GVT are discarded and memory is reclaimed in a procedure called fossil collection. In order to correctly compute the GVT, some global control algorithms were designed to synchronize the logical processes during the computation.

In optimistic synchronization protocols, events are periodically synchronized only "loosely". As the result, additional storage is required to support rollbacks, so the overall order of the events is still strictly maintained. In addition, global control is required to synchronize processes during the GVT computation.

The additional storage is required to store the already executed messages as well as state variables, which means that the size of the additional storage is not only related to the size of the program (size of states), but also the event processing history.

2.2 Related Research

2.2.1 Network Simulator ns2 and PDNS

ns2 [57] represents the state-of-the-art in networking simulation packages. It supports a variety of models, such as UDP, TCP/IP and wireless. The critical feature of ns2 is the ease with which researchers can model networks. The standard ns2 implementation used sequential simulation technology. It did not support parallel network simulation. The simulation in ns2 is also centralized in terms of memory usage as well as execution.

Despite the advances made by ns2 in terms of flexibility and ease of use, it is not adequate for simulating large-scale networks. Consider a simple network with 512 source nodes connected by a 1Gb/s duplex link to 512 sink nodes. The traffic in the network consists of 1000 byte UDP packets and will fully utilize the duplex link configured for drop-tail queuing. To simulate this network on a modern 1GHZ, 250MB RAM PC, ns2 will allocate almost 180MB of RAM and will processes events at a rate of about 1000 per second. Hence, 1 second of the simulated traffic will take nearly 1 hour of simulation time and will use almost entire memory of this platform. For large-scale networks, several orders of magnitude greater memory and performance are needed. Simulation results also showed that ns2 failed to simulate such a network when the number of connections increased to more than 9000, thanks to its centralized computation and memory usage [51]. Parallel and distributed network simulation technology is needed to achieve efficiency and scalability, especially in large scale network simulations.

In addition, the current version of ns2 did not support the simulation of BGP4, which made it difficult to study BGP performance in simulation.

PDNS is an extension of ns2, which is a parallel/distributed version of ns developed by Georgia Tech [61]. PDNS distributes the memory as well as the execution among parallel simulators: each parallel simulator only constructs and simulates part of the full network. The parallelization of ns2 in PDNS is achieved by utilizing a library for implementing parallel/distributed simulations, known as RTIKIT [22]. RTIKIT utilizes a conservative synchronization mechanism to support the interoperability among a group of sequential or parallel simulators. Because a conservative protocol is used, PDNS falls prey to the same problems as conservative parallel simulators.

Besides the extension of conservative synchronization parallel simulation technology, PDNS extends the tcl [59, 79] network model in ns2 by introducing the definitions of remote links, remote agents, remote route, et cetera, to distributively define and construct network. Global routing issues arise when the network is decomposed into distributed simulators. When using PDNS, since only a portion of the global topology is known to each simulator, there are times when choosing the correct route between nodes defined on different simulators is problematic, thus remote routes are required to be defined by users in network model definition file. Another case is the so called “pass-through” routes: sometimes a packet is received by a simulator from a remote one, but the ultimate destination for that packet is still not local to the simulator receiving the packet. In this case, the packet is to be passed through to another simulator. Such routing is called pass-through routing, and also need to be specified by user in network model definition file in PDNS.

The method used in PDNS to solve the global routing problem in distributed simulation is problematic: users are required to pre-define global routing for remote traffic in the network model definition phase. However, when global routing is difficult to predict, it will be very difficult to pre-define it in network script files. More importantly, such an approach makes impossible the simulation of global routing protocols, for example, BGP, which will decide the global routing dynamically during the simulation.

2.2.2 SSFNet and DaSSFNet

SSFNet [67, 58] is a collection of Java SSF-based components for network modeling and simulations. It can simulate Internet protocols and networks from IP packet level up in the protocol stack. Parallel simulation is implemented not at the SSFNet layer, but in the underlying simulation engine. The implementation of the parallelization in the simulation engine is transparent to the SSFNet layer. Such a design simplified the SSFNet layer from the details of the parallel implementation.

The Java based implementation of SSF simulation engine supports parallel simulation using conservative synchronization techniques. Interface for parallelization between SSFNet and the underlying engine is provided as the concept of “alignment”. Network components are defined as “entities” in the simulation, and the simulation entities aligned to different alignment groups could be simulated on different processors. Entities exchange simulation events through “channels”, which are connections between two different entities. Channels connecting two entities which belong to two different alignment groups transmit events between two parallel simulators. This kind of border channels require non-zero lookahead to guarantee the conservative synchronization performance.

Simulation results showed that Java based SSFNet runs slower than ns2 for smaller size of networks, but it has better scalability in terms of memory and computation limitations for large scale network simulations, thanks to its parallel simulation approach as well as to the garbage collection mechanism provided by Java. In a simulation for a network of a dumbbell topology and with large number of connections, ns2 failed to simulate more than 9000 connections, while Java based SSFNet successfully simulated the largest number of connections provided, which was 12000 sessions [51].

Because in SSFNet, the parallel simulation is implemented in the simulation engine, the network model is transparent to the synchronization mechanism. As a result, without any awareness of the semantics of the simulation events, the simulation engine in SSFNet must perform tight event-level synchronization on any individual event. On the other hand, Java based SSFNet does not provide memory distribution, which requires that every involved processor has the access to the full storage of the whole network. This design is more suitable for shared-memory multiprocessor machines and can not scale well for very large network simulations.

DaSSFNet [18] is a C++ implementation of SSFNet, which was designed by Dartmouth College. The simulation engine for DaSSFNet is DaSSF, which is a parallel simulator using also the conservative synchronization approach. DaSSFNet/DaSSF has similar design of the network model and the interfaces between the engine and the network layer as SSFNet/SSF, but it optimizes its C++ implementation to achieve better execution efficiency. One of the biggest improvements in DaSSFNet is that it supports simulations on distributed memory, which allows it to utilize not only shared-memory multiprocessors, but also distributed memory clusters of workstations [43].

DaSSFNet also supports distributed construction of the network in distributed memory. In order to refer to the objects constructed in different memory spaces, a global string name is defined for each object in DML file which defines the network model. In addition, in order to transmit events through the border channels connecting two entities in different memory spaces, "proxies" for these channels are introduced to collect this kind of remote events, and pass them to the proper remote entities. Each event to be transmitted to a remote simulator must be converted into a byte stream. Because DaSSFNet also perform tight synchronization at event-level, the distributed-memory implementation involved only two major extensions: the way to identify the entities on both ends of a remote channel, and the way to pass each individual event in such a remote channel to the other end. As we will see later in our Genesis research, memory distribution in a coarse granularity synchronization system without event-level synchronizations will be more complicated, where events (packets) are handled differently based on their semantics in the network.

As reported by David Nicol from Dartmouth College, DaSSFNet runs nearly as fast as ns2, and uses less memory than ns2 or Java based SSFNet. The performance of DaSSFNet is more stable than other simulation software when the network size scales up thanks to its parallel simulation approach and distributed memory usage [51].

DaSSFNet is one of the closest research works to our Genesis in terms of addressing the memory usage limitation problem. This work was done within the framework of conservative synchronization parallel simulation approach with the effort to improve its performance. In contrast, Genesis is designed to overcome the efficiency limitation of parallel network simulation by avoiding tight synchronization on the event-level, and it utilizes a novel optimistic coarse granularity synchronization mechanism.

One major setback of DaSSFNet is that it does not have any routing protocol implemented – protocols like BGP and OSPF. Static forwarding tables need to be defined for routers in network script files. Like ns2, this makes it impossible to study the dynamics and performance of routing protocols in simulation.

2.2.3 High Level Architecture (HLA)

HLA [15] is, first, a conceptual framework because it defines a collection of related concepts for organization of simulation systems. It defines a standard for connecting several computer-based simulation systems so that they can run together and exchange information. This allows the integration of existing simulation systems with new systems, and the reuse of existing systems for new purposes. Secondly, HLA is also an application framework because it defines interaction interfaces between the federates of the system, and provides implementations of the "Runtime Infrastructure" (RTI). The basic concepts in HLA include:

1. *Federation.* The *Federation* is a collection of *Federates* with a common *Federation Object Model.* The *Federation* constitutes of a number of sub-systems that together with a *Runtime Infrastructure*, *RTI*, form a distributed simulation system.

2. *Federate.* A *Federate* is a member of a *Federation.* It is a simulation component participating in the *Federation*, communicating and cooperating with other *Federates.* A *Federate* can be a sub-system of a simulator or a whole simulator in a multi-simulator *Federation.*

3. *Runtime Infrastructure.* The *Runtime Infrastructure (RTI)* is, in effect, a distribute operating system for the *Federation. RTI* provides a set of services that support the simulation in carrying out these federate-to-federate interactions and *Federation* management support functions.

Intuitively, HLA can be viewed as a parallel/distributed simulation system. It distributes the simulation execution as well as memory usage among participating *Federates. RTI* provides several services to integrate these *Federates*, among which the "Time Management Service" performs the parallel synchronization function. In HLA, *Federates* must explicitly request logical time advances, and the advance does not occur until the *RTI* issues a grant. *RTI* will only grant an advance to logical time T when it can guarantee all time-stamp ordered messages with time-stamp less than T (or in some cases less than or equal to T) have been delivered to the

Federate. This guarantee enables the *Federate* to simulate the behavior of the entity it represents up to logical time T without concern for receiving new events with time-stamp less than T. HLA uses a conservative synchronization technology to coordinate the simulations in *Federates*.

The HLA provides several services to support the inclusion of optimistic *Federates*, e.g., *Federates* synchronized using the Time Warp mechanism. The Lowest Bound Time Stamp(LBTS) value computed by the *RTI* enables the *Federate* to compute Global Virtual Time (GVT). A service for retracting previously scheduled events is provided, thereby enabling the implementation of "anti-messages". A service is provided to allow optimistic *Federates* to flush the time-stamp ordered message queue, enabling the optimistic processing of events. An important property of the time management services is that they ensure that optimistic events (events that may later be canceled) are not delivered to *Federates* unless they explicitly request them via the flush queue primitive, thereby enabling conservative *Federates* to operate in the same *Federation* with optimistic *Federates*. It is not necessary for these conservative *Federates* to implement rollback and message cancellation mechanisms.

Similar to Genesis, HLA is also a framework which supported integration of simulation systems. However, there are some important difference between HLA and Genesis. Because HLA was designed to provide a general framework to integrate both new and existing systems which might use either conservative or optimistic synchronization technology, a conservative based synchronization mechanism became the only choice for RTI. In contrast, the purpose of our research in Genesis was to achieve high efficiency and scalability in simulation and provide the integration ability for Genesis-complied simulators. Integration of existing systems was not the major purpose of Genesis. This allowed us to utilize novel optimistic synchronization technology. In addition, Genesis used a technique of application level checkpointing mechanism in its coarse granularity synchronization. This technique can be used to convert existing sequential simulation systems into Genesis-complied simulation sub-systems, thus also provided the ability to integrate existing systems.

2.2.4 GTW and ROSS

GTW (Georgia Tech Time Warp) [17] and ROSS (Rensselaer's Optimistic Simulation System) [9, 10] are parallel simulation systems using optimistic synchronization technologies. One concern in optimistic protocols is how to efficiently cancel event execution for out-of-order events and use smaller additional storage. One approach is "Rolling Back State Variables", which stores all the state variables within a logical process, or stores them in an incremental way, and then restores their original values in a roll back. Another approach which can save the addition storage required by state saving is "Reverse Computation" [6, 7, 8], which is used in ROSS, where the original values are computed from their current values. To cancel the sending of an event, GTW and ROSS use "anti-messages", which is simply a copy of the original message with a special flag. An LP that receives an anti-message will eliminate the original message received earlier. If the original message has already been processed, the effects of processing this message will also be rolled back. Fossil collection is a mechanism to reclaim additional storage used by optimistic synchronization. The Global Virtual Time (GVT) is computed in Time Warp system. GVT is the lower bound of the time stamp of any future rollback. Once the GVT is computed, the copies of state variables and messages created earlier than the GVT are discarded and memory is reclaimed in the procedure of fossil collection.

GTW is a parallel discrete event simulation system based on a Time Warp mechanism. It was designed to support efficient execution of small granularity discrete-event simulation applications, and targeted at cache-coherent, shared-memory multiprocessor systems. As a result, it used a simplified GVT computation algorithm, in which the GVT execution is asynchronous and is interleaved with normal processing.

ROSS is also a system designed for shared-memory multiprocessor systems and uses the same GVT computation algorithm as GTW. It is a modular, C-based implementation of Time Warp protocols. It introduced some important improvements into previous implementation of Time Warp systems to achieve high performance and low memory requirement. It is capable of extreme performance: on a quad processor PC server ROSS is capable of processing over 1,250,000 events per second for a wireless communications model. Additionally, ROSS only requires a small constant amount of memory buffers greater than the amount needed by the sequential simulation for a constant number of processors. The improvements in ROSS were achieved by the integration of several technologies including its pointer-based, modular implementation framework, its use of reverse computation technique and its use of Kernel Processes.

Memory consumption, event rollbacks and GVT computation overheads are often cited as reasons for commercial network simulation packages using the conservative synchronization techniques. To develop more efficient GVT algorithms, reduce synchronization overheads and minimize optimistic memory consumption was one approach taken by optimistic synchronization simulation research. To develop a novel, more efficient synchronization technology for network simulation and distribute the memory usage was another approach taken in the research work addressed in this thesis.

2.2.5 Fluid Model Network Simulation

The traditional packet-level approach to network simulation is to simulate the arrival, queuing, processing and departure of each individual packet at the various queues along a packet's path through the network. However, as the number of network nodes becomes large, this approach becomes computationally infeasible, or at the best, inefficient. Thus, while packet level simulation may be easy to implement and can yield accurate results for small networks, more scalable simulation methodologies and techniques are needed to simulate truly large scale networks.

To reduce the complexity of network simulation in packet level, some approaches to using a "fluid" technique to approximate the flow between network partitions were invented, such as Kesidis and Walrand [36], itDecisionGuru [30] and the Rensselaer Scalable On-line Simulation project [72]. Fluid simulation uses a fluid flow rather than a packet-by-packet flow to represent a traffic source. In a fluid simulation, a source's fluid flow rate may change as the packet generation rate in the system being modeled changes. Small time-scale variations in the packet arrival stream are abstracted out of the source model by having the fluid's rate remain constant between these changes in the fluid rate. As the fluid flows through the network, a fluid simulator need only to keep track of the rate changes that occur as a result of queuing, multiplexing, and processing of the fluids at the various queues; in contrast, an equivalent packet-level simulation would need to keep track of a potentially enormous number of packets in the network. When a packet-level source is modeled as a fluid source, the fluid's rate (and rate changes) is simulated and tracked exactly at each point in the network. Thus, in fluid simulation, network traffic is modeled in terms of a continuous fluid flow, rather than discrete packet instances.

In fluid simulations, the traffic source rates and its changes are used to calculate queue length, average waiting time in queue, et cetera. Because no individual packet is traced, the simulation is more efficient, and scales better than traditional packet-level simulation when network size grows up. Such a high level of abstraction might achieve high processing efficiency when simulating network traffic [42].

One drawback of a fluid model is the accuracy of the measurements of interest that is compromised because of the abstraction. In addition, performance measurements cannot be obtained in fluid simulation as conveniently as in packet-level simulations.

2.3 Chapter Summary

In this chapter, we provided an overview of Parallel Discrete-Event Simulation (PDES). Synchronization technique is one of the biggest challenge in parallel discrete event simulation. These techniques have been classified into two major categories: conservative synchronization and optimistic synchronization. Either of them has some advantages over the other. More comprehensive surveys of various synchronization techniques are provided in [24, 55, 52, 21, 47]. Large scale network simulation is particularly challenging in terms of scalability, because of the enormous computational power needed to execute all events that packets undergo in the network. We provided an overview of some related research efforts towards building a high performance, scalable network simulation system. Some of them used sequential simulation techniques while the others took the parallel/distributed simulation approach.

The techniques to improve the parallel synchronization performance were different, however, all the packet-level parallel network simulation systems shared one common property in: they all performed tight packet-level synchronization to guarantee the correct time-stamp order in each processor. To improve simulation performance within this tight packet-level synchronization framework was the approach taken by most of current research efforts. In the rest of this thesis, we will introduce a novel approach to improve simulation performance, the coarse granularity synchronization framework.

CHAPTER 3
GENESIS: GENERAL NETWORK SIMULATION INTEGRATION SYSTEM

3.1 A Novel View of Network Simulation

The inherent characteristics of large-scale network simulation is that the number of events (e.g., packets) is huge and the event rate is high, while the granularity of an event is usually small. In the previous chapter, we have shown that synchronization for parallel/distributed network simulation is a big challenge, and researchers have taken different approaches to improve synchronization performance.

This high synchronization overhead comes from a "general rule" for parallel and distributed simulations: each event that is created on one processor and needs to be executed on the other introduces synchronization overhead. The processors involved in such an event need to be synchronized for this event and this delays their execution. This general rule limits the improvement of synchronization performance for network simulation because of the huge number and high frequency of events.

Can we break this "general rule"? Our efforts to find the answer for this question had led us to the research work addressed in this thesis.

In the traditional view of network simulation, we consider a group of parallel or distributed simulation sub-systems as one simulation system which is required to produce exactly the same simulation result as a sequential simulation would do. Under this consideration, the general rule is an indispensable requirement to guarantee the correct result. The answer to our question seems to be negative. However, is it always a requirement to synchronize each individual packet to achieve the purpose of a network simulation? The answer is no. In many network simulations, we do not care what have happened to individual network packets. Instead, we are more interested in some "metrics", for example, traffic throughput, end-to-end packet delay, packet lost rate, et cetera. Thus, from a view at a higher level, we are running simulations to achieve statistics data for the "metrics" we are interested in. A simulation system only needs to produce these data accurately, or with approximations within a satisfactory range, instead of guarantee the correct behavior of each individual packet. This gave us the possibility to simplify a network simulation.

With this novel view of network simulation, we consider a distributed simulation system as a loosely coupled distributed computing system. Each distributed domain simulator runs separately doing local computation (simulating the domain assigned to it) within a period of time, with all the information of the network it has at that time, to produce local results as accurately as possible. Periodically, these distributed simulator exchanges computation results and updates network information among them. Each simulator uses these "fresh" information to update its own computation and information base, to produce more accurate results during the next iteration. In this way, we don't need to synchronize and exchange data among simulators at event-level (packet-level). The synchronization for these loosely coupled sub-systems can be much infrequent and the overhead will be reduced significantly.

In this chapter, we will provide the overview of Genesis, which had materialized this view into a real system.

3.2 System Architecture

In Genesis, a large network is decomposed into parts and each part is simulated independently and simultaneously with the others. Each part represents a subnet or a sub-domain of the entire network. These parts are connected to each other through edges that represent communication links existing in the simulated network. In addition, we partition the total simulation time into separate simulation time intervals selected in such a way that the traffic characteristics change slowly during most of the time intervals.

Each domain is simulated by a separate simulator which has a full description of the flows whose sources are within the domain. This simulator also needs to simulate and estimate flows whose sources are external to the domain but will be routed to or through the domain. In addition to the nodes that belong to the domain by the user designation, we also create *domain closure* that includes all the sources of flows that reach or pass through this domain. Since these are copies of nodes active in other domains, we call them *proxy sources.* Each proxy source uses the flow definition from the simulation configuration file.

The flow delay and the packet drop rate experienced by the flows outside the domain are simulated by the random delay and probabilistic loss applied to each packet traversing in-link proxy. These values are generated according to the average packet delay as well as observed packet loss frequency communicated to the simulator by its peers at the end of simulation of each time interval. Each simulator collects this data for all of its own out-link proxies when packets reach the destination proxy.

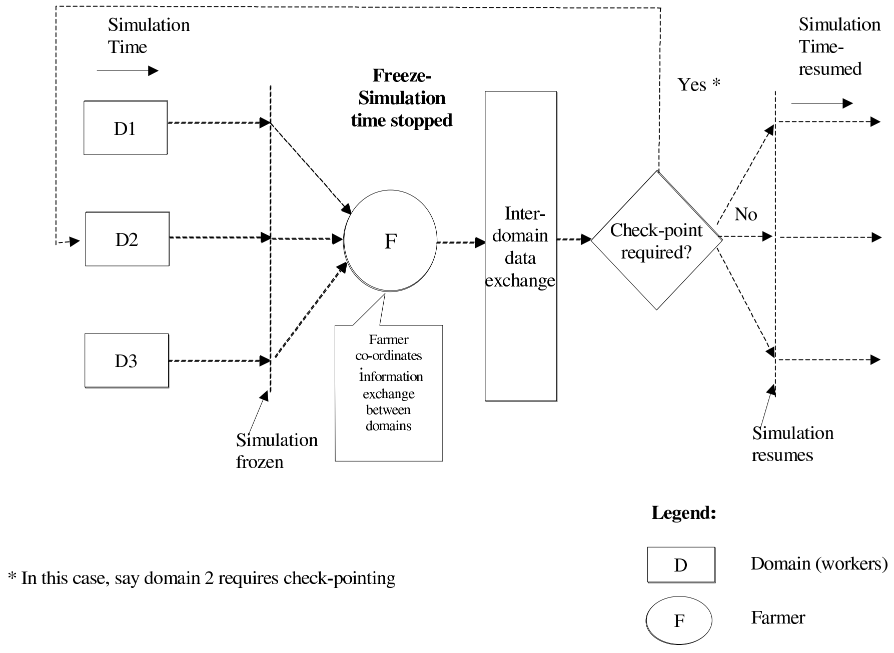
Simulation Time
D1
D2
D3
Freeze-Simulation time stopped
F
Farmer co-ordinates information exchange between domains
Simulation frozen
Inter-domain data exchange
Check-point required?
Yes *
No
Simulation Time-resumed
Simulation resumes
* In this case, say domain 2 requires check-pointing
Legend:
D
Domain (workers)
F
Farmer

Figure 3.1: Progress of the Simulation Execution

A Farmer-Worker system is designed for data exchange among these domain simulators. Each domain simulator runs as a worker, and one stand-alone server runs as a farmer to synchronize domain simulators. Every domain simulator stops its simulation at pre-defined checkpoints, and exchanges data with all the other domain simulators. During a checkpoint, each domain simulator also checks its convergence condition by analyzing the received data, based on some pre-defined metrics (end-to-end packet delay, packet loss rate, etc.) and parameters (e.g. precision threshold). The farmer collects convergence information from all domain simulators and makes global convergence decisions. If some convergence condition is not satisfied, the farmer will inform some or all domain simulators to roll back and re-iterate. Those simulators which need to roll back will go back to the last checkpoint and re-simulate the last time interval, however, utilizing the data received during the current checkpoint. When all the domain simulators converge, a global convergence is reached and the farmer will inform all the domain simulators to go on to the next time interval. The system framework is shown in Figure 3.1, and the details are explained below.

In the initial (zero) iteration of the simulation process, each part assumes on its external in-links either no traffic, if this is the first simulated interval, or the traffic defined by the packet delays and drop rate defined in the previous simulation time interval for external domains. Then, each part simulates its internal traffic, and computes the resulting outflow of packets through its out-links.

In the subsequent $k > 0$ iteration, the in-flows into each part from the other parts will be generated based on the out-flows measured by each part in the iteration $k - 1$. Once the in-flows to each part in iteration k are close enough to their counterparts in the iteration $k - 1$, the iteration stops and the simulation either progresses to the next simulation time interval or completes execution and produces the final results.

Consider a flow from an external source S to the internal destination T, passing through a sequence of external routers $r_1, \ldots r_n$ and internal routers $r_{n+1}, \ldots r_k$. The source of the flow is represented by the sequence of pairs $(t_1, p_1), \ldots (t_m, p_m)$, where t_i denotes the time of departure of packet i and p_i denotes its size. At router i, a packet j is either dropped, or passes with the delay $d_{i,j}$. For uniformity, dropping can be represented as as delay T greater than the total simulation time. Hence, to replicate a flow with the proxy source S' sending packets to router r_{n+1}, packet j produced by S' at time t_j needs to be delayed by time $D_j = \sum_{i=1}^{n} d_{i,j}$. A delay at each router is the sum of constant processing, transmission and propagation delays and a variable queuing delay. If the total delay over all external routers is relatively constant in the selected time interval, a random delay with proper average and variance approximates D_j well. Thanks to the aggregated effect of many flows on queue sizes, this delay changes slower than the traffic itself, making such a model precise enough for our applications.

