ROSENET: A REMOTE SERVER-BASED NETWORK EMULATION SYSTEM

A Thesis
Presented to
The Academic Faculty

by

Yan Gu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
April 2008

ROSENET: A REMOTE SERVER-BASED NETWORK EMULATION SYSTEM

Approved by:

Dr. Richard Fujimoto, Advisor
College of Computing
Georgia Institute of Technology

Dr. Mostafa Ammar
College of Computing
Georgia Institute of Technology

Dr. George Riley
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. David Bader
College of Computing
Georgia Institute of Technology

Dr. Haesun Park
College of Computing
Georgia Institute of Technology

Dr. David Goldsman
School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: December 6, 2007

ACKNOWLEDGEMENTS

I am really grateful to my advisor Dr. Richard Fujimoto for his guidance, patience and continuous support. He always gives me the freedom to pursue my goals and helps me to achieve it, through which I grow from a graduate student to an independent researcher. I have been fortune to study under his tutelage.

I also would like to thank Dr. George Riley for providing technical help on GTNetS and always responding promptly to my questions. I want to express my appreciation to Dr. Mostafa Ammar, Dr. David Bader, Dr. Haesun Park and Dr. Dave Goldsman for serving as my committee members and providing me with advices and feedbacks on my research and career.

I benefit a lot from discussions with many people in the College of Computing, mainly in the parallel and distributed simulation group, the Computational Science and Engineering division, the systems group and the networking group, including but not limited to Steve Ferenci, Kalyan Perumalla, Alfred Park, Thom McLean, Jagrut Dave, Zhenyun Zhuang, Jay Lofstead, Qi He, Ravi Prasad, Yarong Tang. I thank the students in Dr. George Riley's group who have given me help on GTNetS. Lometa Mitchell has given me a lot of help and I appreciate that.

The research described in my Ph.D. dissertation was supported under DARPA contract N66001-00-1-8934 and NSF grants CNS-0540160 and ATM-0326431. This support is gratefully acknowledged.

Finally I would like to thank my parents, my brother and my sister-in-law for their love and tremendous support through all the years.

TABLE OF CONTENTS

LIST OF FIGURES

SUMMARY

Network emulation has been widely used to aid in the development and evaluation of real-time applications. Many of today's applications and protocols need to be tested and evaluated in large scale network environments such as the Internet, which requires emulation tools that meet the requirements of ***scale***, ***accuracy***, ***timeliness***. Due to physical resource constraints in network emulators, existing emulation tools fail to meet these requirements as they are either limited to small and static networks, use simplified network models, or fail to deliver timely emulation results. If more physical resources are devoted to network emulation by utilizing high performance computing facilities, the accuracy and scalability of network emulation can be greatly improved. However, for many users, high performance computing facilities may not be readily available in a local laboratory environment, and co-locating application code with a remote high performance computing facility may be cumbersome and inconvenient.

This thesis proposes a network emulation approach called ROSENET (RemOte SErver-based Network EmulaTion) that utilizes a distributed server-based architecture in which local low-fidelity emulators provide real-time QoS predictions to distributed applications, coupled with a remote large scale high-fidelity simulator that continuously updates and calibrates the local low-fidelity emulators. A library-based modeling approach based on online simulation data collection is proposed and a system identification modeling technique is presented. Experimental results examining emulation end-to-end delay and loss show that ROSENET provides a promising approach to network emulation

supporting accuracy and scale while meeting real-time constraints. Challenges faced in applying ROSENET to real world applications are addressed through two case studies including applying synthetic workload on DARPA's NMS network topology for large scale network simulation and a contemporary real-time distributed VoIP application Skype.