3.3 Coarse Granularity Synchronization in Genesis

Genesis uses a coarse granularity synchronization mechanism, described above, to simulate network traffics, e.g., TCP or UDP flows. This approach avoids frequent synchronization of parallel simulators. Parallel domain simulators are running independently. Each of them uses data that it received from others to represent the external network outside of its own domain. By periodically exchanging data with other domain simulators and reiterating over the same simulation time interval to achieve a global convergence, the simulation of the whole network approximates the sequential simulation of the same network with controllable precision. This is explained more formally as follows.

Consider a network $\Gamma = (N, L)$, where N is a set of nodes and L (a subset of Cartesian product $N \times N$), is a set of unidirectional links connecting them (a bidirectional link is simply represented as a pair of unidirectional links). Let $(N_1, ..., N_q)$ be a disjoint partitioning of the nodes, each partition modeled by a simulator. For each subset N_i, we can define a set of external out-links as $O_i = L \cup (N_i \times (N - N_i))$, in-links as $I_i = L \cup ((N - N_i) \times N_i)$, and local links as $L_i = L \cup (N_i \times N_i)$.

The purpose of a simulator S_i, that models partition N_i of the network, is to characterize traffic on the links in its partition in terms of a few parameters changing slowly compared to the simulation time interval. In the implementation presented in this thesis, we characterize each traffic as an aggregation of the flows, and each flow is represented by the activity of its source and the packet delays and losses on the path from its source to the boundary of that part. Since the dynamics of the source can be faithfully represented by the copy of the source replicated to the boundary, the traffic is characterized by the packet delays and losses on the relevant paths. Thanks to queuing at the routers and the aggregated effect of many flows on the size of the queues, the path delays and packet drop rates change more slowly than the traffic itself.

Based on such characterization, the simulator can find the overall characterization of the traffic through the nodes of its subnet. Let $\xi_k(M)$ be a vector of traffic characterization of the links in set M in k-th iteration. Each simulator can be thought of as defining a pair of functions:

$$\xi_k(O_i) = f_i(\xi_{k-1}(I_i)),\ \xi_k(L_i) = g_i(\xi_{k-1}(I_i))$$

(or, symmetrically, $\xi_k(I_i),\ \xi_k(L_i)$ can be defined in terms of $\xi_{k-1}(O_i)$).

Each simulator can then be run independently of others, using the measured or predicted values of $\xi_k(I_i)$ to compute its traffic. However, when the simulators are linked together, then of course $\bigcup_{i=1}^{q} \xi_k(I_i) = \bigcup_{i=1}^{q} \xi_k(O_i) = \bigcup_{i=1}^{q} f_i(\xi_{k-1}(I_i))$, so the global traffic characterization and its flow are defined by the fixed point solution of the equation:

$$\bigcup_{i=1}^{q} \xi_k(I_i) = F(\bigcup_{i=1}^{q}(\xi_{k-1}(I_i)), \tag{3.1}$$

where $F(\bigcup_{i=1}^{q}(\xi_{k-1}(I_i))$ is defined as $\bigcup_{i=1}^{q} f_i(\xi_{k-1}(I_i))$. The solution can be found iteratively starting with some initial vector $\xi_0(I_i)$, which can be found by measuring the current traffic in the network.

We believe that communication networks simulated that way will converge thanks to monotonicity of the path delay and packet drop probabilities as the function of the traffic intensity (congestion). For example, if in an iteration k, a part N_i of the network receives more packets than the fixed point solution would deliver, then this part will produce fewer packets than the fixed point solution would. These packets will create inflows in the iteration $k + 1$. Clearly then, the fixed point solution will deliver the number of packets that is bounded from above and below by the numbers of packets generated in two subsequent iterations I_k and I_{k+1}. Hence, in general, iterations will produce alternately too few and too many packets in the inflows providing the bounds for the number of packets in the fixed point solution. By selecting the middle of each pair of bounds, the number of steps needed to convergence can be limited to the order of logarithm of the needed accuracy, so convergence is quite fast. In the measurements reported later in this thesis, the convergence for UDP traffic was achieved in 2 to 3 iterations, for TCP or mixed UDP/TCP traffic in 5-10 iterations, and for BGP/TCP/UDP traffic it was about twice the number of Autonomous Systems simulated.

It should be noted that the similar method has been used for implementation of the flow of imports-exports between countries in the Link project [37] led by the economics Noble Laureate, Lawrence Klein. The implementation [66] included distributed network of processors located in each simulated country and it used global convergence criteria for termination [74].

One issue of great importance for efficiency of the described method is frequency of synchronization between simulators of parts of the decomposed network. Shorter synchronization time limits parallelism but decreases also the number of iterations necessary for convergence to the solution because changes to the path delays are smaller. Variance of the path delay of each flow can be used to adaptively define the time of the synchronization for the subsequent iteration or the simulation step.

It is easy to observe that the execution time of a network simulation grows faster than linearly with the size of the network. Theoretical analysis supports this observation because for the network size of order $O(n)$, the sequential simulation time include terms which are of order:

- $O(n * \log(n))$, that correspond to processing events in the order of their simulation time in the event queue;
- $O(n(log(n))^2)$ to $O(n^2)$, depending on the model of the network growth, that result from number and complexity of events that packets undergo flowing from source to destination. The average length of a path traversed by each packet, the number of active flow sources, the number of flows generated by each source and even the number of packets in each flow may grow at the rate of $O(log(n))$ to $O(n^\alpha)$, where $0.5 \leq \alpha \leq 1$, as the function of n, the number of nodes in the network. They together create the super-linear growth in the number of the events processed by the simulation.

Some of our measurements [82] indicate that the dominant term is of order $O(n^2)$ even for small networks. Using the least squared method to fit the measurements of execution time for the different network sizes, we got the following approximate formula for star-interconnected networks:

$$T(n) = 3.49 + 0.8174 \times n + 0.0046 \times n^2 \tag{3.2}$$

where T is the execution time of the simulation, and n is the number of nodes in the simulation. From the above, we can conclude that the execution time of a network simulation is a super-linear function of the network size. Therefore, it is possible to speed up the network simulation more than linearly by splitting a large simulation into smaller pieces and parallelizing the execution of these pieces.

As we demonstrate later in the measurement section, a network decomposed into 16 parts will require less than 1/16 of the time of the entire sequential network simulation, despite the overhead introduced by external network traffic sources added to each part and synchronization and exchange of data between parts. Hence, with modest number of iterations the total execution time can be cut an order of magnitude or more. Our experiment results showed that this approach achieved significant speed-up for TCP or UDP traffic simulations.

Another advantage of the proposed method is that it is independent of any specific simulator technique employed to run simulators of the parts of the decomposed network. Rather, it is a scheme for efficient parallelization based on convergence to the fixed point solution of inter-part traffic. The convergence is measured by a set of parameters characterizing the traffic rather than individual packets. Our primary application is network management based on on-line network monitoring and on-line simulation [82]. The presented method fits very well to such application as it predicts changes in the network performance caused by tuning of the network parameters. Hence, the fixed point solution found by our method is with high probability the point into which the real network will evolve. However, there are open questions such as under what conditions the fixed point solution is unique, or when the solution found by the fixed-point method coincide with the operating point of the real network.

The method can be used in all applications in which the speed of the simulation is of essence, such as: on-line network simulation, ad-hoc network design, emergency network planning, large network simulation, network protocol verification under extreme conditions (large flows).

3.4 Inter-Domain Routing Simulation in Genesis

Genesis achieved performance improvement thanks to coarse granularity synchronization mechanism. Since in many network simulation scenarios, the real data of the traffics packets are not important to the simulation result, no individual packet is synchronized between two parallel simulations in UDP and TCP traffic simulations. Instead, packets are "summarized" on some metrics (delay, drop rate, etc.). Only these data are exchanged between domains at the end of each time interval. This approach was designed to simulate TCP and UDP data traffics, but could not be used to simulate some other flows, for example, data flows providing information for routing protocols. This is because the traffic of a routing protocol cannot be summarized; instead, different content and timing of each routing packet might change the network status. Particularly, our desire to simulate BGP protocol required us to develop additional synchronization mechanism in Genesis.

We developed an event-level synchronization mechanism which can work within the framework of Genesis and support the simulation of BGP protocol. To simulate a network running BGP protocol for inter-AS (Autonomous System) routing, with background TCP or UDP traffics, we decompose the network along the boundaries of AS domains. Each parallel simulator simulates one AS domain, and loosely cooperates with other simulators. When there are BGP update messages that need to be delivered to neighbor AS domains, the new synchronization mechanism in Genesis guarantees that these messages will be delivered in the correct time-stamp order.

3.5 Simulation Systems Integration

3.5.1 Interoperability Between ns2 and SSFNet

Java-based SSFNet and C++/TCL-based ns2 use different network models and different simulation frameworks. The details of the implementation of traffic packets and other network entities are different in these two systems. Thanks to the coarse granularity synchronization framework in Genesis, only traffic statistics data summarized on some metrics are exchanged among domain simulators, while the implementation details of the actual network traffic in one domain can be viewed as a black box to the other. This facilitates the design of a general integration framework.

In Genesis, we design the general format of the traffic statistics data message being exchanged in the framework, and the general conventions for a domain simulator to identify a network entity (e.g., nodes identified by a global node id). Then, the rest of the work is the implementation of conversion between native data format and the general message format for both SSFNet and ns2. Because of this general inter-operation interface, a SSFNet domain simulator can work with either SSFNet or ns2 domain simulators, in exactly the same way. Another advantage of this approach is its extensibility: any domain simulators complies with this general interface can be easily plugged into Genesis.

3.5.2 Interoperability Between SSFNet and GloMoSim

Based on the design of interoperability between ns2 and SSFNet, we adopted a similar approach (described below) to enable interoperability between SSFNet and GloMoSim. Our main objective is to create a scenario where we have mixed-mode traffic between a wired network (modeled using SSFNet) and a wireless network (modeled using GloMoSim).

The network configuration includes wired domains simulated by SSFNet and wireless domains simulated by GloMoSim. The SSFNet part of the network will view the wireless GloMoSim domains as a single node proxy network, which is the source and sink for all traffic originating and destined respectively to the latter. Similarly, for GloMoSim, the SSFNet domains are represented by a single node proxy network as well. At each checkpoint interval, the information about inter-domain traffic statistics data is exchanged between SSFNet and GloMoSim simulations. The receiving simulation uses this information to adjust the single node proxy network and the links connecting to it, to better represent its cooperating simulation. And based on the received information and local conditions in the domain, a decision whether to roll back or not is made by each of the domains.

3.6 Memory Distribution

Simulations of large-scale networks require large memory size. This requirement can become a bottleneck of scalability when the size or the complexity of the network increases. For example, ns2 uses centralized memory during simulation, which makes it susceptible to the memory size limitation. The scalability of different network simulators was studied in [51]. This paper reports that in a simulation of a network of a dumbbell topology with large number of connections, ns2 failed to simulate more than 10000 connections. The failure was caused by ns2's attempt to use virtual memory when swapping was turned off. This particular problem can be solved by using machines with larger dedicated or shared memory. Yet, we believe that the only permanent solution to the simulation memory bottleneck is to develop the distributed memory approach.

In a typical parallel network simulation using non-distributed memory, each of the parallel simulators has to construct the full network and to store all dynamic information (e.g., routing information) for the whole network during the simulation. To avoid such replication of memory, we developed an approach that completely distributes network information. Thanks to this solution, Genesis is able to simulate large networks using a cluster of computers with smaller dedicated memory (compared to the memory size required by shared memory-based SSFNet simulating the same network).

3.7 Chapter Summary

In this chapter, we provided an overview of our distributed simulation system, Genesis. Genesis took the novel coarse granularity synchronization approach to improve simulation performance, and distributed the memory usage to reduce the memory requirement on each participating simulator. In addition, Genesis also supported distributed BGP simulation with full memory distribution. This will facilitate the simulation of large scale BGP networks on cluster of machines with smaller dedicate memory.

CHAPTER 4
DISTRIBUTED SIMULATION WITH FULL NETWORK TOPOLOGY

4.1 Introduction

In this chapter, we will discuss the design of Genesis distributed simulation with full network topology stored in each domain simulator.

4.1.1 Distributed Simulation Approaches

In parallel/distributed network simulation, the simulation system utilizes multiple processors or machines to simulate one network concurrently to improve simulation performance. There are two different approaches of the distribution:

- Approach 1: Computation distribution only. In this approach, each processor or machine has the same full copy of the network topology being simulated, or accesses the same network topology in shared memory (e.g. in parallel simulation). Thus, the computation of the simulation is distributed among processors, while one or multiple copy of the full network topology need to be stored as a whole somewhere.

- Approach 2: Computation and memory distribution. In this approach, the full network topology is divided into partitions and only one partition of the network is constructed for each machine participating in the simulation. Thus, not only the computation of the simulation but also the memory used for constructing the network is distributed.

Generally speaking, the advantage of approach 1 is that it is easier to implement because each participating simulator has information about the full network, which facilitates the cooperation among simulators. These systems can be comparatively simple thus the overheads introduced by parallel simulation is relatively small as well. All of the parallel simulation systems mentioned earlier belong to this category. However, its disadvantage is also apparent. Because the system needs to store the full network, the memory size required by the simulation limits its scalability, especially in large-scale network simulations. Super computers with large memory usually will be required in those cases.

Approach 2 solved the memory limit problem by distributed the memory usage among distributed simulation machines. It is attractive because it provides the ability to use clusters of machines with small memory size to simulate large networks. DaSSFNet mentioned earlier belongs to this category. However it was not a complete solution because it used pre-configured forwarding tables instead of supporting real routing simulation.

In addition, approach 2 also provides potentials for a broader range of distributed applications. Because it does not require storing of the full network, each distributed simulator requires only partial information about the network. Each simulator manages its own network partition and treats others as black boxes to some extend. Thus, it is possible for some potential applications which need to geographically distribute simulators, for example, to the center location of each real network partition being simulated, and allow each of them dynamically adjusts its network partition as long as the communication interface is maintained.

4.1.2 Genesis Approach

Genesis was designed to support distributed simulation with both full network topology and distributed memory. In this chapter, the design and experimental results for distributed simulation with full network topology are presented first. The advantage for keeping the full network topology is that it requires relatively small changes to implement a distributed simulation system based on a sequential simulation system. Such a solution introduces also smaller overheads.

In addition, in the memory distribution approach, each simulator has routing information of only part of the network, which requires additional work to collect the global routing information and forward network traffic among simulators correctly. A design of such a global routing information sharing mechanism might impose limitations on the simulation. For example, DaSSFNet requires pre-configured static forward tables. In Genesis, we implemented our memory distributed simulation system which can represent dynamic global routing changes, with the limitation that network partitioning must observe the boundaries of BGP Autonomous Systems (ASes) or group of ASes.

In our Genesis network simulation with full network topology, users have more freedom to group any neighboring nodes into domains.

4.2 Distributed Simulation System Model

4.2.1 System Components

Genesis took some common approaches for parallel and distributed simulation systems and had all the general components for these systems, while adjusted them to meet the special needs of coarse granularity synchronization.

In conventional parallel or distributed simulation which uses the space partitioning technique to divide network into domains, the system usually consists of these general components:

1. Network partitioning. The network topology being simulated is logically partitioned into areas, and each area is assigned to one processor. The simulation script which defines the network provides some functionalities to divide the network and assign processors.

2. Concurrent Simulation. Network areas are simulated concurrently on different processors. Each processor simulates only the part of the network assigned to it and handles events generated from this part of network, or events received from other processors.

3. Data management. In conventional simulation, the simulation data exchanged among processors are events. Events originated from one processor and targeted to another processor are remote events. The parallel or distributed simulation system should recognize these remote events and forward them to the correct destination, by using either shared memory or explicit information exchanging techniques (e.g. MPI, socket connection).

4. Time management. Parallel or distributed simulators need to be synchronized. As we explained earlier, different synchronization approaches are designed to

achieve the same goal that in each processor, events are handled in the correct order of their time-stamps.

In our novel simulation system using coarse granularity synchronization technique, there are differences in the roles and functions of these components:

1. Network partitioning. A network topology is partitioned in the same way as in conventional simulations, and each network partition is assigned to one processor. However, it does not mean that one processor will only simulation the partition assigned to it. Instead, the assigned partition is the "simulation focus" of this processor, and fragments of other partitions related to this one will also be simulated in this processor.

2. Concurrent Simulation. Each processor simulates the network partition assigned to it in detail the same way as conventional simulation systems. However, processors do not exchange remote events among each other. Instead, each processor contains not only its own part of the network, but also a simplified model of the rest of the network. Thus, a "remote" event related to the rest of the network can be delivered to the corresponding simplified network model. In this way, there is no need to exchange "remote events", all events are "internal events" to a processor.

3. Data management. Data management in Genesis is different from conventional systems. No remote events need to be exchanged among processors. As explained above, for one processor, the simplified network model serves as a representation of the part of network simulated in details by other processors, in other words, the "outside world". In order to correctly represent the "outside world", each processor collects simulation statistics data from the part of network assigned to it and exchange them with other processors. And then, it uses the data received from other processors to adjust the network model representing the "outside world".

4. Time partitioning and management. Time management in Genesis is different because no remote events need to be synchronized. Instead, the simulation

time is partition into intervals separated by checkpoints. During each checkpoint, the simulation time of every processor is synchronized and convergence decision is made. Based on the received data from its peer domains, a domain simulator might need to re-iterate one simulation time interval to produce more accurate results.

4.2.2 Network Partition and Domain Model

In Genesis, network partitions are called “domains”. For one processor, the domain assigned to it is called the “active domain”, and the domains assigned to other processors are called “non-active domains”.

In the “active domain”, the network structure is the same as the non-decomposed network. In “non-active domains”, traffic sources and destinations are represented by *proxy sources*, which can be activated or deactivated dynamically during the simulation. “Path shortcuts” are used to simplify any traffic paths in non-active domains. They are implemented as *proxy links* which connect *proxy sources* directly to border routers in the domain. Simulation statistics data are used to adjust these *proxy links* to represent network traffic changes during the simulation.

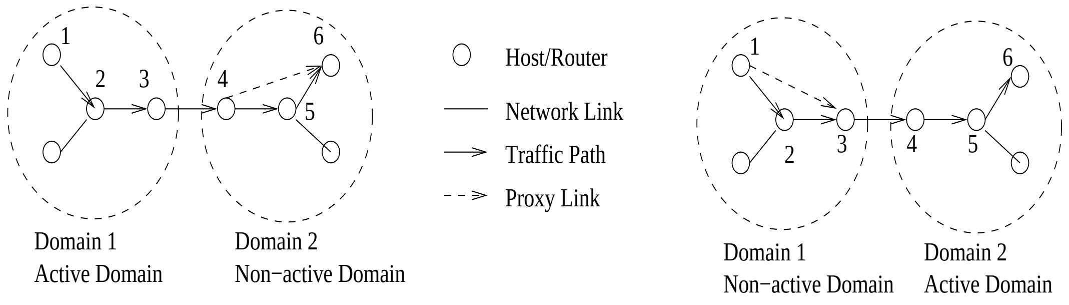

Figure 4.1: Path Shortcuts and Proxy Links

Figure 4.1 shows an example of this domain model. Suppose that a network is partitioned into two domains and simulated by two processors: domain 1 is assigned to processor 1 and domain 2 is assigned to processor 2. Suppose that there is a network traffic from node 1 to node 6 through nodes 2, 3, 4 and 5. In processor 1, domain 1 is the active domain while domain 2 is a non-active domain, thus part of the traffic path, from node 1 to node 3, is inside of the active domain and the other part, from node 4 to node 6, is inside of the non-active domain. On the contrary, in processor 2, the path from node 1 to node 3 is in the non-active domain while the path from node 4 to node 6 is in the active domain.

Processor 1 simulates traffic packets from node 1 to node 4, and creates a *proxy link* from node 4 connecting directly to the destination node 6 in the non-active domain. Node 6 is represented as a *proxy source* in processor 1. At the same time, processor 1 collects traffic statistics data from node 1 to node 3, and sends these data to processor 2.

Concurrently with processor 1, processor 2 simulates traffic packets from node 4 to node 6. This is done by creating a *proxy source* in non-active domain 1 to represent traffic source node 1, and creates a *proxy link* from node 1 connecting directly to node 3. At the same time, it collects traffic statistics data from node 4 to node 6 and sends them to processor 1. Both processors use received data to adjust their own *proxy links*.

In this way, these two processors simulate the network concurrently and cooperate with each other. Each processor is only responsible for the simulation of its active domain, and collects simulation statistics data within the active domain. However, the *proxy source* and *proxy link* structure for non-active domains is also essential that it completes the traffic path from the source node to the destination node. As the result, in Genesis, each processor simulates full traffic paths from source nodes to destination nodes. This is important because for TCP traffic, source nodes must receive ACK packets from destination nodes to continue its packet sending.

4.2.3 Synchronization and Checkpointing

In Genesis, domain simulators cooperate with each other by exchanging simulation statistics data periodically. The Genesis framework synchronizes domain simulators by dividing the simulation time into intervals and performing checkpointing at the end of each interval. During checkpointing, each domain simulator suspends its simulation, checkpoints its states, exchanges statistics data with other domain simulators, and checks convergence conditions.

Simulation state checkpointing is done by calling Unix system routine *fork* from the running process to duplicate the process image in memory, and immediately suspends the child process. The suspended child process serves as a backup copy of the current state of the running simulator. Checkpointing the simulator state makes it possible for the simulator to continue execution and roll back to a saved state when needed. When a simulator decides to roll back to its previous state, it will signal its suspended backup process to resume execution.

During checkpointing, each simulator also exchanges simulation data with other distributed simulators. By comparing the data from its peer simulators received in this checkpoint with those received in the last checkpoint, a simulator decides whether or not it needs to re-run the last interval. If it decides to re-run, the simulator stores the received data to some persistent storage (disk files), resumes the suspended child process and terminates itself. A resumed child process will make a new backup process of its own first, retrieves the simulation data stored by the terminated process, and continues its execution. A simulator re-iterates over one simulation time interval until it decides that the changes in received data are small enough according to some pre-defined convergence conditions.

For example, each domain may run its individual simulation from simulation time t_n to t_{n+1}, and pause thereafter. Then, statistics data about the traffic packets leaving the domain during this time interval is passed onto the target domain to which these packets are directed. If the traffic data differ significantly from what was assumed in the target domain, the simulation of the time interval (t_n, t_{n+1}) is repeated. Otherwise, the simulation progresses to the next time interval (t_{n+1}, t_{n+2}).

4.2.4 Statistics Collection and Convergence Conditions

During the simulation of one time interval, statistics data of network traffic are collected for each flow based on some pre-set metrics. Currently in Genesis, these metrics are packet delay and path drop rate. Packet delay measures the time expired from the instance a packet leaves its source in the active domain to the time it reaches the domain boundary. Packet drop rate is computed as the percentage of dropped packets out of total sent packets in the active domain for each flow. Also recorded is information about each packet source and its intended destination. Having this information enables us to set up *proxy sources* and *proxy links* in non-active domains, and use statistics data to adjust *proxy links*. A *proxy link* will then apply the packet delay and drop probability data of one flow to the packets belonging to that flow which pass through it.

Statistics data are also used for convergence computation. The threshold values of the differences between the data received in the current checkpoint and those received in the previous checkpoint are pre-set by users. If traffic data changes exceed those thresholds, rollbacks and re-iterations are required; otherwise, the current interval is converged and the simulation proceeds to the next interval.

4.3 Simulator Component Design Based on SSFNet

The Genesis system model explained above introduced a new approach of constructing a distributed simulation system. However, Genesis is not only one network simulation system. Instead, it is a general approach which can be used to transform conventional sequential or parallel simulation systems into scalable distributed simulation systems, as well as constructing new systems from scratch. Besides this, different conventional systems transformed by the Genesis approach will be able to cooperate with each other.

In this section, we will introduce our Genesis system constructed on top of SSFNet. In order to integrate SSFNet simulator into Genesis, SSFNet has been extended with the following additions: Domain Definition, Proxy Sources and Proxy Links, Data Collector, Checkpointing and Freeze that are further discussed below.

1. **Network decomposition**. In SSFNet, a network is modeled as a hierarchy of "Net" that is a collection of hosts, routers, links and component sub-nets. Subnet inclusion is a powerful construct that facilitates building very large models from pre-configured sub-networks. Hierarchical "Net" is also a convenient tool for network partitioning required by Genesis. In Genesis, domain definition is simply implemented by adding domain identification numbers into the "Net" definition defining the corresponding network partitions. This domain information is stored in SSFNet Domain Modeling Language (DML) configuration database. The modified *Net* class will retrieve its domain identifier from DML

configuration database and store it at its global data area, which makes it easily accessible by other components.

2. **Proxy traffic source** is a modified traffic source which can be deactivated or re-activated by a controller called “freeze”.

   In SSFNet, when traffic starts, the client will first connect to the known port of the server. Then, the client sends control data (including the size of the file requested) and waits for data from the server. Once the server is initialized, it listens to incoming connections from clients. After accepting a new connection, the server builds a data socket and spawns a slave server that transmits the data between the client and the server.

   If a traffic source is not in the current active domain, it will be deactivated after its initialization. In other words, traffic sources outside of the active domain will not automatically generate traffic in Genesis. The slave server for this deactivated traffic source is called a “proxy source”. The reference to a proxy source is registered at global area with corresponding flow identifier, so that it can be re-activated during checkpointing. When a proxy source is re-activated during the simulation it starts to send packets out. Proxy sources are connected directly to border routers of the active domain by some one-hop “short-cut” path, which is called “proxy links”.

3. **Proxy links** are used to implement “short-cut” paths. Traffic packets generated by proxy sources will not go through the regular network path defined in the network topology. Instead, each host of a proxy traffic source has a “dummy” interface which is connected to a border router of the active domain, via a special network link called “proxy link”, as shown in Figure 4.2.

   Proxy links are special links because they dynamically apply transmission delay and packet drop to the traffic flow passing through it. The values of delay and drop rate are adjustable based on simulation statistics data.

   Each link in SSFNet is implemented by channel mapping between the two attached interfaces, “push” invoked by one interface will put a packet into its peers’ *inChannel* with appropriate delay. This mechanism is used for building

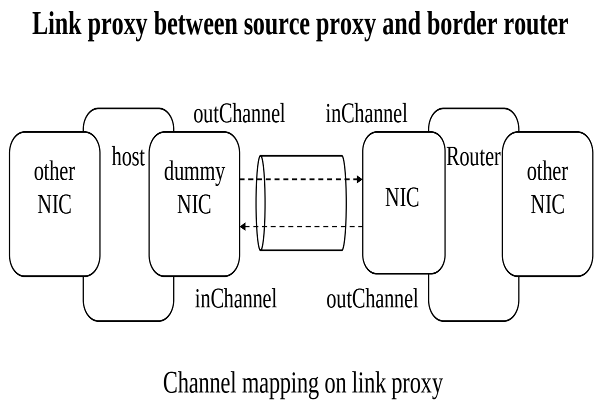
Link proxy between source proxy and border router
outChannel
inChannel
other NIC
host
dummy NIC
NIC
Router
other NIC
inChannel
outChannel
Channel mapping on link proxy

Figure 4.2: Proxy Link Design

proxy links which shortcut the path from proxy source to the corresponding border router of current domain. In addition, the *IP* class in SSFNet has also been enhanced to (i) sent outgoing data through proxy link instead of the normal route, (ii) dump information about outgoing data into statistics data collector, and (iii) preserve the regular routing for control data.

4. **Traffic statistics data collection**. This is done by adding class *Collector* to SSFNet as a global container to hold flow-based information. The working mechanism of *Collector* is based on the packet-level simulation in SSFNet. There are three kinds of delay accumulated in the lifetime of a packet transmission in SSFNet. Link delay is configured as link latency. Queue delay is decided by the queue size, link capacity and traffic volume. NIC (Network Interface Card) delay is defined by the NIC latency. There are three cases in which packet gets dropped: (i) end of life time, (ii) no reachable destination (IP layer), and (iii) dropped by queue manager of its deporting interface (Link Layer). Using these rules, the delay of outgoing packets is accumulated for every flow. The number of packets fired and the number of packets dropped are also recorded.

5. **Simulation Freezing and Synchronization** relies on cloning of the simulation state at the beginning of each interval. The cloned copy is activated when the rerun is necessary. We use Java Native Interface (JNI) to do the

memory checkpointing and the interaction between Java and C copy routines is shown in Figure 4.3.

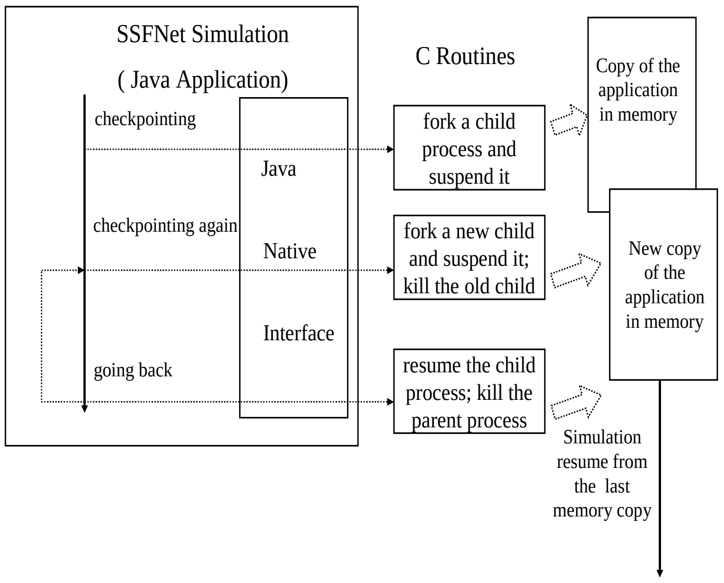
SSFNet Simulation
(Java Application)
checkpointing
checkpointing again
going back
Java
Native
Interface
C Routines
fork a child process and suspend it
fork a new child and suspend it; kill the old child
resume the child process; kill the parent process
Copy of the application in memory
New copy of the application in memory
Simulation resume from the last memory copy

Figure 4.3: In-Memory Checkpointing in Genesis Simulation

A Freeze component paces suspensions of the simulation. Frequency of suspension is defined in the DML configuration database. The simulation is interrupted by Freeze Events. Freeze object is wrapped with a Freeze Timer extended from Timer class of SSFNet. The call-back method of Timer is overloaded to fulfill freeze-related tasks.

Freeze Object is instantiated and initialized by Net object. At the end of initialization, it will register at DML databases, and then it will instantiate and set Freeze Timer. With self-channel mapping, Timer Event fired by Freeze Timer will be received by itself with some appropriate delay. Once a Timer Event is delivered, the call-back method will be invoked implicitly and will execute the exchange of information between domains.

4.3.1 Design of Feedback-based Protocol Simulations

The approach in Genesis described above was originally designed for protocols that generate packets without feedback flow control, such as Constant Bit Rate

(CBR) UDP traffic. However, modeling the inter-domain traffic which uses feedback based flow control, such as any of many variants of TCP, requires more processing capabilities.

To control congestion in a network or the Internet, some protocols use congestion feedback. The most important among them is TCP protocol used in TCP/IP-based Internet congestion control [69]. TCP uses sliding-window flow and error control mechanism for this purpose. The sliding-window flow control provides means for the destination to pace the source. The rate at which the source can send data is determined by the rate of incoming acknowledgment packets. Any congestion on the path from the source to destination will slow down the data packets on their way to destination and the acknowledgment packets on their way back to the source. As a result, the source will decrease its flow rate to lessen or eliminate the congestion. TCP flows demonstrate complex dynamics by adjusting their rate to the changing conditions on their paths to destination.

For our method, the important property of TCP traffic is that the rate of the source is dependent on the conditions not only in the source domain but also in all intermediate and destination domains of the traffic. Additionally, each data flow has a corresponding acknowledgment flow that paces the source. As a result, for the TCP traffic, the precision of our flow simulation depends on the quality of the replication of the round trip traffic by the packets and their acknowledgments. Moreover, the feedback loop for iterations is extended. For example, in two-domain TCP traffic, a change in congestion in the source domain will impact delays of data packets in the destination domain in the following iteration and the delays of the acknowledgment packets in yet subsequent iteration. As a result, convergence is slower in simulation of networks with TCP flows.

Our experience indicates that communication networks simulated by Genesis will converge thanks to monotonicity of the path delay and packet drop probabilities as a function of the traffic intensity (congestion). The speed of convergence depends on the underlying protocol. For protocols with no flow feedback control like UDP, simulations typically requires 2-3 iterations. As presented in this thesis later, protocols with feedback based flow-control, like TCP, require number of iterations up to an order of magnitude larger then UDP-like protocols.

The process of modeling feedback-based traffic is shown in Figure 4.4 and involves the following steps.

1. In the first iteration, the packets with a source within the domain and destination outside that domain flow along the path defined by the network routing through internal links to the destination proxy. We refer to such packets as DATA packets. The same internal links also serve as the path for the flow of feedback, that is acknowledgment, abbreviated as ACK, packets. The timing and routing information of both kinds of packets (DATA and ACK) within the domain are collected at the flow source and the destination proxy.

2. In the second iteration, the timing and routing information collected at the source domain is used to create a proxy source and in-link proxy in the destination domain. The proxy source in the destination domain is activated and the traffic external to the domain and entering the domain is simulated using information collected in the first iteration in the source domain. In addition, the timing and routing information within the destination domain for packets flowing to external nodes is collected at the destination proxy. This information will be used in the source domain to define the source domain in-link proxy that will reproduce ACK packets and send them to the original flow source.

3. In the third iteration, in-link proxy and proxy source are created in the source domain similar to iteration 2, but this time for the ACK packets returned by the flow destination. The timing and routing information is obtained from the previous iteration of the destination node and is used to initiate the flow of ACK packets within the source domain. This completes the definition of the full feedback traffic.

Note that, unlike the traffic without feedback control that uses one iteration delayed data to model traffic in the destination domain, delay here is two iterations. That is, the ACK packet traffic in the source domain in iteration n is modeled

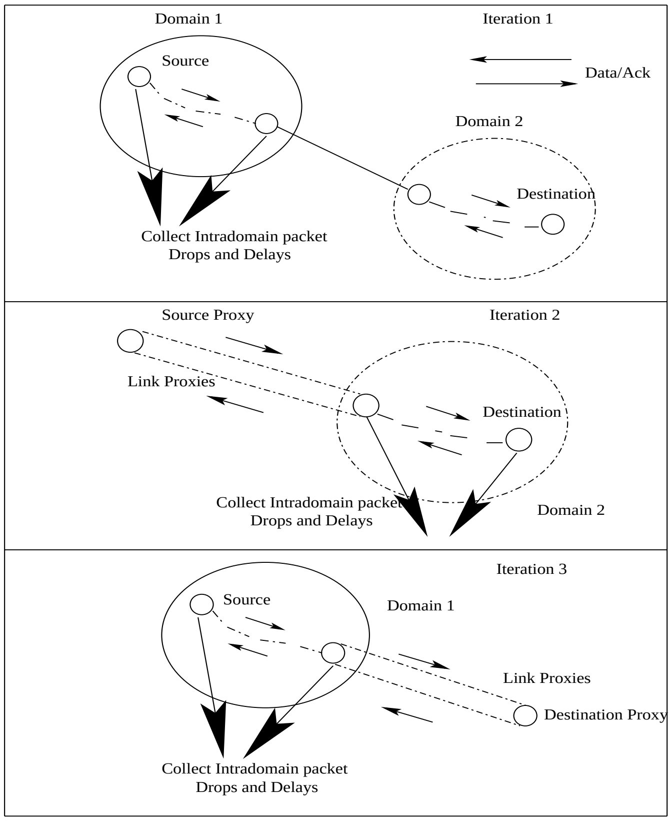

Figure 4.4: Increased Number of Iterations to Support Feedback-based Protocols

based on information from $n-1$ iteration about the ACK packets produced by the DATA packet flow that was modeled using information from $n-2$ iteration about the DATA packets in the source domain. Hence, there is delayed feedback involved in the convergence in this case, since an extra iteration is required to recreate the in-link proxy and proxy sources in both the source and the destination domains.

4.3.2 Transit Traffic Simulation

In Genesis, each domain simulator focuses only on the simulation of its active domain. Thus, in the simulation of UDP traffic, any packet sent from a UDP source inside the active domain stops at the border of the active domain and does not need to be forwarded further. If a packet is sent from a proxy UDP source (outside of the active domain), then it will be forwarded into a proxy link which is connected with the active domain. In both cases, only part of the traffic path, from the traffic

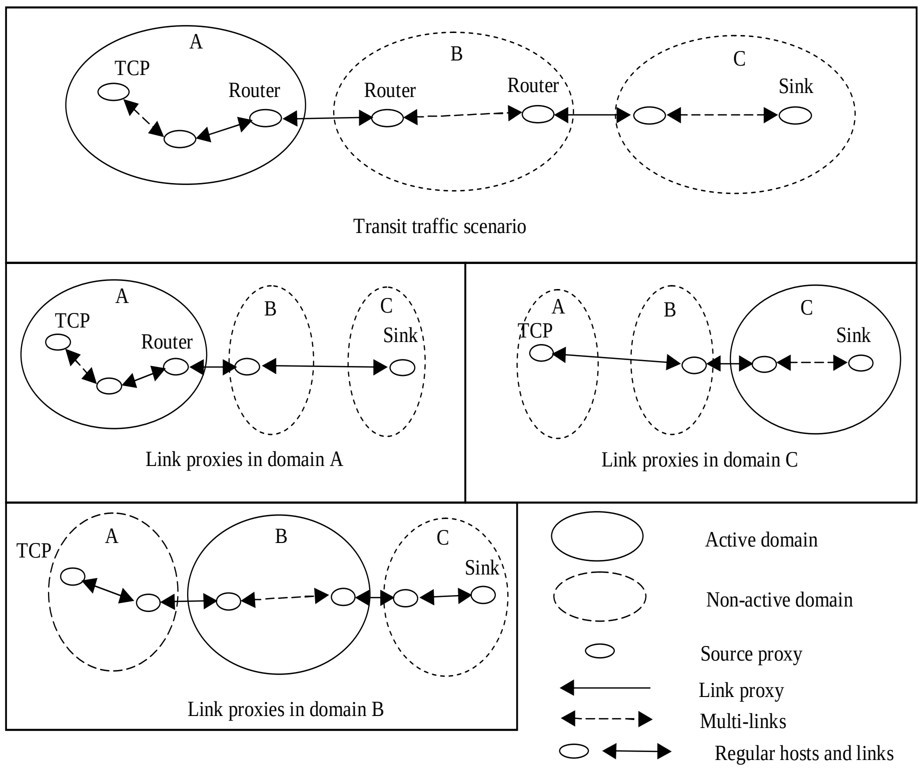

Figure 4.5: Transit Traffic Simulation

source to the last node inside of the active domain, need to be simulated. In this way, all the UDP traffics leaving or entering the active domain can be simulated, and at most just one link proxy need to be set up for a flow. However, this is not enough for TCP simulation. For TCP flows, in order to get the feedback flow (ACK) from the sink, each flow must go through the full path to its sink. When the TCP flow goes through more than two domains, multiple pairs of link proxies will need to be set up for this flow for the domains which are in the middle of the path. This is called the transit traffic scenario and shown in Figure 4.5. In such a case, more complexities will be involved into data collecting and link proxy setup, as explained below.

In the example network shown in Figure 4.5, for the domain simulator B, the traffic flow from domain A to C is a transit traffic. Both the source and destination of the traffic are outside of the active domain B. For such a transit traffic scenario, the network is viewed as consisting of three parts: the current active domain, B, and the two parts of the network outside this domain on both directions of the flow, A and C. Initially, the transit flow does not exist in the simulator for domain B. After one iteration, the simulator for domain A detects the outbound flow targeting domain B, and passes this flow information to domain B simulator during checkpointing. Based on this information, domain simulator B activates the corresponding traffic source in domain A and creates one pair of proxy links to connect the source to domain B. Similarly, domain B receives flow information from domain C and creates another pair of proxy links to connect the traffic sink in domain C to domain B. Thus, two pairs of link proxies will be set up for one transit flow. Domain B will then re-iterates the previous time interval with this transit traffic activated. In other cases when a transit flow passes through more than one intermediate domains, the transit flow information will be passed down to the domains along the traffic path and will activates the proxy sources in those intermediate domains in turns. More intermediate domains will require more re-iterations, however, each intermediate domain only needs two pairs of proxy links for one transit flow.

4.4 Simulator Component Design Based on ns2

As we mentioned earlier, Genesis is a general approach and can be applied to different existing simulation systems. In addition, Genesis also supports integration of different simulators into a coherent network simulation. In this section, we provide an overview of the implementation of Genesis based on another popular network simulator, ns2, in addition to our SSFNet based implementation.

The following extensions were integrated into ns2 to support distributed simulations in Genesis:

1. **Domain definition** in ns2-based Genesis is similar to that in SSFNet-based Genesis. The difference is that Tcl/Tk is used as the network scripting language instead of DML. To declare a domain in the Tcl script, the nodes belonging to that domain are defined as parameters to a new "domain" command and are stored as a list. Each time a new domain is defined, the new node list is added to a domain list. The user also marks a domain as the active domain in the network script.

2. **Proxy Link** in ns2-based Genesis is implemented by extending the existing *connector* class in ns2. A modification has been made to the original ns2 connector class by adding functionality of filtering out packets destined for the nodes outside the domain and storing them for statistical data calculation. In ns2, a connector object is generally associated with a single link. When a link is set up, the simulator checks if this link connects nodes in different domains. If this is the case, this link is classified as a cross-link and the connector associated with this link is modified to record packets flowing across it. Each such packet is either forwarded to the neighboring node or is marked as leaving the domain based on its destination.

3. **Proxy Source** is implemented by extending the existing traffic generator class. The ns2 traffic generator class is modified, so the traffic sources can be activated or deactivated as needed. Initially, at the start of the simulation, the traffic generator suppresses nodes outside the domain from generating any traffic. During the simulation, traffic generators in the nodes which are representing *proxy sources* in non-active domains will be activated.

4. **Connecting Proxy Links to Proxy Sources**. In ns2, connectors are defined as "an nsObject with only a single neighbor". In Genesis, connectors are extended to have a list of proxy targets. Once a new proxy link is set up from this connector, it will be added to this connector's proxy target list.

5. **Border** is a new class added to ns2. A border object represents the active domain in the current simulation. The main functionalities of the border class include:

   (a) Initializing the current domain: setting up the current domain id, assigning nodes to different domains, setting up the data exchange etc..

   (b) Collecting and maintaining information about the simulation objects, such as a list of traffic source objects, a list of the connector objects and a list of the proxy link objects maintained by the border object.

(c) Implementing and controlling the proxy traffic sources; setting up and updating proxy links, etc..

The traffic sources outside the current active domain are deactivated while setting up the network and domains. When a proxy link is set up for a flow, the traffic source of this flow will be reactivated. When the border receives flow information from other domains, it will set up a proxy link for this flow, and initialize the parameter of the proxy link using the received statistical data. All the created proxy link objects are stored in the border as a linked list ready for further updates.

6. **Synchronization and Checkpointing** is done in the same way as in SSFNet-based Genesis. It is another implementation of the Genesis framework in C++ for ns2, in addition to the Java implementation for SSFNet.

4.5 Experiments

4.5.1 Network Models

Our first set of experiments for the Genesis involved two sample network configurations, one with 64 nodes and 576 UDP and TCP flows, the other with 27 nodes and rich interconnection structure with 162 UDP and TCP flows. These networks are symmetrical in their internal topology. We simulated them on multiple processors by partitioning them into different numbers of domains with varying number of nodes in each domain. The rate at which packets are generated and the convergence criterion are varied.

All test were run on up to 16 processors (always the number of processors used is equal to the number of domains) on Sun 10 Ultrasparc and 800 MHz IBM Netfinity workstations. For both architectures, the machines were interconnected by the 100Mbit Ethernet.

For the 64-node network, the smallest domain size is four nodes; there is full connectivity among these four nodes. Four such domains together constitute a larger domain in which there is full connectivity between the four sub-domains. Finally, four large domains are fully connected and form the entire network configuration for the 64-node network, as shown in Figure 4.6.

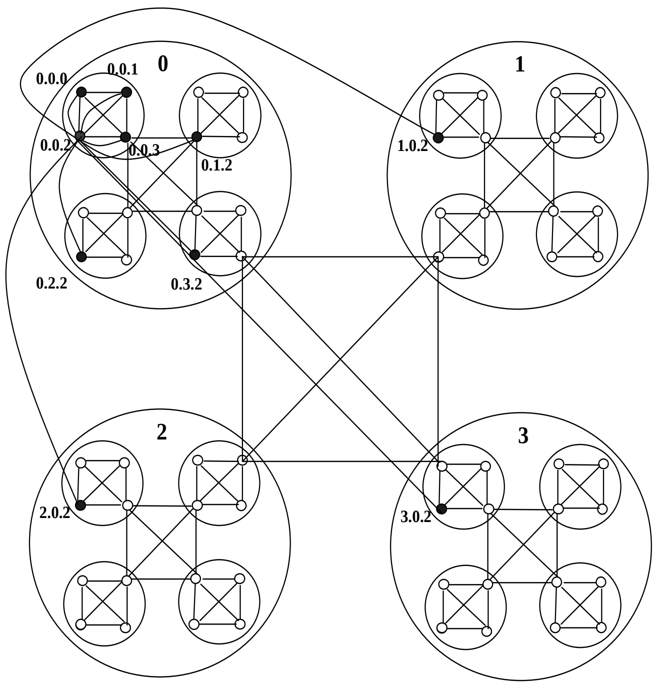

Figure 4.6: 64-node Configuration Showing Flows from a Sample Node to Other Nodes In a Network

Each node in the network is identified by three digits *a.b.c*, where a,b,c is greater than or equal to 0 and less than or equal to 3. Each digit identifies domain, sub-domain and sub-sub-domain and node rank, respectively, within the the higher level structure.

Each node has nine flows originating from it. Symmetrically, each node also acts as a sink to nine flows. The flows from a node *a.b.c* go to nodes:

$a.b.(c+1)\%4$	$a.b.(c+2)\%4$	$a.b.(c+3)\%4$
$a.(b+1)\%4.c$	$a.(b+2)\%4.c$	$a.(b+3)\%4.c$
$(a+1)\%4.b.c$	$(a+2)\%4.b.c$	$(a+3)\%4.b.c$

Thus, this configuration forms a hierarchical and symmetrical structure on which the simulation is tested for scalability and speedup.

The 27-node network is an example of Private Network-Network Interface (PNNI) network [3] with a hierarchical structure. The main feature of PNNI protocols is "scalability", because the complexity of routing does not increase drastically as the size of the network increases. Although we do not support simulation of PNNI protocols in Genesis, we took this network model as an interesting test case for Genesis approach. The smallest domain of this 27-node network is composed of three nodes. Three such domains form a large domain and three large domains form the entire network (cf. Figure 4.7).

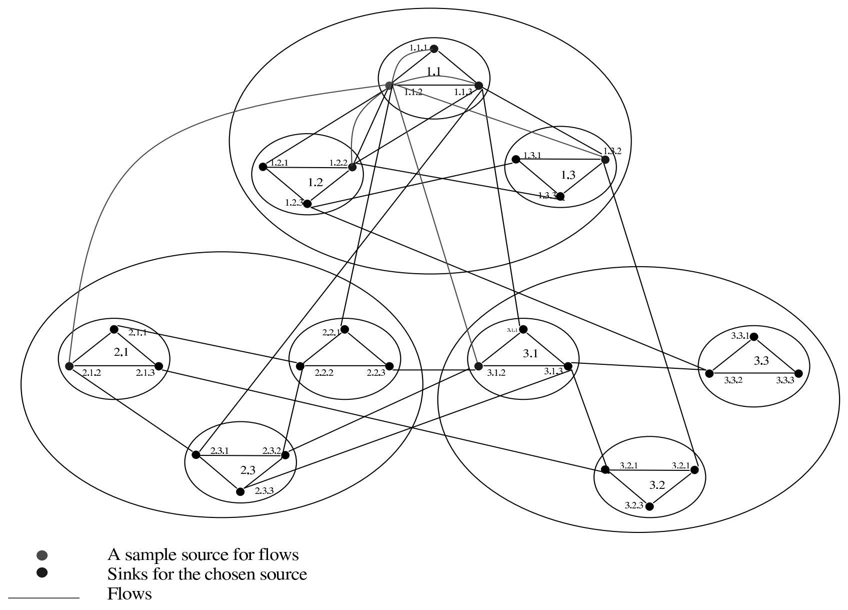

Figure 4.7: 27-node Configuration and the Flows from the Sample Node

In the 27-node network, each node has six flows to other nodes in the configuration and is receiving six flows from other nodes. The flows from a node $a.b.c$ can be expressed as:

$a.b.(c+1)\%3$	$a.b.(c+2)\%3$
$a.(b+1)\%3.c$	$a.(b+2)\%3.c$
$(a+1)\%3.b.c$	$(a+2)\%3.b.c$

4.5.2 Experimental Results

In a set of measurements, the sources at the borders of domains produce packets at the rate of 20000 packets per second for half of the simulation time. The bandwidth of the link is 1.5Mbps. Thus, certain links are definitely congested. For the other half of the simulation time, these sources produce 1000 packets per second. Since such flows require less bandwidth than provided by the links connected to each source, congestion is not an issue. All other sources produce packets at the rate of 100 packets per second for the entire simulation. The measurements were done with the Telnet traffic source that generates packets with constant size of 500 bytes.