1 INTRODUCTION

1.1 Background

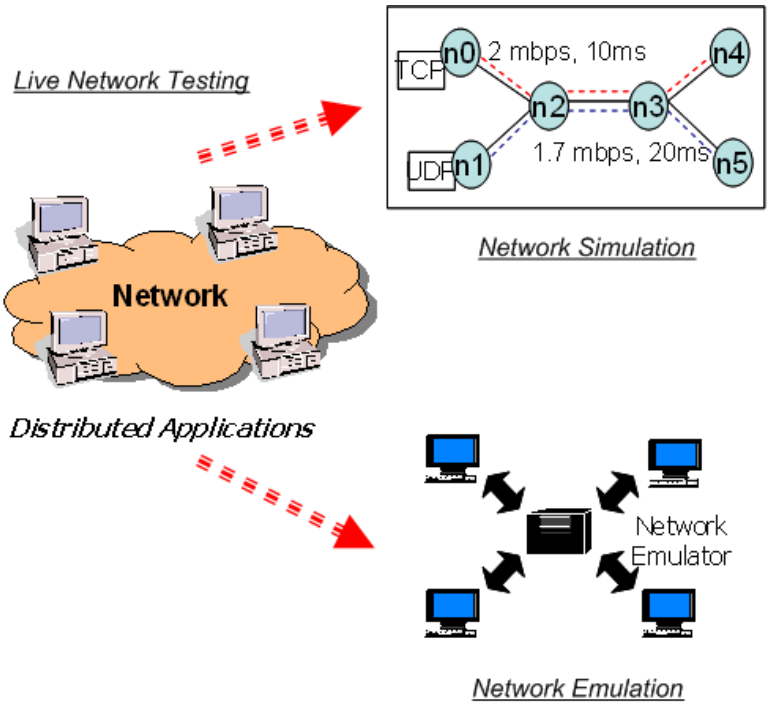

Figure 1: Experiment Techniques

Experimental techniques used in the design and validation of network applications include live network testing, simulation, and emulation, as shown in Figure 1. ***Live network testing*** allows researchers to experiment with new services directly in the real network. PlanetLab [1] is an example of such a network testbed operating over wide geographical distances. This approach tests the applications under realistic conditions, but has the problem that controlled experiments cannot be easily performed and the experiments are not repeatable.

Simulation [2] is the process of exercising a model to characterize the behavior of the

modeled entity process or system over time. ***Computer simulation*** is a simulation that uses a computer program to model the behavior of a physical system over time. Simulations are widely used to analyze systems such as communication networks, transportation systems, electronic systems, and manufacturing system, to mention a few. A simulation of a system can be performed at different levels of fidelity using models that characterize the system using different abstractions.

According to [3], ***Emulation*** is the process by which a device is built to work like another. In the most general sense, an emulator duplicates (provides an emulation of) the functions of one system with a different system, so that the second system appears to behave like the first system. Unlike a simulation, emulation does not attempt to precisely model the state of the device being emulated; it only attempts to reproduce its behavior.

In the scope of this thesis, ***Emulation*** refers to the process that real or physical systems, which can be software, hardware, or applications, interact with virtual systems which are models that characterize the behavior of a modeled entity process or system. Under this context, the main difference between simulation and emulation is that emulation requires real-time interactions between different hardware and software modules while simulation may not support direct execution of real world applications.

Network simulation and emulation are two of the most commonly used tools to test, design, and validate network protocols and applications. ***Network simulation*** uses software to simulate the behavior of the network. It allows users to select an appropriate

level of abstraction. Network simulators provide flexibility in that different types of networks can be modeled. On the other hand, ***network emulation*** provides a test environment that interacts with real world software or hardware systems. The network emulator routes live network traffic generated from real world systems and applies the modeled network's QoS such as delay and loss on the traversing traffic flow. Using network emulation, real hosts can interact with a virtual network that is modeled. In that sense, network emulation can also be defined as a ***real time simulation*** since the simulation interacts with live traffic.

Compared with experimentation over a real wide area network, emulation environments are easier to configure. Further, applications can be co-located in a single facility, eliminating the need to have physically distributed computing facilities and wide area networks, and simplifying debugging. Lastly, network emulators allow researchers to test their applications and protocols in network topologies and conditions that are sometimes hard to achieve in real network environments.

In order to interact with real world hosts and applications, a network emulator must execute at least as fast as the actually system or at ***real-time***, meaning the event occurring in the emulation is paced to occur in synchrony with the interactions with outside applications. Network simulation tools may be used for emulation provided the network simulator is able to compute the required network characteristics in real time. The real-time constraints, also referred to as ***timeliness***, require that network emulators process events and deliver the results to applications within certain deadlines.

In addition to timeliness, scale and accuracy are two other challenges in network emulation. Many of today's research efforts require the test and evaluation of systems in large scale networks (such as across the Internet) with realistic network scenarios. However, existing emulation tools fail to meet the three requirements simultaneously as they are either limited in scale to small and static networks, use highly simplified network models, or fail to deliver timely emulation results.