Speedup was measured in three cases involving (i) feedback-based protocols, (ii) non-feedback based protocols, and (iii) the mixture of both, with UDP traffic constituting 66 percent of flows and TCP traffic making up the rest of the flows. We noticed that if mixed traffic involves a significant amount of non-feedback based traffic, then it requires fewer iterations over each time interval and hence yields greater speedup up than the feedback based traffic alone.

Table 4.1: Measurements Results on IBM Netfinity (Times are in Seconds) for Non-feedback Based Protocols. Large Domains Contain 9 or 16 Nodes and Small Domains Contain 3 or 4 Nodes

Networks	27-nodes	64-nodes
Non-decomposed Network	3885.5	1714.5
Network with Large Domains	729.5	414.7
Network with Small Domains	341.9	95.1
Speedup	11.4	18.0
Distributed Efficiency	126%	112%

Table 4.1 presents a small subset of the timing results obtained from the simulation runs. It shows that partitioning the large network into smaller individual domains and simulating each on an independent processor can yield a significant speedup.

Initial implementation of feedback based protocols used delays from the previous iteration as a starting value for the next iteration, leading to modest speedup (cf. Table 4.2 and Figure 4.8). The fixed point solution delay lays in between the

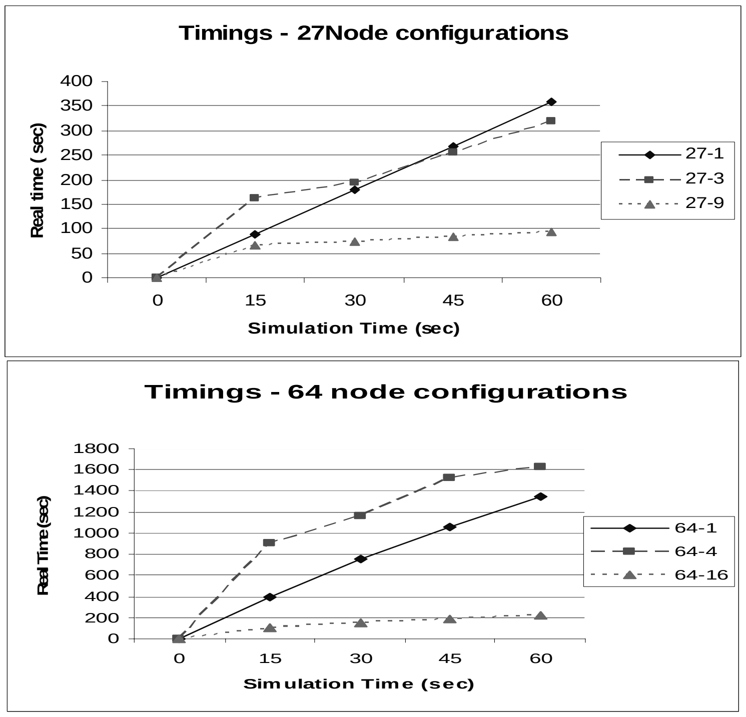

Figure 4.8: Speedup Achieved for 27 and 64-node Network for TCP Traffic

Table 4.2: Measurements Results on IBM Netfinity (Times are in Seconds) for Feedback-based Protocols

Networks	27-node-TCP	64-node-TCP	64-node-mixed
Non-decomposed Network	357.383	1344.198	6280
Network with Large Domains	319.179	1630.029	1120
Network with Small Domains	93.691	223.368	298
Speedup	3.81	6.02	21
Distributed Efficiency	42.3%	37.5%	131%

delays measured in the two subsequent iterations. Hence, a delay for each flow used in the next iteration is a function of the delays from the current and previous iterations. As expected, using this method of computing delay improves the Genesis performance. This is shown in Figure 4.9 and Table 4.2 for 64-node domains with mixed traffic. If d_{old} is the previous estimate of the delay, and d_m is currently observed values of the delay, then the new estimate of the delay is computed as $d_{new} = a * d_m + (1 - a) * d_{old}$, where $0 < a < 1$. Varying values of the parameter a

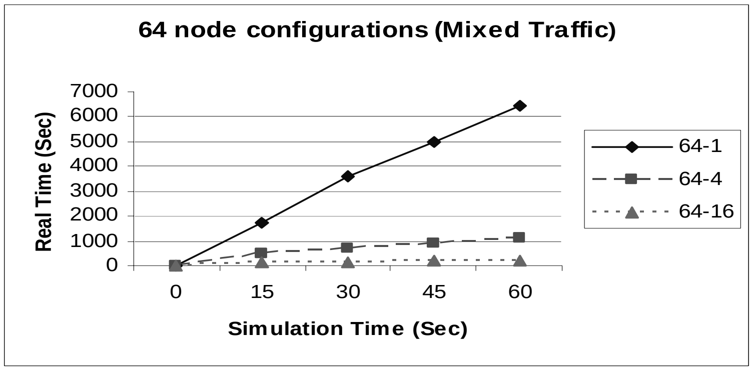

Figure 4.9: Speedup Achieved for 64-node Network for Mixed Traffic TCP-1/3 UDP-2/3

impacts the responsiveness of the delay estimate to new and old values of observed delays. As a result, a impacts the speed of the simulation by increasing/decreasing the time required for convergence, the optimum value of a is 0.5.

We will discuss Genesis performance and the effects of different parameter values in more details in Chapter 7.

4.6 Chapter Summary

Genesis is a novel approach for large scale network simulation. In this chapter we presented an overview of the Genesis approach, which combines simulations and modeling in a single system. In Genesis, each domain simulator simulates its own network partition in details, while using statistics data collected from its peer simulators to simplify the model and simulation of other partitions of the network. Distributed domain simulators run concurrently and cooperate with each other in the Genesis coarse granularity synchronization framework.

Our experimental results showed that Genesis worked efficiently in distributed network simulations. Thanks to its coarse granularity synchronization approach which did not require exchanging and synchronizing each individual remote packet, Genesis significantly reduced the overheads in distributed simulations. It achieved significant execution speedups over conventional sequential network simulations.

CHAPTER 5
DISTRIBUTED BGP SIMULATION

5.1 Introduction

The Internet consists of tens of thousands of inter-connected networks. Despite its enormous size, the Internet is organized into a hierarchical topology: the top level ISPs (Internet Service Providers) connect with each other to form the Internet backbone network, and there are many secondary and smaller ISPs where each of them attaches to one or more ISPs on a higher level, and then many other networks connect to these ISP networks. Each of these networks is independently managed and there is no central control about how a packet should be routed from one computer in the Internet to another. Because of this inherent characteristic, the routing in Internet is done in a two-leveled way: the top-level inter-domain routing decides how to route a packet from one network to another, while the second-level intra-domain routing decides how to route a packet from one computer to another in the same network. Some common routing protocols are RIP, OSPF and BGP. RIP and OSPF are used for intra-domain routing while BGP (Border Gateway Protocol) is widely used for inter-domain routing in the Internet nowadays.

Because BGP is widely used in the Internet for inter-domain routing, many models of large-scale networks from the real world are BGP networks. This requires that a modern simulation system for large-scale networks should provide the capability of BGP simulation. On the other hand, because of the importance of BGP to the Internet performance, the study of BGP protocol itself is also of interest. A system which supports detailed BGP simulation is a powerful tool for analyzing the BGP protocol.

Dartmouth's BGP4 [62] implementation extended SSFNet with BGP supports. It was the first detailed BGP model for network simulators. The model was built within the framework of SSFNet simulation suite and had already been used by many researchers in their studies, including BGP protocol improvements [60], security [54] and attacks on BGP [49].

However, with a simulation system which supports detailed BGP simulation, there are still some challenges to simulate Internet-scale, realistic BGP network models, as explained by David Nicol in [50]. One of these challenges is the large memory size required by large-scale BGP simulation. Because the storage size for each individual BGP route can be on the order 1000 bytes in the Internet, a detailed simulation of 10,000 routers will require on the order of 10Gb of memory [50]. This huge memory requirement will be a bottleneck for large-scale BGP simulations. Although super computers with huge shared memory can be a solution to this challenge to some extent, we believe that a thorough solution will need to distribute this memory.

Thanks to its SSFNet based implementation, with little efforts, Genesis supported detailed BGP simulation within its domain simulators. Besides this, one important contribution of Genesis was that its distributed domain simulators could cooperate with each other to support distributed BGP simulation. With the memory distribution technique which will be explained in the next chapter, Genesis completely distributed BGP simulation, from execution to memory usage, and significantly reduced the memory requirement for each single simulation machine.

Genesis was the first simulator which supported distributed, detailed BGP simulation. It facilitated the simulation of large-scale BGP network simulation and improved simulation efficiency. In addition, Genesis made it possible to simulate very large BGP networks on computer clusters.

5.2 Distributed BGP Simulation Design

The Genesis was originally designed for network data traffic simulation, e.g., for TCP or UDP data traffic. In many network simulations, the real data carried by UDP/TCP packets are not important to the simulation results. However, for BGP packets, summaries of packet flows are not enough. The information stored in a BGP update packet needs to be faithfully delivered to its destination. In addition, these packets need to be delivered in the correct time-stamp order. These two requirements are necessary to preserve correct network dynamics in the simulation. Hence, new synchronization mechanism is needed in Genesis to satisfy these two requirements.

5.2.1 Design of Event-level Synchronization for BGP Simulations

We developed an event-level synchronization mechanism which can work within the coarse granularity synchronization framework of Genesis. We modified Genesis to support the simulation of BGP protocol, in addition of TCP and UDP protocols. In Genesis, when we simulate a network running BGP protocol for inter-AS (Autonomous System) routing, with background TCP or UDP traffics, we decompose the network along the boundaries of AS domains. Each parallel simulator simulates one AS domain, and loosely cooperates with other simulators. When there are BGP update messages that need to be delivered to neighbor AS domains, the new synchronization mechanism in Genesis guarantees that these messages will be delivered in the correct time-stamp order.

In other simulation systems which use only event-level synchronization based on either conservative or optimistic protocol, the correct order of event delivery is guaranteed by the protocol. The price, however, is frequent synchronization.

In Genesis, we took advantage of coarse granularity synchronization for TCP and UDP traffic, and at the same time synchronized BGP update messages by doing extra rollbacks, to reflect the actual routing dynamics in the network.

In Genesis, simulators are running independently of each other within one iteration. To simulate BGP routers separately from the Genesis domain in each parallel AS domain simulator, and to make them produce BGP update messages for its neighbor domains, we introduced proxy BGP neighbor routers. Those are routers mirroring their counterparts which are simulated by other simulators. The proxy BGP routers do not perform the full routing functionality of BGP. Instead, they maintain the BGP sessions and collect the BGP update messages on behalf of their counterpart routers.

At the synchronization point in Genesis, the BGP update messages collected in the proxy BGP routers, if there are any, are forwarded to the corresponding destination AS domain simulators through a component called BGP agent. These update messages are delivered to the BGP agent in the destination AS domain through the connections among BGP agents, and are distributed there to the BGP routers which are the destinations of these messages.

This framework enables the system to exchange real BGP message data among Genesis simulators. But this is not a full solution yet. Within the independent simulation of one iteration in Genesis, BGP routers produce update messages for their neighbors, but do not receive update messages from their neighbors in other AS domains. Had they received these update messages, as it happens in an event-level synchronization simulation system, they would have probably produced different update messages. In addition, the routing might also have been changed. To simulate BGP protocol correctly, these BGP updates need to be executed in their correct time-stamp order in each BGP router. Genesis achieved this event-level synchronization for BGP updates by doing extra rollbacks.

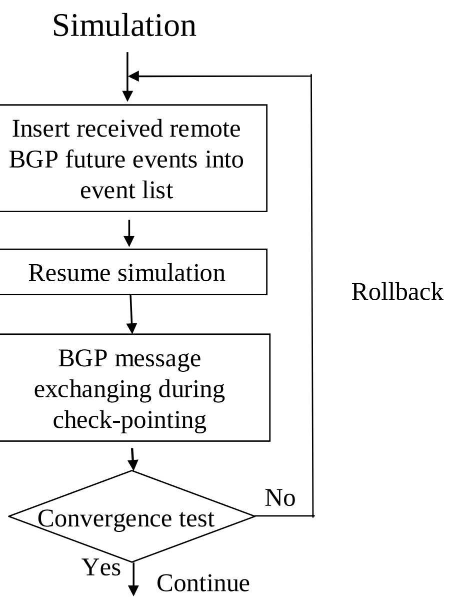
Simulation
Insert received remote BGP future events into event list
Resume simulation
BGP message exchanging during check-pointing
Convergence test
No
Rollback
Yes
Continue

Figure 5.1: Synchronization for BGP Update Messages

During the Genesis checkpoint after one time interval, the BGP agent in each AS domain collects BGP update messages from other BGP agents. If it receives some update messages for the previous interval, it will force the AS domain simulator to rollback to the start time of the previous interval. Then, it inserts all the received update messages into its future event list. Its domain simulator will reiterate the time interval again, and will "receive" these update messages at the correct simulation time and will react to them correspondingly. The BGP messages produced in the current reiteration might be different from those seen at previous iteration. Hence, the rollback process might continue in domain simulators until all of them reach a global convergence (for each domain, the update messages produced in subsequent rollback iterations are the same as those produced in the previous iteration). Figure 5.1 shows the flowchart of rollback in the BGP agent. High cost of checkpointing the network state makes it impractical to introduce separate rollbacks for BGP activities. Hence, the UDP/TCP traffic checkpoints are used for all rollbacks in Genesis.

5.2.2 Grouping Multiple ASes in One Genesis Domain

The initial design of distributed BGP simulation in Genesis supported partitioning the network on the boundary of each AS domains. As a result, each distributed Genesis simulator simulated one AS domain. The BGP AS domain IDs could be used as the identifier of Genesis simulators and were used to route BGP messages to the corresponding destination simulator. This design was straightforward but it required that the number of simulators must be the same as the number of BGP AS domains in the simulation. However, in the case of simulating a network with large number of BGP AS domains with relatively small number of machines, one distributed domain simulator need to simulate multiple BGP ASes. In other words, it is required that we could partition the network in a more flexible way to group multiple ASes into one Genesis domain. Another advantage of such a flexible partitioning is that for a given network with multiple BGP ASes, there could be multiple partitioning schemes to construct a distributed simulation with different number of machines.

To support this flexible network partitioning and study the effects of different partitioning schemes on distributed simulation, Genesis was extended to support AS domain grouping [44]. With AS domain grouping, one Genesis domain could have any number of BGP AS domains. During a synchronization point, the BGP agent for one Genesis domain collected all the BGP update messages stored in the

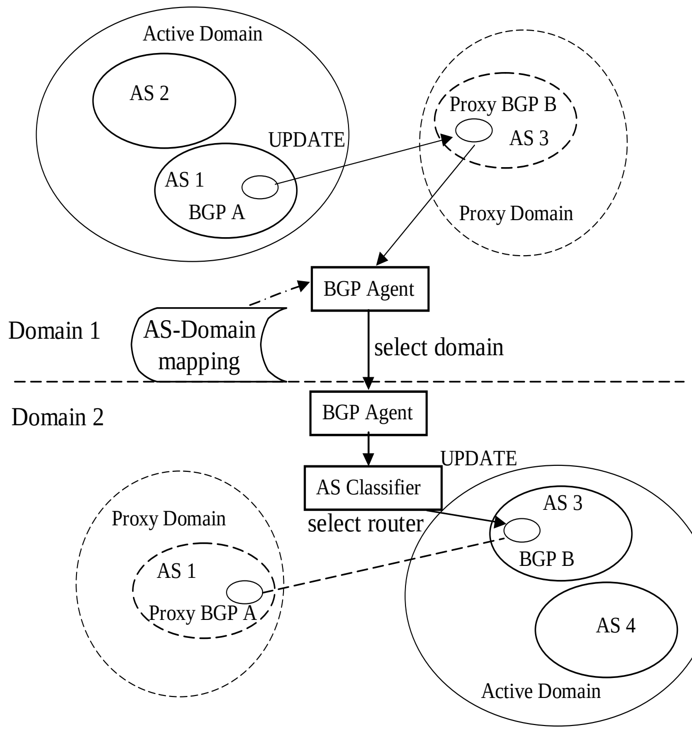

Figure 5.2: BGP Message Forwarding with AS Grouping

proxy BGP routers in this domain, and forwarded each of them to its destination BGP router. This forwarding was done through a two-layered routing, as shown in Figure 5.2.

To route a BGP message to its destination, the first-layer routing was a mapping between BGP ASes and Genesis domains. This could be done in two different ways: by specifying a grouping factor (e.g., N BGP ASes per domain) when each group had the same number of ASes, or by providing a mapping file when the grouping was not even. This grouping information was shared by all domain simulators and used to forward every BGP update message to the domain which contained the corresponding destination BGP router. The second-layer routing contained a message classifier in the BGP agent of each domain. The classifier held a list of all BGP routers in the ASes in that domain. The classifier first identified the destination BGP router by the address of each received BGP message, and then retrieved the reference of that router instance and called its receive-message method to deliver the message.

5.3 Simulation Component Design

An internal router of the current active domain in a domain simulator is called an "active router". When a BGP update message is received by an active router, the router will handle the update message the same way as a regular BGP router will do. If the receiving router is outside of the active domain, then this is an deactivated BGP router, or a "proxy BGP router". A proxy BGP router does not store any actual routing information and will not handle the actual routing update. Instead, it will send the received update message to its counterpart BGP router, which is in another domain simulator where it is an active BGP router. At the second domain, this update message can be handled correctly. In such a design, only the BGP routers in the active domain store and update the actual routing information.

5.3.1 BGP Update Message Transmission

To support distributed simulation, the actual BGP update message will need to be sent through the socket connections between BGP agents. The update message is a data structure that contains some attributes and some sub-structures. The two sub-structures are *Withdrawal Routes* and *Advertisement Routes*, for route withdrawal and advertisement, respectively. A *Withdrawal Route* is a vector of *IPaddress* objects, and an *Advertisement Route* is a vector of *Route* objects. Each *Route* object consists of one *IPaddress* object, and a list of path attribute objects. Path attributes are different classes of path attribute objects. Such an update message will need to be encoded into a string for transmission through a socket connection, as shown below:

```
UPDATE message encoding/decoding:
Class Name      String Representation
UpdateMessage   ‘‘msg_nh num_withdrawals  IPaddress1 IPaddress2 ...
                  num_routes nlri1#Attr1#Attr2#Attr3 nlri2#Attr1#Attr2#Attr3 ...’’
Route          ‘‘IPaddress#Attr1#Attr2#Attr3 ...’’
ASpath         ‘‘num_segments$Segment1$Segment2 ...’’
Segment        ‘‘type@num_AS@AS1@AS2@AS3 ...’’
```

The BGP agent at the receiving domain will decode the update message and reconstructs the update message data structure, and then delivers the message to the destination BGP router.

5.3.2 Update Timer and Output/Input Buffers

Output and input buffers are used for BGP update message exchanging among domain simulators. During one simulation time interval, each domain simulator buffers the BGP update messages received by proxy BGP routers, with their timestamps. During the checkpointing at the end of this simulation time interval, domain simulators exchange the BGP update messages stored in their output buffers. Because of possible rollbacks, domain simulators need to store received messages into a persistent storage, the input buffer file. If a domain simulator received at least one BGP update message with a timestamp smaller than its current simulation time, then at the end of the checkpointing it will roll back the simulation to its previous checkpoint.

The update timer is used to trigger the delivery of received messages. When a domain simulator rolls back and restarts from a checkpoint, it retrieves the BGP update messages from its input buffer file, and inserts them into its future event list by utilizing a BGP update timer. Thus, during the reiteration of the same simulation time interval, the update timer will deliver these BGP update messages originated from peer domains into the simulation at correct simulation time.

5.4 Experiments with Distributed BGP Simulation

5.4.1 Campus Network Model

Our distributed BGP simulation experiments used a modified version of the baseline model defined by the DARPA NMS community [56]. This baseline model is a large-scale BGP network with a topology which can be visualized as a ring of nodes, where each node (representing an AS domain) is connected to one node preceding it and another one succeeding it. We refer to each node or AS domain as the "campus network", as shown in Figure 5.3. Each of the campus networks is similar to the others and consists of four subnetworks. In addition, there are two additional routers not contained in the subnetwork, as shown in the diagram. The total number of AS domains is adjustable for different experiments. Such a synthetic network model consisting of BGP domains with the same topology makes it easy to automatically generate networks with large number of AS domains.

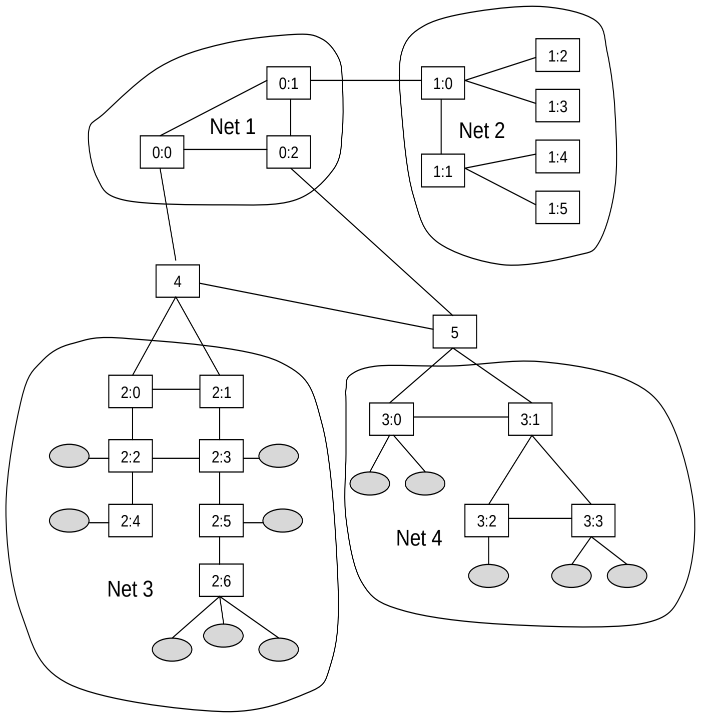

Figure 5.3: One Campus Network

Figure 5.3 shows the internal topology of one AS domain. The subnetwork labeled Net 0 consists of three routers in a ring, connected by links with 5ms delay and 2Gbps bandwidth. Router 0 in this subnetwork acts as a BGP border router and connects to other campus networks. Subnetwork 1 consists of 4 UDP servers. Subnetwork 2 contains seven routers with links to the LAN networks as shown in the diagram. Each of the LAN networks has one router and four different LAN's consisting of 42 hosts. The first three LAN's have 10 hosts each and the fourth LAN has 12 hosts. Each of the hosts is configured to run as a UDP Client. Subnetwork 3 is similar to Subnetwork 2, so internal links and LAN's have the same property as those in Subnetwork 2.

The traffic that is being exchanged in the model is generated by all the clients in one domain choosing a server randomly from the Subnetwork 1 in the domain that is a successor to the current one in the ring.

Our experiments were run on a cluster of Sun UltraSPARC-III, 750MHz dual-cpu machines, which were interconnected by a 100Mbit Ethernet. Each Genesis domain simulator was assigned to one processor.

5.4.2 Convergence with Different Network Topologies

In this set of distributed simulations, we partitioned a network in the way that each Genesis domain contained one Campus Network (CN), and assigned it to one simulation machine, thus the number of machines (processors) used in simulation was equal to the number of CNs (ASes) in each experiment.

In BGP network simulations, the first round of BGP update message burst happens when AS domains start to exchange BGP information to set up the global inter-AS routing. In Genesis, AS domains are simulated distributively, and BGP update messages are synchronized by re-iterating over one time interval until the BGP messages exchanged among domains converge (no more changes) on that interval, as we showed in Figure 5.1. We measured the number of re-iterations required by this BGP convergence on different sizes of networks and different network topologies to evaluate performance.

In our experiments, we also defined the "maximum distance" in a network as the maximum length of the shortest paths between any two distributed domains in the network, where the path length was measured in the number of intermediate distributed domains on the path plus one.

The first set of experiments were done on the baseline topology as we described earlier, in which the Campus Networks (CN) were connected as rings, as shown in Figure 5.4. We measured the convergence of the synchronization for ring sizes of 3, 4, 8 and 12 CNs. Table 5.1 shows the number of re-iterations needed in Genesis to converge during the BGP message burst. It should be noted that the number of needed iterations grows sub-linearly with the number of domains.

Table 5.1: BGP Synchronization Convergence with Ring Topologies

Ring Size(No. of CN)	No. of Domains	Max Distance	Iterations
3	3	2	3
4	4	2	4
8	8	4	6
12	12	6	8

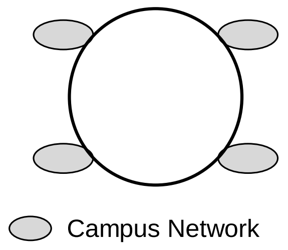

Figure 5.4: Four Campus Networks Connected as a Ring

The results show that the number of re-iterations is related to the size of the network, and is proportional to the maximum distance of a network. This can be explained as follows: because in each re-iteration, BGP messages are exchanged between neighboring distributed domains, thus they propagate to the domains with distance one to themselves. The maximum distance in a network decides how many re-iterations are needed for BGP decisions from one distributed domain to reach all the other domains in the network. Any received BGP message might cause response messages from the receiver and propagate to the rests in the network, and in each round the number of re-iterations is also decided by the maximum distance in the network. In our ring-based topologies, this maximum distance is proportional to the ring size.

In another set of experiments, we used line-based network topologies where different number of CNs were connected as lines, as shown in Figure 5.5. The convergence results are shown in Table 5.2.

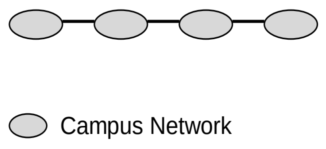

Figure 5.5: Four Campus Networks Connected as a Line

In a line topology, the longest distances for every CN to reach other CNs are not the same, where the CNs on both ends of the line have the biggest value which decides the maximum distance of the network. The maximum distance of a line-

Table 5.2: BGP Synchronization Convergence with Line Topologies

Line Length(No. of CN)	No. of Domains	Max Distance	Iterations
3	3	2	3
4	4	3	4
8	8	7	8
12	12	11	12

based network is bigger than that of a ring-based network of the same size, which is easy to understand. Our results show that the maximum distance decided the number of re-iterations required in convergence.

The results above show that when the number of Campus Networks increases, the number of re-iterations for BGP convergence increases as well. However, simulation of a larger BGP network does not necessarily require more re-iterations. This is because that it is the "maximum distance" of a network which decides the rounds of propagations, not the number of BGP ASes itself. A network with more connectivities among BGP ASes will significantly reduce this "maximum distance", thus reduces the number of re-iterations. To demonstrate this, in the following set of experiments, we connected Campus Networks as cliques, as shown in Figure 5.6, and varied the number of CNs as 3, 4, 8 and 12. The results are shown in Table 5.3.

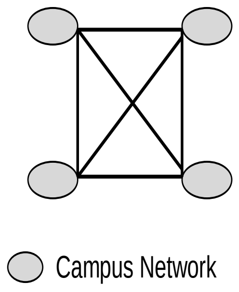

Figure 5.6: Four Campus Networks Connected as a Clique

In a clique topology, each CN is directly connected with all other CNs. It is obvious that the "maximum distance" of the network is 1 for each of these cliques, despite the number of CNs in the network. Simulation results showed that the number of re-iterations did not increase when the number of CNs in the clique

Table 5.3: BGP Synchronization Convergence with Clique Topologies

Clique Size(No. of CN)	No. of Domains	Max Distance	Iterations
3	3	1	3
4	4	1	3
8	8	1	3
12	12	1	3

increased while the “maximum distance” remained the same.

Besides network connectivities, network partitioning schemes can also affect the convergence performance. It is important that the “maximum distance” is decided by the distance between distributed domains instead of BGP ASes. In the experiments above, a network was always partitioned on the boundary of each BGP AS. In other words, each distributed domain had only one BGP AS. As a result, for a network with large number of BGP ASes, there will be large number of distributed domains and very likely the longest distance between two domains will be large (as we showed, this distance was also depended on the topology), and will need more reiterations to converge on BGP propagation. However, by grouping neighboring BGP ASes into one distributed domain, we can reduce the “maximum distance” of a network, and improve the convergence performance.

In the following set of experiments, 24 Campus Networks (CN) were connected in a ring, as described in the previous section earlier. We grouped N adjacent CNs into one Genesis domain, and constructed each domain distributively in Genesis domain simulators, as shown in Figure 5.7. N was varied as we set it to 8, 6, 3 and 2, thus the “maximum distance” of each network was, correspondingly, 2, 2, 4 and 6.

Table 5.4 shows the results of synchronization convergence. The results show that by grouping BGP ASes together, we had reduced the number of distributed domains and the number of convergence re-iterations. The cost was, however, the degree of parallelism was also reduced. On the other hand, because Genesis supports flexible BGP AS grouping, users have the freedom to select network partitioning schemes. The results from Table 5.3 give us a hint that a partitioning scheme which

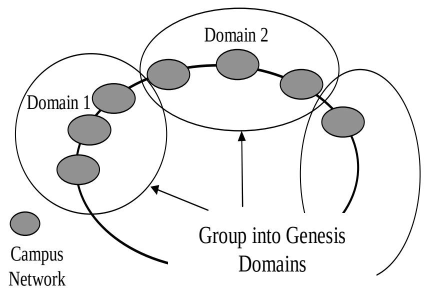

Figure 5.7: Grouping Adjacent Campus Networks

has higher connectivity among distributed domains will converge faster.

Table 5.4: Synchronization Convergence with Different Ring Sizes

Ring Size(No. of CN)	N	No. of Domains	Max Distance	Iterations
24	8	3	2	3
24	6	4	2	4
24	3	8	4	6
24	2	12	6	8

5.4.3 Grouping BGP ASes with Background Traffics

In this set of experiments, we intended to study the performance impacts of different types of traffic, specifically, local or remote traffic, on the distributed simulation under different network partition schemes.

In order to compare the simulation performance of Genesis and SSFNet, we first did some experiments under SSFNet on the same campus network model. We ran a set of simulations on a Sun UltraSPARC-II 4-CPU machine, and varied the number of processors in these SSFNet parallel simulations setting it to 1, 2, 3 and 4. Ring size of 6 campus networks was used for the 1, 2 and 3 processors simulations, while ring size of 4 campus network was used for 1 and 4 processors simulations. We always assigned adjacent campus networks to the same processor. Thus, by varying the number of processors, we varied the percentage of remote traffic (in this case, traffic exchanged between parallel processors) as well.

Table 5.5 shows the experiment results. The results demonstrate that when

Table 5.5: **SSFNet Parallel Efficiency A**

Network Size (CN)	Partitions	Remote Traffics	Speedup	Distributed Efficiency
6	1	0%	N/A	N/A
6	2	50%	1.78	89%
6	3	75%	1.68	56%
4	4	100%	2.1	53%

the percentage of remote traffic increased, the parallel efficiency of SSFNET decreased. The parallel efficiency dropped to just above 50% when the percentage of remote traffic was close to 100%. Because all links between two campus networks were the same, the lookaheads were the same when the partitioning were different in these simulations. However, larger amount of events (packets) needed to be exchanged and synchronized among processors introduced higher overheads in the parallel simulation.

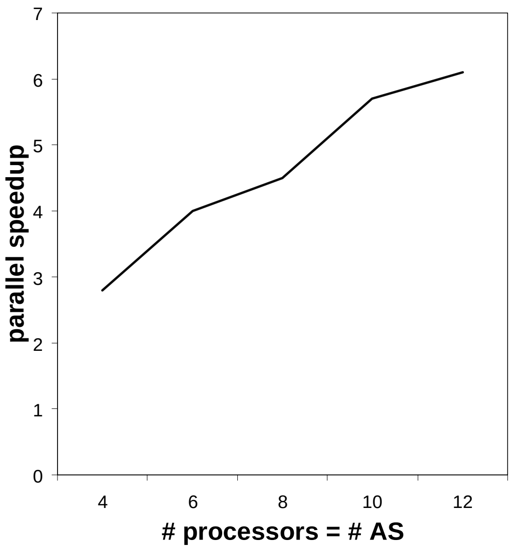

Figure 5.8: SSFNet Parallel Simulation Speedups

The parallel simulation performance of SSFNet was also reported in [68], where experiments were done on Sun enterprise servers with up to 12 processors. A similar ring of campus network model was used while the size of an individual campus network was smaller. We included the reported results for the simulations in which each campus network exchanged traffic only with its neighboring network in Figure 5.8. We computed the parallel efficiencies of those experiments based on these results and showed them in Table 5.6.

Table 5.6: SSFNet Parallel Efficiency

Network Size (CN)	Partitions	Remote Traffic	Speedup	Distributed Efficiency
4	4	100%	2.7	67.4%
6	6	100%	4.0	66.7%
8	8	100%	4.5	56.2%
10	10	100%	5.7	57.0%
12	12	100%	6.1	50.8%

The results in Table 5.6 shows that when the number of processors increased, the parallel efficiencies dropped not very significantly, from above 60% to about 50%. Because in these simulation, each campus network was simulated by one processor and only exchanged traffics with its neighboring network, the remote traffic percentages were the same as 100%. The volume of remote traffic and the overheads for each processor to handle remote events from other processors were about the same. Also the lookaheads were the same because the link delays between campus network did not change. However, with more parallel processors, the overheads of global synchronization increased. From these results, we observed that in the simulations with high percentage of remote traffic, the parallel efficiency of SSFNet was only about 50% to 60%.

Another set of experiments were done under Genesis. 24 campus networks (CN) were connected in a ring, as described in the previous section. We grouped N adjacent CNs into one Genesis domain, as shown earlier in Figure 5.7. N was varied as we set it to 24, 6, 3, 2 and 1, thus the number of domains was, correspondingly 1, 4, 8, 12 and 24. Because each campus network exchanged traffic only with its adjacent networks, when we changed the grouping scheme, the percentage of remote traffic (traffic exchanged between distributed domains) in the simulation also changed. We compared the results of different grouping schemes with the non-distributed simulation (24 CNs simulated in one simulator) and computed the efficiencies.

The individual campus network (CN) model in these experiments was the same as the one we used in the experiments we reported in [78], however, we set the traffic send-rate at the higher rate of 0.02 second per packet. In addition, we reduced the total simulation time from 400 seconds to 100 seconds, while keeping the traffic time as 60 seconds. As a result, the network was loaded with higher intensity traffic throughout the simulation and the communication overheads during each checkpoint became comparatively smaller.

Table 5.7: Distributed Efficiency in Genesis Distributed Simulation

Full Network Size (CN)	Genesis Domains	Remote Traffic	Total Packet-hop Difference	Speedup	Distributed Efficiency
24	1	0%	N/A	N/A	N/A
24	4	28.6%	3.8%	4.15	104%
24	8	50.0%	3.2%	7.73	96.7%
24	12	66.7%	4.0%	11.0	91.7%
24	24	100%	5.0%	19.8	82.6%

Table 5.7 shows the experiment results. From these results, we observed that when the percentage of remote traffic increased from about 28% to 100%, the distributed efficiency dropped slightly. Even with 100% remote traffic, the distributed efficiency was still above 80%. In addition, the error of the total packet hops was within 5% compared to non-distributed, accurate simulation. Genesis showed better performance in these experiments than SSFNet.

We attributed this advantage of Genesis to its coarse granularity synchronization approach. Unlike conventional event-level synchronizations in which each individual remote packet (event) need to be exchanged and synchronized, Genesis aggregated these simulation data and reduced both the amount of data exchange and frequency of synchronization. As the result, higher percentage of remote traffic did not introduce significant synchronization overheads in Genesis, as they usually did in other conventional simulation systems.

5.4.4 Distributed UDP Flooding in U.S. Backbone Network Model

In order to study the performance of Genesis with large scale, real-world network models, we selected the U.S. backbone network model introduced in [39]. This model was a large scale BGP network topology consisting of 8 national-level ISP networks. The full topology included 9828 backbone routers and 787 BGP speakers.

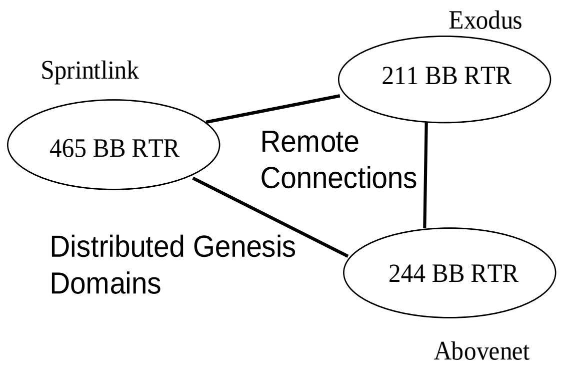

Figure 5.9: Network Model of Three ISP's

Due to our hardware limitation, we selected three ISPs to construct a subset of this U.S. backbone network topology for our experiment. The selected ISP's were Sprintlink with 465 backbone routers and 56 BGP speakers, Exodus with 211 backbone routers and 32 BGP speakers and Abovenet with 244 backbone routers and 39 BGP speakers. We connected them as a clique, as shown in Figure 5.9. We distributively constructed each of these ISP BGP networks into one Genesis domain and assigned each of them to one distributed simulator.

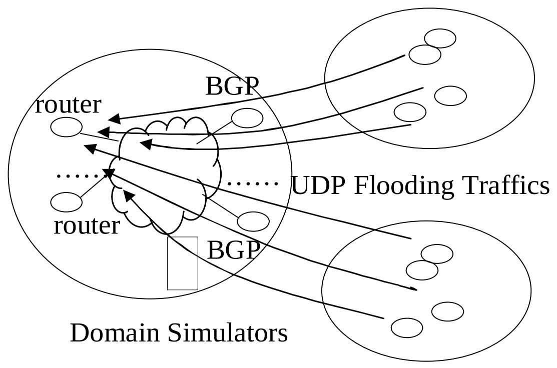

Figure 5.10: Distributed UDP Flooding Simulation

Distributed UDP flooding simulation was used in network security research to study the effectiveness of different protection techniques. Thanks to its distributed traffic pattern and high traffic intensity, it usually introduces high synchronization overheads in parallel and distributed simulations.

We designed our experiment to demonstrate the performance of Genesis in a simulation with burst-out distributed UDP flooding. Initially, each of the three ISP networks had some background traffic, and at simulation time 40 second, we randomly selected 20 distributed hosts in each network to generate high intensity UDP traffics (15Mb/sec) that flooded the other two ISP networks. These burst-out flooding traffics increased the percentage of remote traffic in the network to over 90%. We monitored the packet-hop rate changes during the simulation. Higher packet-hop rate represents higher simulation efficiency, and lower packet-hop rate represents higher synchronization overheads in the simulation.

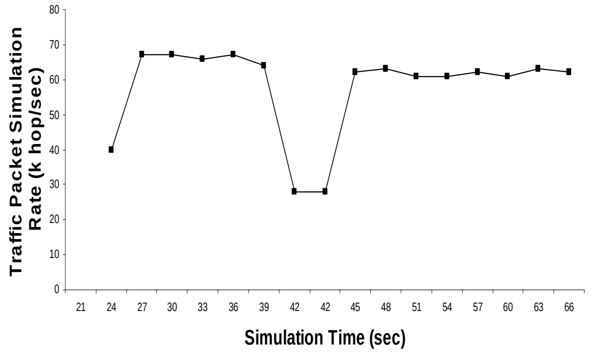

Figure 5.11: Simulation Rate

Figure 5.11 shows the experiment results. We observed that when the distributed UDP flooding bursted out, the packet-hop rate dropped significantly. This was because initially, during the synchronization, Genesis rolled back the simulation for the iteration in which the burst happened. After the Genesis simulation converged on this change in the network, it proceeded with UDP flooding traffic and the packet-hop rate raised back to above 90% of the rate before the flooding. Genesis was able to simulate very high percentage of remote traffic with very little extra synchronization overhead.

5.5 Chapter Summary

In this chapter, we discussed the design and implementation in Genesis to support distributed BGP simulation. With this capability, Genesis facilitated the simulation of large-scale BGP network simulation and made it possible to simulate very large BGP networks on clusters of workstations.

We also demonstrated that Genesis worked efficiently in distributed BGP network simulations. Thanks to its coarse granularity synchronization approach for background traffic which did not require exchanging and synchronizing each individual remote packet, Genesis significantly reduced the overheads in distributed simulations and achieved better distributed efficiency than conventional systems.

Although larger BGP network simulations will require more re-iterations to converge on BGP bursts, our approach supports scalability by distributing the BGP networks as well as the background traffic. In networks in which BGP bursts are not frequent compared to the changes in the background traffic, this approach also assures good run-time performance thanks to coarse granularity synchronization of the background traffic. In the next chapter, we will explain how we go further to distribute the memory usage as well.

With the support of flexible partitioning of BGP networks, and distributed simulation of large scale, real world network models, study of the performance and stability of BGP and defensive techniques against flooding and worm attacks will be directions of our future research.

CHAPTER 6
MEMORY DISTRIBUTED NETWORK SIMULATION

6.1 Introduction

Simulations of large-scale networks require large memory size. This requirement can become a bottleneck of scalability when the size or the complexity of the network increases.

In Chapter 4, we discussed the design and implementation of distributed Genesis simulation. However, that version of Genesis did not distribute network information among domain simulators. Each domain simulator constructs the full network topology and stores all the dynamic information (e.g., routing information) for the whole network during the simulation. To avoid such replication of memory, we developed a new version of Genesis which completely distributed network information among simulators. In such a solution, each domain simulator only needs to construct and store the partition of network assigned to it, none of them needs to construct the full network. As the result, the full network is constructed distributively among domain simulators, and the memory usage for each simulator is reduced significantly.

6.2 Design of Memory Distributed Network Simulation

Memory distribution is particularly challenging in Genesis, because of its special coarse granularity synchronization approach. In Genesis, within one time interval, one domain simulator is working independently of the others, simulating the partial packet flows within or through that domain. If the network information is completely distributed among the domain simulators, then each simulator will have information about only one part of the network. Hence, these simulators cannot simulate global traffic independently because information about some flow sources or destinations, or both will not be there. We should notice the difference here from other event-level synchronization systems. In those systems, to simulate distributed network, each individual event crossing the boundary is forwarded to remote simulators regardless of its “semantic meaning”. Such a parallel simulator does not need to simulate global flows independently, but they must synchronize their execution tightly on event level.

In Genesis solution, each domain uses traffic proxies that work on behalf of their counterparts in the remote domains. Traffic proxies send or receive TCP or UDP data packets as well as acknowledgment packets accordingly to produce feedbacks. To simulate inter-domain flows, partial flows are constructed between local hosts and *proxy hosts*. Thus, in the simulation of one AS domain, the simulator just simulates one part of an inter-domain traffic by using *proxy hosts* and *proxy links*, as shown in Figure 6.1.

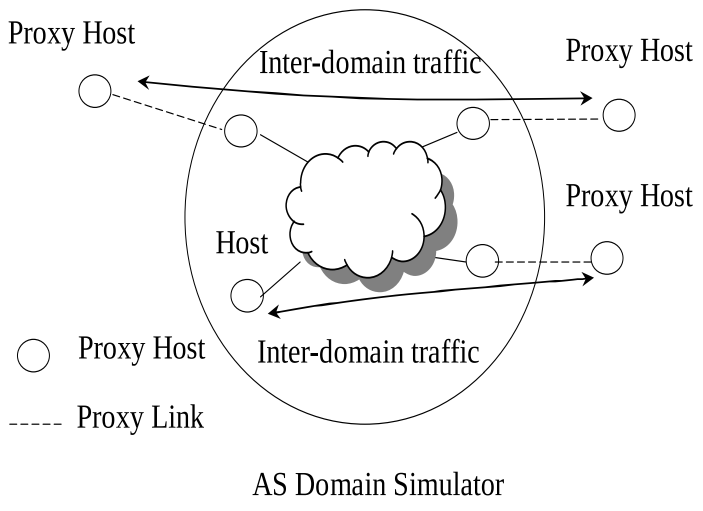

Figure 6.1: Proxy Hosts and Inter-domain Traffic

The actual traffic path between local hosts and remote hosts must be decided by inter-AS routing. For example, inter-AS routing changes can cause remote inbound traffic to enter the current AS domain from different entry points and flow through a different path inside the domain. We developed a method, described below, to construct these remote traffic paths and to automatically adjust them to reflect the current inter-AS routing decision.

To support distributed memory simulation in Genesis, changes were made to both DML definition and SSFNet based implementation.

Global routing information consistency: To compute global routing in separate simulations, each of which has only information about a part of the network, IP address consistency is required to make the routers understand the routing update messages. In addition, we use BGP proxies and traffic proxies to act on behalf of their counterparts. To use routing data, these proxies need to use the IP addresses of their counterparts when they produce traffic packets. We used a global IP address scheme for the whole network, and introduced a mechanism of IP address mapping, which translates local addresses to and from global addresses used in our BGP update messages. In our global IP address scheme, domains are assigned different IP address blocks to avoid address conflicts among domain simulators. Inter-domain routing information is stored based on these global addresses. Each proxy host stores the IP address of its counterpart host which has a global IP address. When packets are sent from proxy hosts, the IP addresses in the packet headers would be replaced with corresponding global IP addresses. In this way, the addresses in these packets are consistent with the routing information and can be correctly routed to the destinations.

***Remote host*, *traffic* and *link*:** Those definitions were added to the current DML definitions for SSFNet [67]. *Remote host* defines the traffic host (source or sink) which is not within the current simulating domain, and specifies the global IP address for this proxy. *Remote traffic* pattern extends SSFNet to allow the definition of a traffic which will use proxy IP address instead of its own local IP address. *Remote link* is defined to connect the *remote host* to the current domain, and it is implemented as a Genesis *proxy link* which can adjust its link delay and applied packet drop rates during the simulation.

Remote traffic path construction: The difficult part of remote traffic path construction was to decide how to connect *proxy hosts* to the current AS domain. Changes in inter-AS routing decision might change the entry (exit) point of traffic packets to (from) the domain. Such a change cannot be determined during the network construction phase. We designed a structure which connected remote traffic hosts to a *proxy switch*, instead of connecting them to any entry point directly, as shown in Figure 6.2. When a packet sent by a *proxy host* reaches the *proxy switch*, the *proxy switch* will lookup an internal mapping from flow id to the current inter-

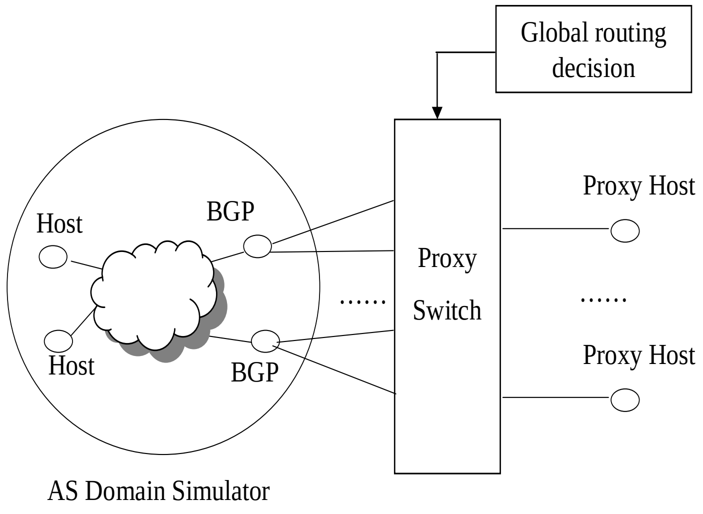

Figure 6.2: Remote Traffic Path Construction with Proxy Switch

AS routing table, and will forward this packet via the correct inbound link to one of the BGP routers on the domain boundary. If the inter-AS routing is changed by some BGP activities later, the *proxy switch* will automatically adjust its internal mapping, and then the packets with the same flow id will be forwarded to the new inbound link.