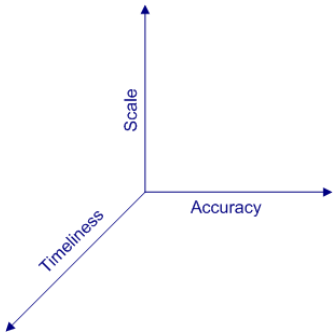

Figure 2: Tradeoffs for the Three Requirements in Network Emulation

Due to constrained physical resources (memory, computation power, and physical bandwidth, etc.), network emulators are often designed to trade one requirement for another. As shown in Figure 2, if a 3-D space is constructed with each axis representing one of the three requirements, different emulation approaches can be mapped as points in this space. Degrees of trade-off among timeliness, accuracy, and scale can be mapped to this 3-D space.

When a large scale network is to be modeled on machines with limited physical resources, the most commonly used approach is to trade accuracy for scale by using simplified models (abstractions) to model the network and traffic, thus increasing the

scale of the network that can be modeled with the same amount of physical resources. By devoting more physical resources to the emulation through the use of high performance computing facilities such as parallel computers or clustered machines, accuracy and scale of network emulators can be greatly improved. However, for many users, high performance computing facilities may not be readily available in a local laboratory environment, and co-locating application code with a remote high performance computing facility may be cumbersome and inconvenient.

A naïve approach to solve the above problem is to perform network emulation remotely by utilizing a remote network emulator on a distant high performance computing facility. This approach achieves accuracy but sacrifices timeliness because each message generated by the application would result in a message being sent to the remote emulator to obtain the desired network characteristics. However, this requires a round trip delay through a wide area network, which may exceed the predicted delay, and it may introduce an unacceptably large amount of traffic over the often limited bandwidth between the application and the remote emulator.

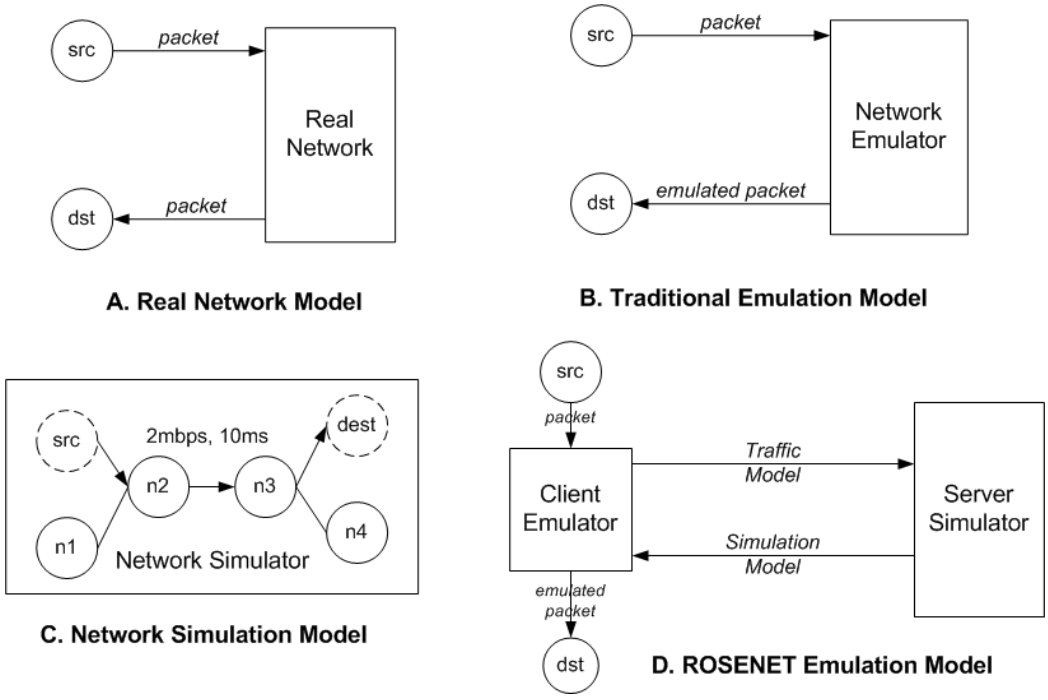

Figure 3: Network Experimental Models

This thesis proposes a novel network emulation approach called ROSENET (RemOte SErver-based Network EmulaTion) that balances the tradeoffs among scale, accuracy, and timeliness. In the ROSENET approach, local low-fidelity emulators quickly deliver results to distributed applications for timeliness, and remote parallel computing facilities perform large scale packet-level simulations and continuously update and calibrate the local low fidelity emulators for scale and accuracy. The tradeoffs among the three requirements can be dynamically adjusted through network models that are periodically exchanged between emulators and simulators. Figure 3 illustrates how the proposed ROSENET approach differs from existing experimental approaches including real network testing, the traditional emulation approach, and the network simulation approach.