6.3 Simulation Component Design

This section provides an overview of Genesis extensions to support memory distributed BGP network simulations.

6.3.1 Network Partitioning for Memory Distributed Simulation

In Genesis simulations without memory distribution, network partitioning provides the mechanism to identify the active domain and non-active domains in the full network topology. However, network partitioning for memory distributed simulation is different. Because the full network topology is no longer stored in any of the domain simulators, network partitioning provides the mechanism to construct partial networks for domain simulators. In addition, some auxiliary network information is also stored to coordinate the inter-operation among domain simulators. Genesis User Manual [46] provides the details about the requirements for constructing the network partitions in DML network scripts

Genesis Domain, Active Domain and Proxy Domain

Domains are partitions of the original network. The domain which is assigned to the current domain simulator is referred as the “active domain” of the current domain simulator. The rest of the network outside of the “active domain” is represented in an abstract level, and only the part of outside network which will have network traffic or BGP sessions with the “active domain” need to be represented in the current domain description file. These structures are represented as different kinds of Genesis proxies. All the domains other than the “active domain” are non-active domains in the current domain simulators, and are represented mainly by Genesis proxies.

Each Genesis domain description file will contain one “proxy domain” which does not represent any part of the original network. Instead, the “proxy domain” is one group of Genesis proxy structures used by Genesis itself to coordinate the distributed domain simulators and to manage proxy traffic. The active domain, non-active domains and the proxy domain together are called a “Genesis domain”. If the active domain is BGP AS 1, then the Genesis domain is called the Genesis domain for BGP AS 1. For example, if a network with three BGP Automatic System (AS) domains is partitioned into three Genesis domains, Figure 6.3 shows the structure of the Genesis domain for BGP AS 1.

One Genesis domain description file needs to be constructed from the original network description file for each Genesis domain simulator.

Proxy Domain Structure

Proxy hosts in non-active domains are not directly connected with the active domain. Instead, they are connected with one “proxy switch” router in the “proxy domain”. The proxy domain is a special BGP AS domain constructed specially for Genesis domain simulators. The basic functionality for the proxy switch router is to forward each packet coming out from the active domain to the correct proxy host, and forward each packet coming out from a proxy host to the correct BGP router in the active domain according to the current routing configuration of the network. If the inter-AS routing changes, the proxy switch will automatically adjust itself

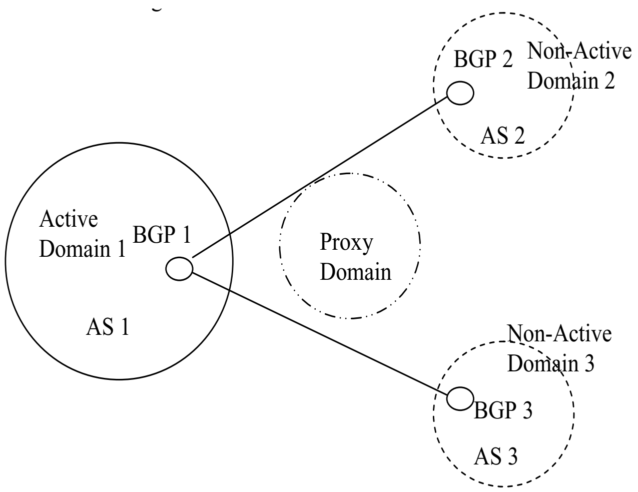

Figure 6.3: Domain Structure for Genesis Domain Simulator 1

to forward packets accordingly. All proxy hosts in non-active domains are directly connected with this proxy router, and all packets coming out from the active domain are immediately forwarded to this proxy router. This router serves as a central "switch" for the packets, so it is called the "proxy switch router". An overview of the structure of a proxy domain is shown in Figure 6.4.

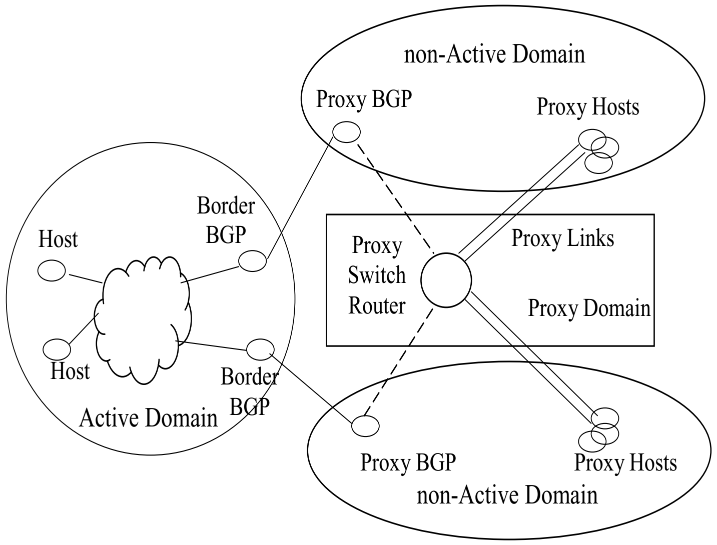

Figure 6.4: Genesis Proxy Domain and Proxy Switch Router

The "proxy domain" is usually assigned an unused number as its BGP AS net id. It has one router defined with mark "proxy switch". This router has one interface defined for each proxy hosts which will be directly connected with it, and each of these interfaces is marked with "proxy interface". In addition to this proxy domain, one link is defined to attach each proxy host to one of the proxy interfaces of the proxy switch router. These proxy links are regular point-to-point links with special marks, which tells the simulator that this is a Genesis proxy link instance. Proxy links have pre-set delay and drop rate, and will be automatically adjusted by Genesis during the simulation.

6.3.2 Packet Forwarding for Remote Traffic

For remote traffic, when one packet going out of the active domain reaches the proxy BGP router outside of the domain, it needs to be forwarded to the proxy switch router, and the proxy switch router needs to forward it further to the corresponding proxy link for this flow. On the other hand, when one packet from a proxy host goes through the proxy link and reaches the proxy switch router, the proxy switch router needs to forward it to the correct NIC to allow it to enter the active domain. This packet forwarding is done in the IP layer of the proxy switch router and the proxy BGP routers.

To forward remote traffic packets correctly, the first step is to identify the position of the packet in the partitioned network. This is done by adding special marks into host/router definitions. Hosts or routers at different positions in the network are marked with different host types. These host types are defined in class *Host*, as shown in Table 6.1.

These marks are automatically set by parsing Genesis DML simulation scripts instrumented with special marks for Genesis [46]. They are stored in the attribute *ghtpye* of class *Host*, so it can be used to identify the role of a host/router in the Genesis simulation. For example, the proxy switch router has its *ghtpye* set to *PROXY_SWITCH*.

After identifying the host type, the IP layer operations will forward a remote packet correspondingly. When one remote packet is to be pushed down to the link

Table 6.1: Host Types in Genesis

```
// host inside of the active domain
public static final int NORMAL_HOST                     = 0;
// host at the border of the active domain
public static final int BORDER_HOST                     = 1;
// proxy server outside of the active domain
public static final int PROXY_SERVER                    = 2;
// proxy client outside of the active domain
public static final int PROXY_CLIENT                    = 3;
// proxy bgp router connected to the active domain
public static final int PROXY_BGP                       = 4;
// proxy ospf router connected to the active domain
public static final int PROXY_ROUTER                    = 5;
// proxy switch router in the proxy domain
public static final int PROXY_SWITCH                    = 6;
// UDP and TCP servers representing transit traffic
public static final int DYN_SERVER_UDP_TCP              = 7;
// udp client representing transit traffic
public static final int DYN_CLIENT_UDP                  = 8;
// TCP client representing transit traffic
public static final int DYN_CLIENT_TCP                  = 9;
```

layer for transmission, the method *push* of class *IP* is invoked, and the related *ghtype* is checked. If the packet is in the IP layer of the proxy switch router, this means that the packet has just come out of the proxy link and needs to be pushed into the corresponding entry NIC to enter the active domain. The entry NIC instance for this flow is retrieved using the current packet's flow id, and this packet is pushed into that NIC.

If the packet is in the IP layer of a proxy BGP router with mark *PROXY_BGP*, this packet needs to be pushed into the corresponding proxy NIC in the proxy switch router. The mapping between flow IDs and proxy NICs is stored in a Genesis information base. The proxy NIC instance for this flow is retrieved and the current packet is directly pushed into that NIC.

6.3.3 Traffic Data Collection and Update

Per flow traffic data (e.g. packet path delay, packet sends and drops) are collected during the simulation. To record the packet trace information during the simulation, the IP header is extended with attributes for Genesis, including the fields shown in Table 6.2.

Gfid stores the Genesis flow id; *gflag* identifies different types of remote/local traffic; *gdata* is 1 for UDP or TCP traffic packets with data payloads, and is

Table 6.2: New Fields in IP Header

public int gfid	= 0;
public int gflag	= 0;
public int gdata	= 0;
public long sent_time	= 0;
public long entered_time	= 0;
public long left_time	= 0;
public long received_time	= 0;

0 for other packets; other attributes record the send-from-source and receive-at-destination time, and the time one packet enters or leaves the active domain.

Sent_time is set when the packet just comes out from the traffic source; *received_time* is set when the packet is to be received by the traffic destination; *entered_time* is set when the packet just comes across a link from a router outside of the active domain and reaches a border router inside of the active domain; the *left_time* is set when the packet reaches a border router and will be pushed into a NIC marked as "border NIC" (a NIC in the active domain connects to a NIC outside of the active domain). At the same time, the counters related to the sending, receiving, entering and leaving for each flow are updated. Based on these recorded timing data, the delays for each packet from the sending to receiving and from the entering to leaving are computed as well.

The other information collected in push is the "exit point" of a remote flow. When a packet goes out of the current active domain and reaches a proxy BGP router, the NIC of the proxy BGP router from which the packet coming out is the "exit point" of this flow for the current active domain. This "exit point" will be the "entry point" for the domain which contains the proxy BGP router. Thus, the interface id of the NIC "exit point" is recorded. During the checkpoint, this "exit point" information will be forwarded to the domain simulator which simulates the active domain containing this proxy BGP router, where this "exit" NIC will be set to the "entry" NIC for this remote flow. In addition, this "exit" NIC will also be set to be the "entry" NIC for the feedback flow.

All these traffic data are stored in a global information base for the domain simulator. Genesis provides the functionalities to manage this information base, which includes:

- Store flow identification, flags, traffic types (UDP/TCP), traffic source and destination, etc.;
- Store per flow traffic data and statistics data. Per flow packet delay and drop information and the computed statistics data are stored in different arrays;
- Store per flow traffic path information. For each flow, the information about the entry NIC and proxy links needed by Genesis simulation is stored in arrays. For each flow, the information of "cross links" (the links connect the active domain and non-active domains) on its traffic path is also stored in arrays. It stores the mapping between traffic flow IDs and entry NICs, which decides the packet forwarding in proxy switch router;
- Store some other statistics information, like packet hop counts;
- Statistics computation over stored data;
- Access methods to retrieve and update stored data.

6.3.4 Transit Traffic Simulation

In Genesis, a transit traffic to an active domain is a traffic which both the source node and the destination node are outside of the current active domain. Unlike the remote traffic which has one end of traffic, either the source or destination, inside of the active domain, transit traffic cannot be identified in advance for a domain simulator. Transit traffic for one active domain is determined by the inter-domain routing, and it can change during the simulation when inter-domain routing changes. This requires domain simulators to simulate transit traffic dynamically.

In the Genesis simulation with full network topology, when a domain simulator needs to add a transit traffic during the simulation, it only needs to activate the *proxy source* nodes for that traffic to create the transit flow, because all the nodes in the network have been constructed before the simulation starts. However, in memory distributed simulation, only one part of the network is constructed in a domain simulator, thus the source nodes for a transit traffic do not exist in advance. The solution is that Genesis creates a pair of special server and client hosts in it proxy domain to handle transit flows dynamically. These special server and client can dynamically create UDP or TCP flows using specified source and destination IP addresses.

In Genesis, when one domain simulator received traffic statistics data for a network flow from its peer simulators, if it identifies this flow as a new transit flow, it will create a client request packet for this flow and sends it to a pre-designated UDP or TCP server. The receiving server will then create a new flow to represent this transit flow. At the same time, this domain simulator also uses the received flow information related to this transit flow to set up the proxy links and entry/exit points for this flow. When this transit flow expires, the domain simulator will remove it from its transit flow list.

6.4 Experiments

6.4.1 Distributed Simulation of Campus Network Model

6.4.1.1 Simulation Model

To test the performance and scalability of BGP simulations and memory distributed simulations in Genesis, we use a modified version of the Campus Network model defined by the DARPA NMS community [56], as we introduced in Chapter 5. We placed the figure of one Campus Network here again for reference.

We used different send-intervals of 0.1, 0.05 and 0.02 second to vary the traffic intensities, and used different numbers of nodes (AS domains) to vary the size of the network. Each simulation was run for 400 seconds of the simulated time.

All tests were run on up to 30 processors on Sun 10 Ultrasparc workstations, which were interconnected by a 100Mbit Ethernet. One of these workstations had 1G large memory, and each of the others had at least 256M dedicated memory. In the simulations under distributed Genesis, the number of processors used was equal to the number of campus networks.

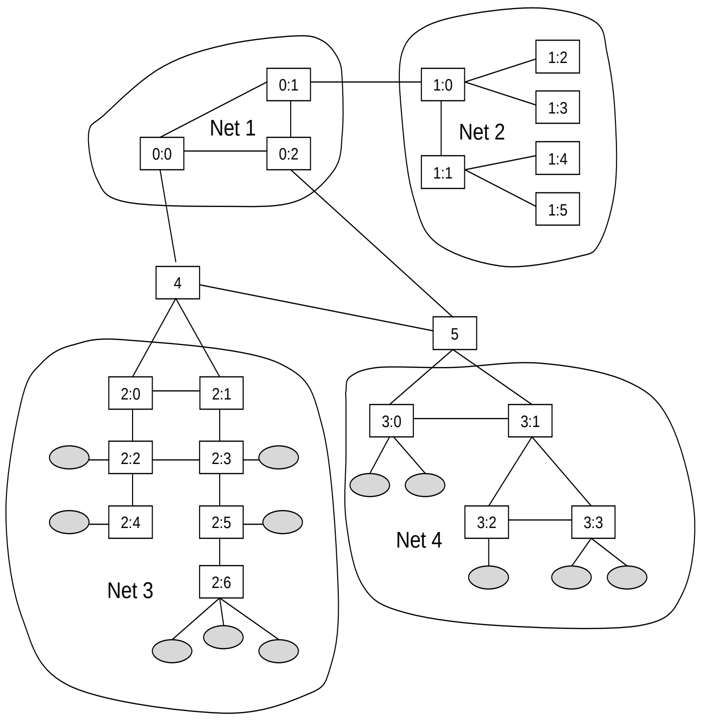

Figure 6.5: One Campus Network

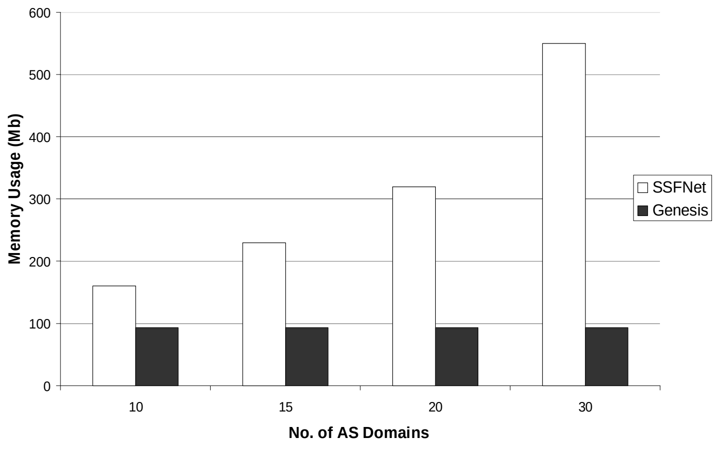

Figure 6.6: Memory Usage of SSFNet and Genesis for Simulating Different Sizes of BGP Networks. Memory Sizes of Genesis Shown above are the Requirements for Single AS Simulator

6.4.1.2 Performance of Simulations on Distributed Memory

Genesis distributively constructs and simulates BGP routers in AS domain simulators. To measure scalability of this solution in terms of network size, we simulated BGP networks of 10, 15, 20 and 30 AS domains, each run by a Sun 10 Ultrasparc workstation with 256Mb of memory. As shown in Figure 6.6, when the number of AS domains increases, unlike SSFNet, the memory usage of one Genesis AS simulator does not increase significantly. As a result, by utilizing more computers with smaller memories, we are able to simulate much larger networks.

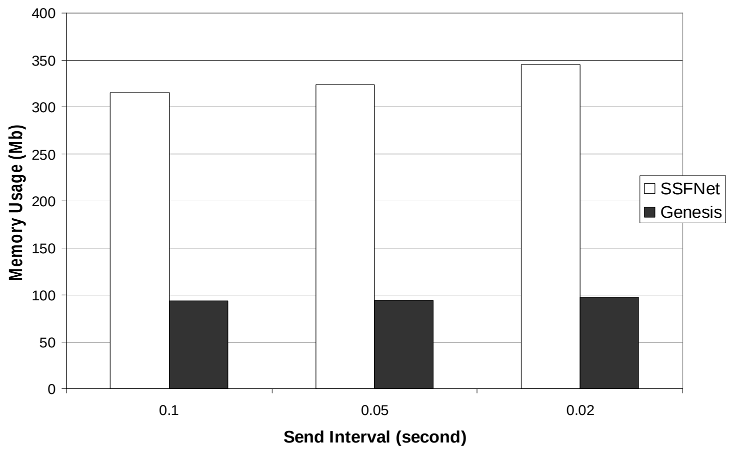

Figure 6.7: Memory Usage of SSFNet and Genesis for 20-AS BGP Network Simulations with Different Send-rates

Memory usage of simulation is related not only to the static network size, but also to the traffic intensity. We increased the traffic intensity by reducing the traffic send-interval from 0.1 to 0.05 and 0.02 second. As shown in Figure 6.7, although we did not observe very big changes in memory usage in SSFNet on this campus network model, Genesis showed even smaller increase in memory size with the same changes in traffic (thanks to its smaller base memory size).

As we have shown, Genesis achieved execution speedups thanks to its coarse granularity synchronization mechanism. In addition, despite the extra overheads introduced by distributing the network, good speedups where achieved for 10, 15, 20 and 30 domain simulators with BGP routers. The Genesis domains were defined by the AS boundaries. Figure 6.8 shows the speedups of simulations for these networks.

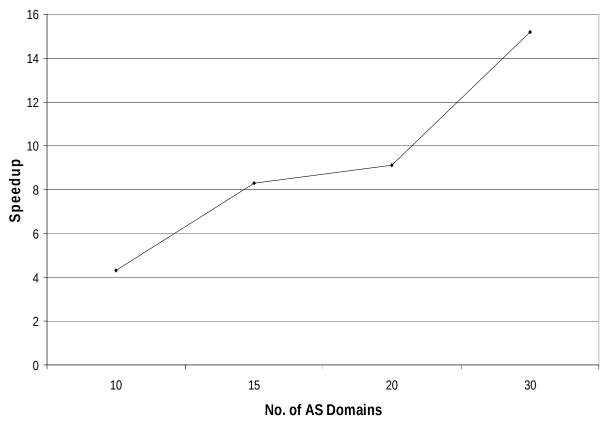

Figure 6.8: Speedup Achieved for Simulations of Different BGP Network Sizes

Figure 6.9 shows that Genesis achieved higher speedups with higher traffic intensities. This is because with higher traffic intensity, more events need to be simulated in a fixed simulation time. Theoretical analysis tells us that sequential simulation time includes terms of order $O(n * \log(n))$, due to sorting event queues. Genesis distributes the simulation among domain simulators, which reduces the number of events needed to be simulated by one simulator, so it can achieve higher speedups when the traffic increases as well as when the network size increases.

To measure the accuracy of the simulation runs, we monitored the per flow end-to-end packet delays and packet drop rates. We compared the results from distributed Genesis with the results from sequential simulations under SSFNet, and calculated the relative errors. Our results showed that for most of the flows, the relative errors of both packet delay and drop rate were within the range from 2% to 10%, while a small number of individual flows had higher relative errors of up to 15% to 20%. Considering the fact that in a simulation with large number of flows, the network condition was mainly determined by the aggregated effects of sets of flows, we calculated the root-mean-square of the relative errors on each set of flows which went through the same domain. These root-mean-squares of relative errors were below 5%, which seems sufficiently close approximation of the sequential simulation for many applications.

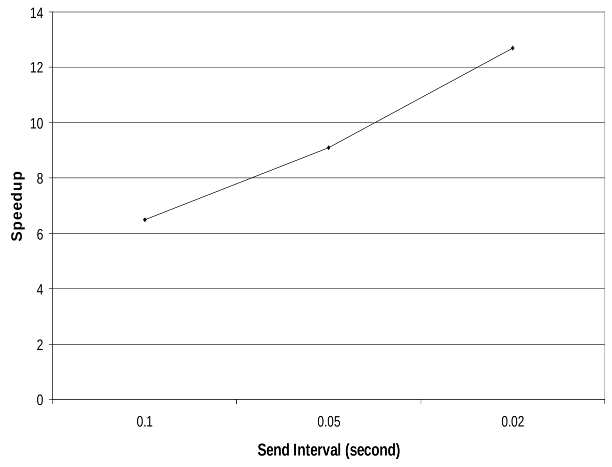

Figure 6.9: Speedup Achieved for 20-AS BGP Network Simulations with Different Send-rates

Simulation results showed that by fully distributing the simulation in Genesis, we gained the scalability of memory size. In addition, the parallel simulation in Genesis still achieved performance improvement in this distributed framework, compared to sequential simulations.

6.4.2 Impacts of On-going BGP Activities

We have shown the synchronization convergence on BGP bursts under Genesis in section 5.4.2. When a BGP update message propagation happens, Genesis re-iterates over the same time interval until the propagation converges. Each re-iteration consumes simulation run time. When BGP update message propagations happen periodically during the simulation, the additional run time required by these re-iterations will decrease the speedups achieved by utilizing parallel simulation. An interesting question is how such on-going BGP activities would affect the simulation performance.

To investigate this question, we introduced BGP session crashes into our experiments. The simulation time was fixed at 400 seconds for 20 AS's and, correspondingly 20 Sun 10 Ultrasparc workstations. The BGP session between two neighboring AS domains, campus network 3 and 4, crashed every D seconds, each time staying down for $D/2$ seconds, and then coming back and staying alive for another $D/2$ seconds.

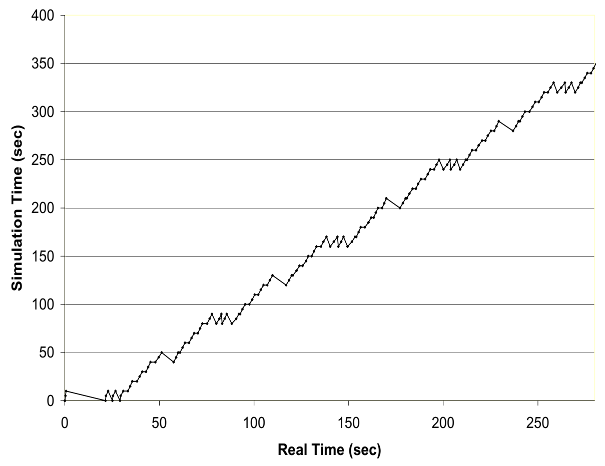

Figure 6.10: Impacts of BGP Crashes on Simulation Progress

As expected, in the simulation time intervals in which the specified BGP session went down, BGP update messages were propagated causing the broken routes to be withdrawn and back up routes being set up. Accordingly, data packet flows also changed and used the new routes. When that BGP session came back again, BGP update messages propagated again and re-established the broken routes. In either case, the relevant time interval had to be re-iterated again and again until it converged. So these intervals were "slow-down" periods. The time intervals with no BGP propagations were "speed-up" periods thanks to parallel simulation mechanism used by Genesis, as discussed in the previous section. Figure 6.10 shows the simulation progress with BGP crashes: periodical BGP crashes cause rollbacks in simulation time, which are "slow-down" periods; after these BGP propagations converge, the simulation can progress to the next time interval. As a result, the proportion of the "slow-down" time in the whole simulation time affects the overall speedup: the less frequently the crashes happen, the greater the speedup that can be achieved. When the frequency of BGP crashes is not high, these "slow-down" periods will not significantly slow down the progress of the simulation.

On the other hand, the length of the time interval also affects the total re-iteration time. Ideally, smaller the length of the time interval, shorter the total re-iteration time. But there are two other factors which benefit from longer intervals. First, small interval length increases the synchronization and checkpointing overheads between intervals and can overwhelm the simulation speedup. Second, if the interval length is too small to cover the BGP propagation period, then the next time interval will need to be re-iterated, in addition to the current one.

To study the impact on performance of different crash frequencies and simulation interval lengths, we varied the value of D from 80 to 60, 40 and then 20 seconds, and also used different lengths of iteration time intervals for Genesis checkpointing, denoted as T, which was set to 20, 10 and 5 seconds. Figure 6.11 shows the speedups achieved in these experiments.

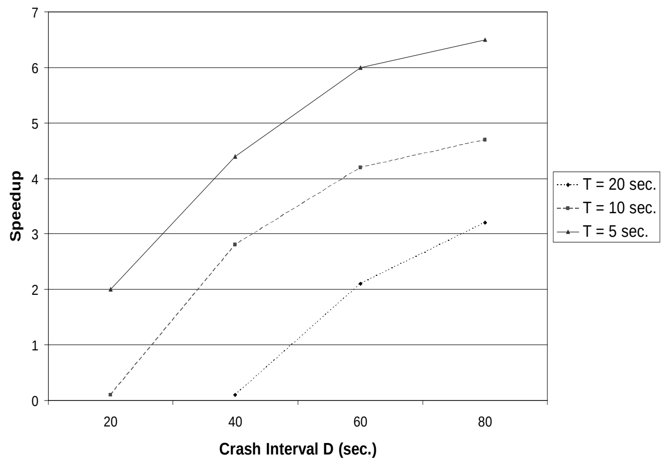

Figure 6.11: Speedup Achieved with BGP Crashes Run on 20 Processors. D Denotes Crash Interval Length. T Stands for Checkpoint Interval Length

In our experiments, we were able to reduce the iteration time interval length to 5 seconds, as we observed that the BGP propagation in our experiment scenario was around 3 seconds. As shown in Figure 6.11, we achieved significant speedups for crash intervals greater than 40 seconds. Besides the crash frequency, the iteration length also played an important role in the performance. When using big iteration interval length of 20 seconds, Genesis failed to produce any speedup with short crash interval of 40 seconds. These results indicate that a method to automatically decide the optimal iteration length for a given simulation scenario could be a valuable future extension that can improve the overall performance of the system.

6.5 Chapter Summary

Memory distribution introduced some extra overheads into Genesis simulation. However, thanks to its coarse granularity synchronization approach, Genesis was able to achieve significant execution speedup over conventional sequential simulations.

With the support of memory distributed simulation, Genesis significantly reduced the memory usage in each domain simulator. Thanks to this solution, Genesis was able to simulate large-scale networks using a cluster of computers with smaller dedicated memory (compared to the memory size required by shared memory-based SSFNet simulating the same network). This will facilitate research work using large-scale network simulations.

CHAPTER 7
GENESIS SIMULATION PARAMETERS

7.1 Introduction

In Genesis, there are several parameters which can be customized in simulation scripts. These parameters include the length of the simulation intervals (checkpoint intervals), re-iteration convergence conditions and computation parameters for statistics calculation. We observed that different configurations of these parameters affected the performance of Genesis simulation. They affected the simulation execution time and the accuracy of the simulation results as well, and these effects would be different for different types of simulations. In this chapter, we discuss these Genesis parameters in details, and demonstrate their effects in simulation experiments.

7.2 Network Model

The simulation network model used in this section was the same campus network model described before. The size of the model was a ring of 4 campus networks. Each campus network had 504 UDP or TCP traffic flows connecting to its neighboring campus networks in the ring. For each set of experiments, we simulated this network topology with SSFNet sequential simulation first, and then partitioned the network into distributed domains and simulated it with 4 Genesis distributed simulators (one campus network per simulator). We varied the values of Genesis parameters for different simulations and compared the results with that of corresponding SSFNet sequential simulation.

To evaluate the performance of Genesis against sequential SSFNet, we compared the execution time of the simulations to calculate speedups, and compared the accuracy of simulation results based on average packet end-to-end delay, traffic data throughput and average packet drop rate. We also measured and compared the total packet hop counts in simulations. Total packet hop count and hop rate reflect how much work has been done during a network simulation, so they are also used to measure the correctness as well as the efficiency of a simulation.

7.3 Simulation of UDP Traffic

In the first set of experiments, we simulated UDP traffic in the network with low traffic intensity. We used constant bit rate UDP traffic source and set the UDP traffic send rate at 20 packets per second with packet size of 10K bytes.

We used simple convergence condition for this set of experiments: we compared the recorded average delays (as explained early in this thesis), and if the changes between iterations exceeded a *threshold* of 5%, then the simulator rolled back to re-iterate the last simulation interval.

Convergence Condition A:

```
$Error_N$    $= (Delay_N - Delay_{N-1})/Delay_{N-1}$
IF           $Error_N > threshold$
THEN
             roll back interval $N$ and re-iterate
ELSE
             continue to interval $N+1$
```

where *threshold* was set to 5%.

We also used simple statistics computation that recorded average delay and drop rate for the current interval, $Delay_N$ and $Droprate_N$, are directly applied to the next interval, as

$$Delay_{N+1} = Delay_N \tag{7.1}$$

$$DropRate_{N+1} = DropRate_N \tag{7.2}$$

We varied the length of simulation intervals as 5, 10 and 20 seconds, and compared the results with SSFNet sequential simulation. The results are shown in Table 7.1.

In this set of experiments, no UDP packet was dropped during the simulation because the traffic intensity was low and no congestion happened. From the results, we observed that Genesis achieved very good accuracy for all three different

Table 7.1: UDP Traffic Simulation with Packet Rate of 20 Pkt/sec

	SSFNet	Genesis	Genesis	Genesis
Interval(sec)		5	10	20
Run-time(sec)	960	250	260	286
speedup		3.8	3.7	3.4
Throughput(Mb/sec)	40.36	40.36	40.36	40.36
difference		0%	0%	0%
Drop Rate	0%	0%	0%	0%
difference		0%	0%	0%
Delay(sec)	0.2384	0.2372	0.2357	0.2347
difference		0.5%	1.1%	1.6%
Hop Count(Mhops)	20.97	21.95	22.00	22.90
difference		4.67%	4.92%	9.20%

interval lengths. In addition, when shorter interval length was used, the accuracy of the result was slightly better. This was because with shorter length interval, Genesis domain simulators updated their proxy links more frequently with statistics data collected by its neighboring domains, which represented the current network condition better.

We should also notice that the distributed efficiencies with 4 domains in these experiments were lower than those we reported in Chapter 6. This was because we used smaller domain size which consisted of only one Campus Network, instead of grouping multiple Campus Networks into one domain. Smaller domain size meant less simulation work for each domain simulator during the same interval, thus the synchronization and checkpointing overheads became comparatively bigger. We chose smaller domain size in these experiments because it facilitated our running of large number of simulations, and our purpose in this chapter was to study the effects of Genesis parameters.

To introduce traffic congestion into the network and study the performance of Genesis in the network condition that packets were dropped, we increased the UDP packet rate to 50 packets per seconds. Under this traffic intensity, the maximum traffic load on the inter-campus links increased to slightly over 2.0 Gb/second, while the capacity of inter-campus links in the model was 2.0 Gb/second. Congestion would happen and packets would be dropped. The simulation results for this set of experiments are shown in Table 7.2.

Table 7.2: UDP Traffic Simulation with Packet Rate of 50 Pkt/sec

	SSFNet	Genesis	Genesis	Genesis
Interval(sec)		5	10	20
Run-time(sec)	950	285	320	410
speedup		3.3	3.0	2.3
Throughput(Mb/sec)	39.87	39.84	39.78	39.77
difference		0.10%	0.21%	0.25%
Drop Rate	1.20%	1.30%	1.42%	1.46%
difference		8.33%	18.3%	21.6%
Delay(sec)	0.2390	0.2364	0.2335	0.2299
difference		1.1%	2.3%	3.8%
Hop Count(Mhops)	20.75	22.67	22.69	22.72
difference		9.22%	9.29%	9.90%

Results in Table 7.2 shows that Genesis performed well in this slightly congested network as well. Similar to the results shown in Table 7.1, simulation with the smallest interval length, 5 seconds, produced the most accurate results among the three different interval lengths. In this congested network and with Genesis interval length of 5 seconds, the errors of both traffic throughput and average end-to-end delay were under 1 percent and the errors of packet drop rate and total packet hop count were under 10 percent. The execution speedup was 3.3 for interval length of 5 seconds, thus the distributed efficiency of Genesis with 4 processors was 82.5%.

We again increased the UDP packet send rate to 100 packet/second to make the network highly congested. The simulation results of SSFNet and Genesis are shown in Table 7.3.

In the highly congested network, about half of the packets sent were dropped. Genesis performed consistently in such a congested network condition and achieved speedup of 2.6 with 5 seconds interval length, and the error in traffic statistics was under 5%. We also observed that the total packet hop count in this case was much smaller than those of the previous cases, thanks to the increased packet drop rate. Consequently, the speedup in this case was smaller than those of the previous

Table 7.3: UDP Traffic Simulation with Packet Rate of 100 Pkt/sec

	SSFNet	Genesis	Genesis	Genesis
Interval(sec)		5	10	20
Run-time(sec)	564	215	285	300
speedup		2.6	2.0	1.9
Throughput(Mb/sec)	20.99	20.49	20.47	20.32
difference		2.41%	2.46%	3.2%
Drop Rate	48.0%	49.2%	49.3%	49.6%
difference		2.50%	2.71%	3.33%
Delay(sec)	0.2395	0.2349	0.2309	0.2299
difference		1.9%	3.6%	4.0%
Hop Count(Mhops)	12.32	13.01	13.03	13.07
difference		5.59%	5.84%	6.09%

cases as well. The reason was that when less packet hops were simulated during one interval, less wall-time would be spent on this interval, and the overheads introduced by checkpointing between intervals became comparatively bigger and decreased the distributed efficiency.

In all of the three cases above, we observed that the simulation with the smallest interval length had the best performance. One interesting question would be, is the smaller the simulation interval, the better the Genesis performance? To find out the answer, we further reduced the interval length to 2 seconds for the simulation with UDP packet rate of 50 packets/second, and we show the changes in simulation accuracy and distributed efficiency with different interval lengths in Figure 7.1.

Figure 7.1 shows that when shorter interval length was used, the simulation result would be better. However, distributed efficiency decreased when the interval was too short. When the interval length was 2 seconds, the speedup of Genesis with 4 processors dropped to 2.64, thus the distributed efficiency was 66%, which was lower than the 82.5% of the simulation with interval length of 5 seconds.

Although shorter simulation interval length improved the accuracy of simulation results, the effect of interval length on execution time was two-fold. With longer interval length, a simulation would be divided into less iterations with less check-

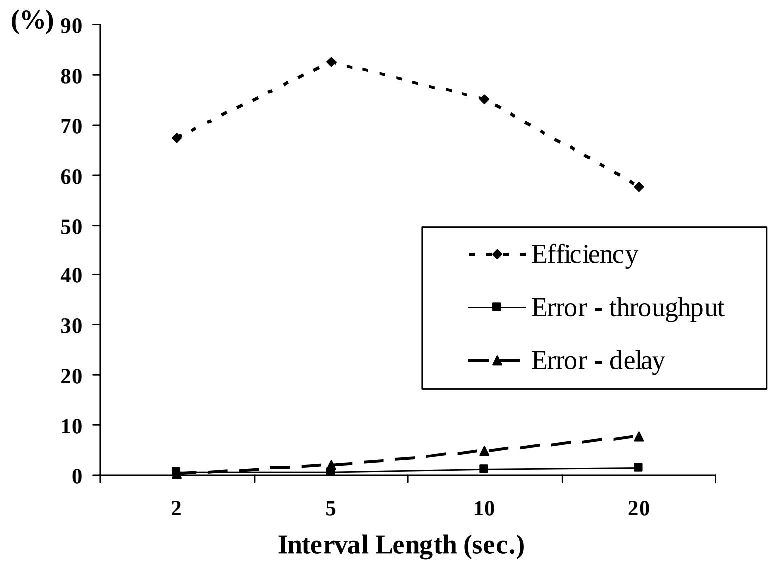

Figure 7.1: Simulation Accuracy and Distributed Efficiency with Different Interval Lengths

points, thus the overheads introduced by checkpointing would be smaller. However, when rollback happened, a simulator will roll back (go back to the last checkpoint) more with longer interval length, thus requires more time to re-iterate and converge. On the other hand, with shorter interval length, the simulation will converge faster with shorter re-iterations, but more checkpoints will introduce more overheads.

The optimized interval length would be different for different simulations. Our simulation experiments showed that the length of 5 seconds produced significant execution speedups while the accuracy of the results were above 90%.

7.4 Simulation of TCP Traffic

7.4.1 Using the Same Parameters as UDP Simulation

In the previous section, we studied the results of UDP traffic simulation under Genesis. Constant bit rate UDP traffic simulation is a simpler case than TCP traffic simulation because the traffic source does not response to the changes of the network condition. In the three different network conditions (non-congested, slightly congested and highly congested) in the previous section, the same amount of UDP packets were poured into the network. However, TCP is a more sophisticated feedback-based protocol that a traffic source will response to the network condition and adjusts its data send rate. In this section, we study the performance of Genesis and the effects of Genesis parameters in TCP traffic simulation.

Unlike UDP simulations, when simulating TCP, we can not control the network condition by simply changing the packet send rate. TCP has its own congestion control mechanism. However, the behavior of TCP is sensitive to the size of buffers in the interfaces of routers. Different interface buffer sizes will affect both TCP throughput and TCP packet drop rate. We set up 3 different scenarios as “large buffer”, “median buffer” and “small buffer”, in which the buffer size of border router interfaces was set to 200 MSS(Maximum Segment Size), 10 MSS and 5 MSS, respectively, where the MSS was set to 1K bytes. With these selected buffer sizes, TCP packet drop rates varied from 0% to 6% in SSFNet sequential simulations. For each scenario, we compared the simulation results from Genesis with those from SSFNet.

To demonstrate the difference between UDP and TCP simulation, we first used the same convergence condition and statistics computation as we used for UDP simulation in the previous section.

Table 7.4: TCP Traffic Simulation with Large Interface Buffer Size

	SSFNet	Genesis	Genesis
Interval(sec)		5	10
Run-time(sec)	1336	483	533
speedup		2.7	2.3
Throughput(Mb/sec)	8.092	8.093	8.094
difference		0.01%	0.02%
Drop Rate	0%	0%	0%
difference		0%	0%
Delay(sec)	0.2371	0.2341	0.2366
difference		0.1%	0.2%
Hop Count(Mhops)	42.05	46.10	46.10
difference		9.63%	9.63%

Table 7.4, 7.5 and 7.6 show the simulation results for the three scenarios with different buffer sizes. From these results, we observed that i) in the scenario

Table 7.5: TCP Traffic Simulation with Median Interface Buffer Size

	SSFNet	Genesis	Genesis
Interval(sec)		5	10
Run-time(sec)	1185	413	420
speedup		2.9	2.8
Throughput(Mb/sec)	7.74	6.67	5.95
difference		13.8%	23.1%
Drop Rate	0.47%	0.58%	0.7%
difference		23.4%	48.9%
Delay(sec)	0.2371	0.2366	0.2340
difference		0.2%	1.6%
Hop Count(Mhops)	40.44	38.40	34.75
difference		5.04%	14.1%

Table 7.6: TCP Traffic Simulation with Small Interface Buffer Size

	SSFNet	Genesis	Genesis
Interval(sec)		5	10
Run-time(sec)	227	120	105
speedup		1.9	2.2
Throughput(Mb/sec)	1.26	1.10	0.98
difference		12.7%	22.2%
Drop Rate	5.83%	5.36%	4.85%
difference		8.06%	16.8%
Delay(sec)	0.2373	0.2354	0.2338
difference		0.8%	1.5%
Hop Count(Mhops)	6.928	6.398	5.480
difference		7.60%	20.9%

of large buffer size, TCP successfully transmitted all the packets and no packet was dropped. Genesis performed well in that case, errors in the traffic simulation results were small (under 5%); ii) in the scenarios of median and small buffer sizes, some TCP packets were dropped and the throughput decreased. In those cases, errors in Genesis simulation increased significantly. Especially, errors in packet drop rates were large in those cases; iii) Errors in Genesis simulation with 10 seconds interval length were much larger than those of simulation with 5 seconds interval length.

Unlike UDP traffic sources, TCP sources response to packet drops by adjusting its send window and congestion control windows. TCP slow start and congestion avoidance mechanisms work together to control the number of TCP packet sent without receiving acknowledgments. As a result, in a congested TCP network, we will observe that packet send rates and packet drop rates change more frequently than those rates in a congested UDP network. In the scenarios with median and small buffer sizes, we observed that the average end-to-end delay for received packets did not change much from interval to interval, however, TCP packet send rates and drop rates changed more frequently during the simulation, thanks to TCP flow control mechanisms. This had two effects on Genesis simulations: i) using long interval length would produce large errors, because the proxy links were updated too infrequently to reflect the traffic changes. We observed that length of 5 seconds should be used instead of 10 seconds to produce more accurate results. ii) simple convergence condition based only on changes in average delay was no longer enough for TCP simulation. It would produce larger errors in congested TCP networks than in UDP networks, even with the shorter interval length of 5 seconds.

7.4.2 Improving Convergence Condition Tests

To improve the accuracy of simulation results, we combined average packet drop rate with average packet delay into the convergence condition test. The convergence condition was set to

```
Convergence Condition B:
  $Error_{Delay_N}$        $= (Delay_N - Delay_{N-1})/Delay_{N-1}$
  $Error_{DropRate_N}$     $= (DropRate_N - DropRate_{N-1})/DropRate_{N-1}$
  IF                       $Error_{Delay_N} > threshold$
  OR                       $Error_{DropRate_N} > threshold$
  THEN
                           roll back interval $N$ and re-iterate
  ELSE
                           continue to interval $N+1$
where $threshold$ was set to 5%.
```

We simulated the median and small buffer size scenarios with convergence condition B and interval length of 5 seconds. We compared these results with those using convergence condition A, and showed them in Table 7.7 and Table 7.8.

Table 7.7: Simulation with Different Convergence Conditions and Median Buffer Size

	SSFNet	Convergence Condition A	Convergence Condition B
Run-time(sec)	1185	413	494
speedup		2.9	2.4
Throughput(Mb/sec)	7.74	6.67	7.35
difference		13.8%	5.04%
Drop Rate	0.47%	0.58%	0.49%
difference		23.4%	4.26%
Delay(sec)	0.2371	0.2366	0.2366
difference		0.2%	0.2%
Hop Count(Mhops)	40.44	38.40	39.56
difference		5.04%	2.18%

These results show that by combining both average delay and packet drop rate changes into convergence condition test, it improved the accuracy of the results. This was because once the changes in packet drop rates was included in convergence condition test, if TCP packet drop rate changes exceeded the *threshold*, then those intervals would be rolled back and re-iterated. As a result, proxy links had been updated more frequently to reflect those changes. These improved the accuracy of simulation results to about 90% to 95% of those results from SSFNet.

7.4.3 Improving Statistics Computation

However, combined condition tests introduced more rollbacks which also slowed down the simulation. For the small buffer scenario, with convergence condition B, the distributed efficiency dropped to 35%. We analyze how Genesis updates the proxy links in the case that rollbacks are required. Let's assume that interval N will be rolled back; at the beginning of interval N, each domain simulator updates proxy links with statistics data collected from other domains at the end of interval $N-1$, $staticstic_{N-1}$, to represent the network condition in other domains. At the end of interval N, the simulator collects $staticstic_{N}$ from other domains again, and because the difference between $statistics_{N}$ and $statistics_{N-1}$ exceeds the *threshold*,

Table 7.8: Simulation with Different Convergence Conditions and Small Buffer Size

	SSFNet	Convergence Condition A	Convergence Condition B
Run-time(sec)	227	120	161
speedup		1.9	1.4
Throughput(Mb/sec)	1.26	1.10	1.18
difference		12.7%	7.14%
Drop Rate	5.83%	5.36%	5.49%
difference		8.06%	5.83%
Delay(sec)	0.2373	0.2354	0.2356
difference		0.8%	0.7%
Hop Count(Mhops)	6.928	6.398	6.604
difference		7.60%	4.70%

it rolls back to the beginning of interval N. Then it applies $statistics_N$ to the proxy links to represent other domains, and replaces saved $staticstics_{N-1}$ with the current $statistics_N$. This process will continue until the difference between $statistics_N$ and $statistics_{N-1}$ is not greater than the *threshold*. Both $statistics_{N-1}$ and $statistics_N$ are computed on the boundaries of intervals, neither of them represents the accurate changes during the interval and this slows down the convergence. The ideal solution would be to use very small interval length, then the statistics collected on the boundaries of intervals will be closer to the actual values during the interval. However, we have shown that when the interval length was too small, the checkpointing overheads would significantly slow down the simulation and overcame the advantage of coarse granularity synchronization.

Our solution to this was to update proxy links with a value in between of $statistics_{N-1}$ and $statistics_N$. Thus, the statistics computation with equation 7.1 and 7.2 was changed to

$$Delay_{N+1} = \alpha * Delay_N + (1-\alpha) * Delay_{N-1} \tag{7.3}$$

$$DropRate_{N+1} = \alpha * DropRate_N + (1-\alpha) * DropRate_{N-1} \tag{7.4}$$

where $0 \leq \alpha \leq 1$.

Equation 7.1 and 7.2 were the special case of 7.3 and 7.4 with $\alpha = 1$.

In the case that no rollback is required, $Delay_{N+1}$ and $DropRate_{N+1}$ will be applied to the next interval, interval $N+1$; in the case that rollback is required, they will be applied to the re-iteration of interval N. We varied the value of coefficient α for equation 7.3 and 7.4, and simulated the "median buffer" and "small buffer" scenarios with convergence condition B again. The simulation results for these two scenarios are shown in Figure 7.2 and Figure 7.3, respectively.

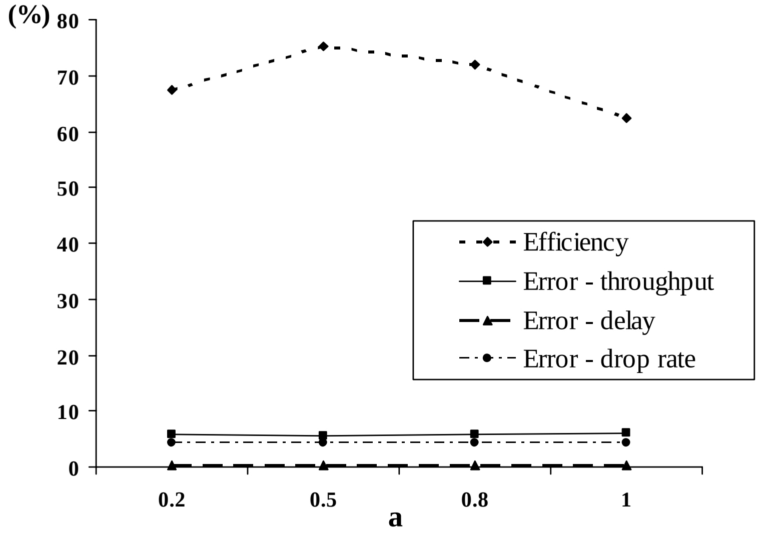

Figure 7.2: Simulation Results with Median Buffer Size and Different α Values

Figure 7.2 showed that when α was 0.5, the distributed efficiency increased to about 74%, which was better than the efficiency of 60% when $\alpha = 1$. In other words, the speedup increased to be about 3.0 with 4 processors. Use of proper α value reduced the number of rollbacks and speeded up the convergence. This speedup was similar to the speedup of 2.9 achieved with convergence condition A, however, the accuracy was improved significantly by applying convergence condition B. We observed similar effects in the "small buffer" scenario. We compared the simulation results and showed them in Figure 7.3. When α was 0.5, the distributed efficiency was improved to about 56% from that of 35% when α was 1. So in the case that more packets are dropped and packet drop rate changes more frequently, a proper

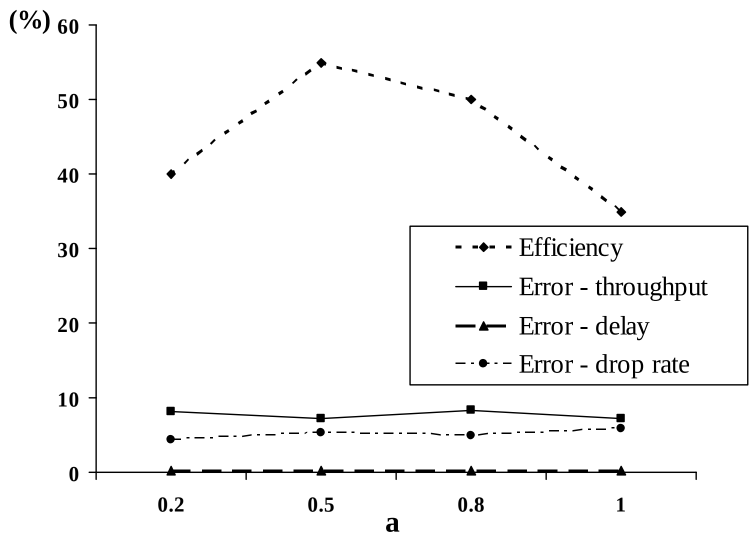

Figure 7.3: Simulation Results with Small Buffer Size and Different α Values

α value can improve the convergence performance more significantly.