1.2 Related Work

1.2.1 Classifications of Emulators

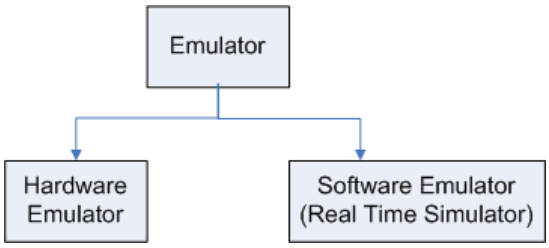

Figure 4: Taxonomy of Emulators

Many network emulation tools have been developed to test network protocols and distributed applications. Depending on whether the emulator is implemented using a hardware device or a software device, the emulator can be classified as a ***hardware emulator*** or a ***software emulator*** (or real time simulator) as shown in Figure 4. Hardware emulator tools can be implemented using a single node machine or a cluster of machines. Software emulators are simulation-based emulators in which a simulation executes on a single node, a cluster of workstations, or parallel machines to simulate a virtual network and interface with live traffic.

The following sections provide a survey of network emulation tools. Section 1.2.2 introduces hardware emulators where the emulators are implemented using hardware devices. Section 1.2.3 discusses software emulation tools that use real time simulators to model the network. Section 1.2.4 summarizes these two approaches.

1.2.2 Hardware Emulators

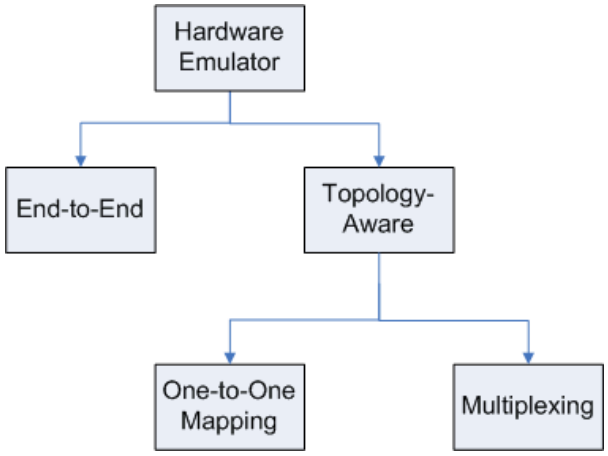

Figure 5: Hardware Emulator by Topology Modeling

Depending on how the network topology details are modeled, hardware emulators can be further categorized as end-to-end emulators (or topology-unaware emulators) or a topology-aware emulator as shown in Figure 5. An ***end-to-end emulator*** does not model the intermediate topology between end hosts and abstracts the network as a cloud or black-box in which only the end-to-end behavior of the network is modeled. A ***topology-aware emulator*** models the intermediate nodes by mapping physical nodes to virtual nodes. An end-to-end emulator is usually executed on a single machine while a topology-aware emulator can run on one machine or a cluster of machines. In a topology-aware emulator, a virtual node in the emulated network can be mapped one-to-one to a physical node in the emulation cluster, or multiple virtual nodes can be multiplexed on one physical node.

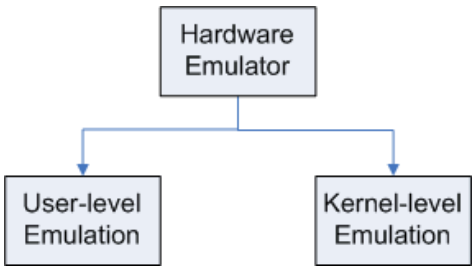

Figure 6: Hardware Emulator by Emulation Layer Placement

Hardware emulators model the virtual network through an emulation layer that intercepts packets and models network characteristics such as delay and packet loss. Depending on where this emulation layer is placed in the system, hardware emulators can also be classified as user-level emulators or kernel-level emulators as shown in Figure 6. The emulation layer can be inserted directly into the operating system kernel but it requires recompiling and rebooting the OS kernel to load the emulation layer. Some emulators are implemented in Linux as a kernel module which can be loaded and unloaded from the operating system kernel. User level emulators usually replace system calls that access TCP or UDP sockets with the emulation's own function libraries to intercept packets. Kernel-level emulation is not easily portable as it requires changes to the operating system but no changes are necessary for the user level application. User level emulations can be easily ported to other operating system environments but it is limited to applications communicating through sockets.