7.5 Chapter Summary

In this chapter, we discussed the simulation parameters of Genesis, and their effects on the simulation performance. Our experimental results showed that when simulating UDP traffic, Genesis achieved good performance using convergence condition tests based on packet delays. And with interval length of both 5 seconds and 10 seconds, the accuracy of simulation results was above 90%. In contrast, simulating TCP traffic was more challenging to Genesis. When simulating TCP traffic, interval length of 5 seconds should be used instead of 10 seconds to improve the accuracy of results. In TCP simulation under Genesis, by applying combined convergence condition tests based on both packet delays and drop rates, and updating proxy links with statistics data computed from two consecutive iterations, we achieved significant speedup compared to SSFNet sequential simulation, and the accuracy was above 90%.

CHAPTER 8
CONCLUSIONS AND FUTURE DIRECTIONS

8.1 Conclusions

The need for scalable and efficient network simulators increases with the rapidly growing complexity and dynamics of the Internet. In this thesis we introduced a novel scheme, implemented in Genesis, to support scalable, efficient distributed network simulation.

Our experimental results showed that Genesis worked efficiently in distributed network simulations. Thanks to its coarse granularity synchronization approach which did not require exchanging and synchronizing each individual remote packet, Genesis significantly reduced the overheads in distributed simulations. It achieved significant execution speedups over conventional sequential network simulations.

When simulating UDP traffic, Genesis achieved good performance using convergence condition tests based on packet delays. With interval length of both 5 seconds and 10 seconds, the accuracy of simulation results was above 90%. When simulating TCP traffic, interval length of 5 seconds should be used instead of 10 seconds to improve the accuracy of results. In TCP simulation under Genesis, by applying combined convergence condition tests based on both packet delays and drop rates, and updating proxy links with statistics data computed from two consecutive iterations, we achieved significant speedup compared to SSFNet sequential simulation, and the accuracy was above 90%.

In addition to execution speedups, the advantages of the presented method include fault tolerance, ability to integrate simulations and models in one run and support for truly distributed execution. When one of the participating processes fails, the rest can use the old delay and packet loss data to continue a simulation. When the only information available about a domain are delays across the domain and its outflows, the simulation of the other parts of the networks can directly use these data to perform the simulation.

We also demonstrated the support for distributed BGP simulation in Genesis. With this capability, Genesis facilitated the simulation of large-scale BGP network on clusters of workstations. Thanks to its coarse granularity synchronization approach for background traffic in large scale BGP network simulations, Genesis significantly reduced the overheads and achieved better distributed efficiency than conventional systems.

Although larger BGP network simulations will require more re-iterations to converge on BGP bursts, our approach supports scalability by distributing the BGP networks as well as the background traffic.

We also demonstrated that our system can work efficiently with fully distributed network memory. This design reduces memory and makes scalable the memory size requirement for large-scale network simulations. As a result, Genesis is able to simulate huge networks using limited computer resources. Particularly, the memory size required by BGP network simulation increases very fast when the number of BGP routers and AS domains increases. Simulation of large BGP networks was hindered by the memory size limitation. Genesis offers a new approach to simulating BGP on distributed memory that is scalable both in terms of simulation time and the required memory.

8.2 Future Research Directions

The future research directions fall into two categories: to improve the performance of coarse granularity synchronization and the Genesis system, and to apply Genesis as a simulation tool to solve networking problems.

8.2.1 Network Partitioning

In Genesis, a network is partitioned into domains for distributed simulation. A network partitioning scheme which can minimize the inter-domain traffic flows will improve the simulation performance. This is because each inter-domain traffic flow will require proxy links to be set up in related domains, and changes of the traffic might require rollbacks in the simulation. Minimizing inter-domain traffic can reduce these overheads.

The graph partitioning problem has been studied and applied to many areas, including VLSI design [1], efficient storage of large databases [65] and data mining [34]. The problem was to partition the vertices of a graph into k parts, such that the number of edges connecting vertices in different parts is minimized. Different graph partitioning algorithms have been proposed. To minimize number of cut-edges, Levelized Nested Dissection (LND) algorithm put connected vertices together by starting with a subdomain that contains only a single vertex, and then incrementally growing this subdomain by adding adjacent vertices [27]. Kernighan-Lin [35] algorithm starts from an initial partitioning of a graph, for example, a partitioning produced by LND algorithm or even a random partitioning, and produces refined partitioning. Given a bisection of a graph, Kernighan-Lin algorithm refines the partitioning by swapping two equal-sized subset of vertices, one from each part, to yield the greatest possible reduction in the edge-cut. Multilevel graph partitioning is an approach that consists of three phases: graph coarsening simplifies the graph by collapsing together subset of nodes; initial partitioning is done on coarsened graph; multilevel refinement refines the partitioning recursively to reduce edge-cut [33].

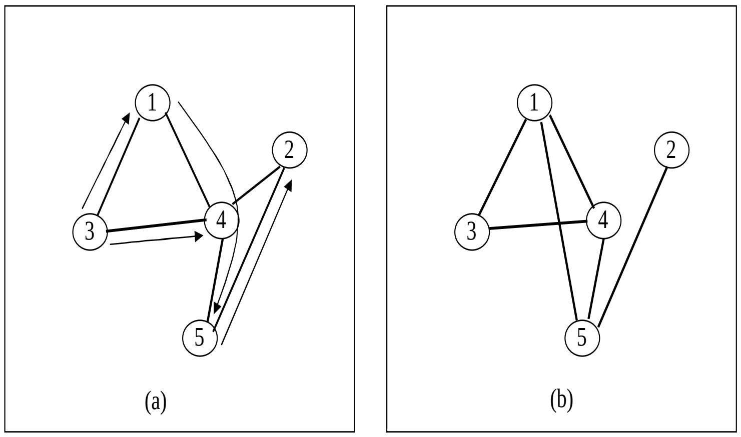

Figure 8.1: Graph Conversion (a) a graph of network connectivity with five nodes and four traffic flows (b) the corresponding graph of flow connectivity

To apply these algorithms to Genesis network partitioning problem, a graph of network connectivity need to be converted to a graph of flow connectivity. Because in Genesis network partitioning, our objective is to minimize the "cut-flows" instead of the cut-links in the network. In the graph of flow connectivity, if there is a flow from node A to node B, directly or indirectly, then there will be an edge connecting from vertex A to vertex B. Figure 8.1 shows an example of this graph conversion.

Because the actual path of a flow is routing, the better we can predict the flow path, the more accurate graph of flow connectivity we can produce.

The questions how to construct the accurate graph of flow connectivity for better network partitioning, and what will be the actual impacts on performance when different partitioning scheme is selected are not covered in the scope of this thesis. However, these will be interesting future research directions.

8.2.2 System Performance Improvement

Several possible directions for improving efficiency of the current Genesis system includes:

- Adaptive selection of time interval length based on a variance of the delay and packet drop rate of the inter-domain traffic.
- Non-constant model of the flow delay, for example using the linear model of the flow delay based on empirical data or using empirically collected delay time distribution should speed up convergence to the fixed point solution.
- Aggregation of inter-domain flows passing through the same border router may improve efficiency by enabling replacement of many individual source proxies by a single aggregate proxy.
- Dividing each checkpoint interval into smaller sub-intervals for statistics data collection may improve the convergence. For example, instead of computing the average delay for the full interval length T, we can compute the average delay for every $T/4$ sub-interval and store them in a vector. Convergence can be improved by selecting proper sub-interval lengths.

8.2.3 Possible Applications

One other direction of the future work in this area is to apply Genesis to more real-world applications, to construct more practical network models and to simulate and analyze them in Genesis.

With the support of flexible partitioning of BGP networks, and distributed simulation of large scale, real world network models, study of the performance and stability of BGP and defensive techniques against flooding and worm attacks will be possible directions of future research.

LITERATURE CITED

[1] Alpert, C.J., and A.B. Kahng. Recent Directions in Netlist Partitioning. *Integration, the VLSI Joural*, 19(1-2):1-81.

[2] Bellenot, S. State Skipping Performance with the Time Warp Operating System. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation (PADS '92)*, pages 53–64. January 1992.

[3] Bhatt, S., R. Fujimoto, A. Ogielski, and K. Perumalla. Parallel Simulation Techniques for Large-Scale Networks. *IEEE Communications Magazine*, 36, 1998.

[4] Bryant, B.E. Simulation of Packet Communication Architecture Computer Systems. MIT-LCS-TR-188, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1977.

[5] Carothers, C., and B. Szymanski. Checkpointing Multithreaded Programs. *Dr. Dobb's Journal*, 15(8):45–60, August 2002.

[6] Carothers, C., K. Perumalla and R. M. Fujimoto. Efficient Parallel Simulation Using Reverse Computation In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, pages 126–135, May 1999.

[7] Carothers, C., K. Perumalla and R. M. Fujimoto. Efficient Parallel Simulation Using Reverse Computation. *ACM Transactions on Modeling and Computer Simulation*, volume 9, number 3, pages 224-253, July 1999.

[8] Carothers, C., and R. M. Fujimoto. Efficient Execution of Time Warp Programs on Heterogeneous, NOW Platforms. *IEEE Transactions on Parallel and Distributed Systems*, pages 299–317, 11(3), March, 2000.

[9] Carothers, D., D. Bauer and S. Pearce. ROSS: A High-Performance, Low Memory, Modular Time Warp System. In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*, pages 53-60, May 2000.

[10] Carothers, D., D. Bauer and S. Pearce. ROSS: A High-Performance, Low Memory, Modular Time Warp System. In *Journal of Parallel and Distributed Systems*, 2002.

[11] Chandy, K.M., and R. Sherman. Space-time and simulation. In *Proceedings of Distributed Simulation*, Society for Computer Simulation, 53–57, 1989.

[12] Chandy, K.M., and J. Misra. Distributed simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September 1979.

[13] Coffman, K. G., and A. M. Odlyzko Internet Growth: Is There a “Moore’s Law” a Data Traffic? Preliminary Version at *http://www.research.att.com/~amo/doc/internet.moore.ps*

[14] Cowie, J., A. Ogielski, and D. Nicol. Modeling the Global Internet. *Computing in Science and Eng.*, vol. 1, no. 1, pp. 42-50, Jan 1999.

[15] Dahmann, J., R. Fujimoto, and R. Weatherly. The Department of Defense High Level Architecture. In *Proceedings of the 1997 Winter Simulation Conference*, pp. 142-149.

[16] Das, S., and R. M. Fujimoto. A Performance Study of the Cancelback Protocol for Time Warp. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS ’93)*, pages 135–142. May 1993.

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

[18] Dartmouth College. DaSSFNet(*Dartmouth SSFNet*). See web site at *http://www.cs.dartmouth.edu/research/DaSSF/*.

[19] Davis, H., S. Goldschmidt, and J. Hennessy. Multiprocessor Simulation and Tracing Using Tango. In *Proceedings of 1991 Int’l Conf. Parallel Processing*, pp. II99-II107, Aug. 1991.

[20] Dickens, P., P. Heidelberger, and D. Nicol. Parallelized Direct Execution Simulation of Message Passing Programs. *IEEE Trans. Parallel and Distributed Systems*, vol 7, no. 10, pp. 1090-1105, Oct. 1996.

[21] Ferscha, A. Parallel and Distributed Simulation of Discrete Event Systems. *Parallel and Distributed Computing Handbook*, 1995.

[22] Ferenci,S.L., K. S. Perumalla, and R. M. Fujimoto. An Approach for Federating Parallel Simulators. In *Proceedings of the Workshop on Parallel and Distributed Simulation (PADS 2000)*, pages 63-70, May 2000.

[23] Fujimoto, R.M. The virtual time machine. In *the International Symposium on Parallel Algorithms and Architectures*, pages 199–208, June 1989.

[24] Fujimoto, R.M. Parallel Discrete Event Simulation. *Comm. of the ACM*, 33:31–53, Oct. 1990.

[25] Fujimoto, R.M. *Parallel and Distributed Simulation Systems*. John Wiley and Sons, New York, 2000.

[26] Fujimoto, R.M., and M. Hybinette. Computing Global Virtual Time in Shared Memory Multiprocessors. *ACM Transactions on Modeling and Computer Simulation*, volume 7, number 4, pages 425–446, October 1997.

[27] George, A., and J. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[28] Gerla, M., and J.T.-C.Tsai. Multicluster, Mobile, Multimedia Radio Network. *ACM/Baltzer Journal of Wireless Networks*, 1(3):244–265, 1995.

[29] Gomes, F. Optimizing Incremental State-Saving and Restoration. Ph.D. thesis, Dept. of Computer Science, University of Calgary, 1996.

[30] itDecisionGuru. Available at *http://www.opnet.com/products/itdg/home.html*

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

[32] Jefferson, D.R. Virtual Time II: The Cancelback Protocol for Storage Management in Distributed Simulation. In *Proceedings of the 9th ACM Symposium on Principles of Distributed Computing*, pages 75–90, August 1990.

[33] Karypis, G. Multilevel Hypergraph Partitioning. Available at *http://www-users.cs.umn.edu/~karypis /publications/Papers/Postscript/MOVchapter.ps*.

[34] Karypis, G., E. Han, and V. Kumar. Chameleon: A Hierarchical Clustering Algorithm Using Dynamic Modeling. *IEEE Computer*, 32(8):68-75.

[35] Kernighan, B., and S. Lin. An Efficient Heuristic Procedure for Partitioning Graphs. *The Bell System Technical Journal*, 49(2):291-307, 1970.

[36] Kesidis, G., and J. Walrand. Quick Simulation of ATM Buffers with On-Off Multiples Markov Fluid Sources. *ACM Transactions on Modeling and Computer Simulation*, pages 269–276, July 1993.

[37] Klein, L.R. The LINK Model of World Trade with Application to 1972-1973. *Quantitative Studies of International Economic Relations*, Amsterdam, North Holland, 1975.

[38] Law, L.A., and M. G. McComas. Simulation Software for Communication Networks: The State of The Art. *IEEE Comm. Magazine*, 32:44–50, 1994.

[39] Liljenstam, M., J. Liu and D.M. Nicol. Development of An Internet Backbone Topology for Large-scale Network Simulations. In *Proceedings of the 2003 Winter Simulation Conference*, December 2003.

[40] Lin, Y.B., and B. R. Preiss. Optimal Memory Management for Time Warp Parallel Simulation. *ACM Transactions on Modeling and Computer Simulation*, volume 1, number 4, pages 283–307, October 1991.

[41] Lin, Y.B., B. R. Press, W. M. Loucks, and E. D. Lazowska. Selecting the Checkpoint Interval in Time Warp Simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '92)*, pages 3–10. May 1993.

[42] Liu, B., D. R. Figueirido, Y. Guo, J. Kurose, and D. Towsley. A Study of Networks Simulation Efficiency: Fluid Simulation vs. Packet-level Simulation. In *Proceedings of IEEE Infocom 2001*, April 2001.

[43] Liu, J. and D.M. Nicol. Learning not to share. In *Proceedings of the 15th Workshop on Parallel and Distributed Simulation (PADS 2001)*, May 15-18, 2001, Lake Arrowhead, CA.

[44] Liu, Y. and B. Szymanski. Distributed Packet-Level Simulation for BGP Networks under Genesis. In *Proceedings of 2004 Summer Computer Simulation Conference (SCSC'04)*. July 25 - 29, 2004, San Jose, CA.

[45] Liu, Y. Genesis Reference Manual. Available at *http://www.cs.rpi.edu/∼liuy6/genesis/RefManual.pdf*, February 2004.

[46] Liu, Y. Genesis User Manual. Available at *http://www.cs.rpi.edu/∼liuy6/genesis/UserManual.pdf*, February 2004.

[47] Low, Y.H., C.C. Lim, W. Cai and S.Y. Huang. Survey of Languages and Runtime Libraries for Parallel Discrete-Event Simulation. *Simulation*, vol. 72, no. 3, pp. 111-123, 1989.

[48] Meyer, R.A., and R. L. Bagrodia Path Lookahead: a Data Flow View of PDES Models. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS '99)*, pages 12–19. May 1999.

[49] Mulligan, J.J. Detection and Recovery from the Oblivious Engineer Attack. *Master's thesis*, Massachusetts Institute of Technology, September 2002.

[50] Nicol, D. Challenges in Using Simulation to Explain Global Routing Instabilities. In *Proceedings of the 2002 Conference on Grand Challenges in Simulation*, San Antonio, TX, January 2002.

[51] Nicol, D. Comparison of Network Simulators Revisited. Available at *http://www.ssfnet.org/Exchange/gallery/dumbbell/ dumbbell-performance-May02.pdf*, May 2002.

[52] Nicol, D. Principles of Conservative Synchroniation. In *Proceedings of 1996 Winter Simulation Conf.*, pp. 128-135, Dec. 1996.

[53] Nicol, D., and P. Heidelberger. A Comparative Study of Parallel Algorithms for Simulating Continuous Time Markov Chains. *ACM Trans. Modeling and Computer Simulation*, vol. 5, no. 4, pp. 326-354, Oct. 1995.

[54] Nicol, D., S.W. Smith, and M. Zhao. Efficient Security for BGP Route Announcements. *Technical Report TR2003-440*, Dartmouth College, February 2003.

[55] Nicol, D., and R.M. Fujimoto. Parallel Simulation Today. *Annals of Operations Research*, vol. 53, pp. 249-280, Dec. 1994.

[56] NMS (*Network Modeling and Simulation DARPA Program*) baseline model. See web site at *http://www.cs.dartmouth.edu/∼nicol/NMS/baseline/*.

[57] ns2(*network simulator version 2*). See web site at *http://www-mash.cs.berkeley.edu/ns*.

[58] Ogielski, A.T. SSFnet. *Presentation at the DARPA Next Generation Internet Conference*, Arlington, VA, Dec. 15 - 17, 1999.

[59] Ousterhout, J.K. *Tcl and the Tk Toolkit*. Addision-Wesley, Mass., 1994.

[60] Pei, D., X. Zhao, L. Wang, D. Massey, A. Mankin, S. Wu, and L. Zhang. Improving BGP Convergence through Consistency Assertions. In *Proceedings of INFOCOM 2002*, pages 902–911, June 2002.

[61] PDNS(*Parallel/Distributed NS*). See web site at *http://www.cc.gatech.edu/computing/compass/pdns/*.

[62] Premore, B.J. An Analysis of Convergence Properties of the Border Gateway Protocol Using Discrete Event Simulation. *Ph.D. thesis*, Dartmouth College, May 2003.

[63] Reinhardt, S., M. Hill, and J. Larus. The Wisconsin Wind Tunnel: Virtual Prototyping of Parallel Computers. In *Proceedings of 1993 ACM SIGMETRICS Conf.*, pp. 48-60, May 1993.

[64] Schloegel, K., G. Karypis, and V. Kumar. Graph Partitioning for High Performance Scientific Simulations. *CRPC Parallel Computing Handbook*, Morgan Kaufmann, 2000.

[65] Shekhar, S., and D.R. Liu. Partitioning Similarity Graphs: A Framework for Declustering Problems. *Information Systems Journal*, 21(4).

[66] Shi, Y., N. Prywes, B. Szymanski, and A. Pnueli. Very High Level Concurrent Programming. *IEEE Trans. Software Engineering*, SE-13:1038–1046, Sep. 1987.

[67] SSFNet(*Scalable Simulation Framework Network Models*). See web site at *http://www.ssfnet.org/homePage.html*.

[68] SSF Research Network. Java SSFNet parallel performance. Available at *http://www.ssfnet.org/parallelSolaris.pdf*.

[69] Stallings, W.. *High Speed Networks: TCP/IP and ATM Design Principles*, Prentice Hall, Upper Saddle River, NJ, 1998.

[70] Steinman, J.S. Incremental Sate-saving in SPEEDES using C++. In *Proceedings of the 1993 Winter Simulation Conference*, December 1993, pages 687–696.

[71] Steinman, J.S. SPEEDES: Synchronous Parallel Environment for Emulation and Discrete-event Simulation. In *Advances in Parallel and Distributed Simulation*, volume 23, pages 95–103, SCS Simulation Series, January 1991.

[72] Szymanski, B., C. D. Carothers, S. Kalyanaraman, and K. Vastola. Scalable Online Network Modeling and Simulation. DARPA NMS funded project. Web page: *http://www.cs.rpi.edu/∼szymanski/sonms.html*

[73] Szymanski, B., J.-F. Zhang, and J. Jiang. A Distributed Simulator for Large-Scale Networks with On-Line Collaborative Simulators. In *Proceedings of European Multisimulation Conference - ESM99*, Warsaw, Poland, SCS Press, II:146–150, June, 1999.

[74] Szymanski, B., Y. Shi, and N. Prywes. Synchronized Distributed Termination. *IEEE Trans. Software Engineering*, SE-11:1136–1140, Sep. 1987.

[75] Szymanski, B., Y. Liu, A. Sastry, and K. Madnani, Real-Time On-Line Network Simulation. In *Proceedings of the 5th IEEE International Workshop on Distributed Simulation and Real-Time Applications DS-RT 2001*, August 13-15, 2001, IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 22-29.

[76] Szymanski, B., Q. Gu, and Y. Liu. Time-Network Partitioning for Large-Scale Parallel Network Simulation Under SSFNet. In *Proceedings of Applied Telecommunication Symposium, ATS2002*, B. Bodnar (edt), San Diego, CA, April 14-17, SCS Press, pp. 90-95.

[77] Szymanski, B., A. Saifee, A. Sastry, Y. Liu and K. Madnani. Genesis: A System for Large-scale Parallel Network Simulation. In *Proceedings of the 16th Workshop on Parallel and Distributed Simulation*, pp. 89–96, May 2002.

[78] Szymanski, B., Y. Liu and R. Gupta. Parallel Network Simulation under Distributed Genesis. In *Proceedings of the 17th Workshop on Parallel and Distributed Simulation*, June 2003.

[79] Wetherall, D., and C. J. Linblad. Extending Tcl for Dynamic Object-Oriented Programming. In *Proceedings of 1995 Usenix Tcl/Tk Workshop*, page 288, Berkeley, California, 1995.

[80] Wieland, F. The Detailed Policy Assessment Tool (DPAT). In *Proceedings of 1997 Spring INFORMS Conference*, May 1997.

[81] Xiao, Z., B. Unger, R. Simmonds and J. Cleary. Scheduling Critical Channels in Conservative Parallel Discrete Event Simulation. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS '99)*, pages 20–28, May 1999.

[82] Ye, T., D. Harrison, B. Mo, S. Kalyanaraman, B. Szymanski, K. Vastola, B. Sikdar, and H. Kaur. Traffic Management and Network Control Using Collaborative On-line Simulation. In *Proceedings of International Conference on Communication, ICC2001*, 2001.

[83] Yuksel, M., B. Sikdar, K. S. Vastola, and B. Szymanski. Workload Generation for ns Simulations of Wide Area Networks and the Internet. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, 93–98, SCS Press, 2000.