Hardware emulators run on either a single machine or a cluster of machines. On a single node the network can be modeled only in terms of end-to-end characteristics or a small scale network topology may be modeled. Clusters are used to expand the size of the network topology modeled and this method usually models the network by mapping the

virtual network to the physical machines in the cluster. Some cluster-based emulators simply apply single node emulators in the cluster to expand the size of the modeled network topology. The following two sections introduce the emulator tools in this context.

1.2.2.1 Single-node Emulators

A single node network emulator runs on a single general purpose computer and models either a single link or abstracts a target network to a gateway with a set of static network parameters. Examples of these network parameters include delay and loss which describes a LAN's or WAN's end-to-end characteristics. NIST Net [4] uses a single Linux machine set up as a router and a Linux kernel module to intercept packets and apply network traffic dynamics in order to emulate critical end-to-end performance characteristics. Similarly, Ohio Network Emulator ONE [5] emulates a network between a pair of interfaces on a single Solaris-based workstation. Dummynet [6] is the freeBSD version which characterizes each link as a pipe with a certain bandwidth, delay, queue size, and loss rate. Packets can be passed through multiple pipes with applied QoS properties in each pipe. Delayline [7] is a user-level emulation tool that changes the characteristics of the underlying network by providing alternative versions of a number of socket system call routines that are bound to the application program at compile time.

Single node emulators are generally easy to set up because they only require a commodity computer. However, emulation by a single node is often limited in scalability and simulation accuracy because few details of the network can be modeled. For

example, Dummynet cannot capture the congestion of multiple flows on a single path since every flow's behavior is modeled independently and there is no global coordination among the nodes. NIST Net treats the intermediate topology as a cloud in which end-to-end behavior across a network is modeled using statistical methods.

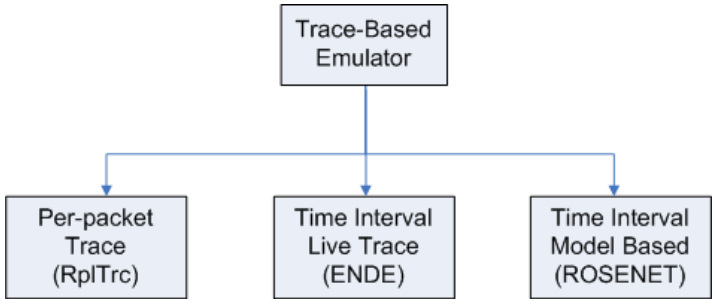

Figure 7: Trace-Based Emulators

Trace-based emulators are developed to solve the problem of generating realistic traffic in network simulations and emulations. The difficulty with synthesizing such traffic lies in the fact that no widely accepted models of Internet traffic exist. Also network traffic shows invariant statistical characteristics that cannot be reproduced easily with simple mathematical models or small distribution tables. Trace-based emulators reproduce network dynamics collected from traces of measured real world traffic as shown in Figure 7. RplTrc [8] adds trace replaying functionality to the NIST Net kernel module so that packet delay traces can be replayed in real time. With the trace replaying functionality, the network environment characteristics can be recreated to enable performance evaluation on a commodity computer with enhanced accuracy. ENDE [9] integrates the functionality of network measurement and network emulation into one system. It generates accurate one-way delay traces using ICMP packets to measure the network status between the local host and a remote host on the Internet. Then it simulates

end-to-end delay between two virtual hosts on a local host using statistics obtained from ICMP measurements and without routing the real packet to the remote host. The problem with this approach is that it can only support certain types of Internet traffic that can be measured by ICMP packets. Also the network measurement results from ICMP packets are not accurate in many network scenarios.

ROSENET, as proposed in this thesis, can be categorized as a trace-based emulator approach since it collects traces from a remote high fidelity simulation, characterizes the measurements as network models and applies them in the local network emulation. The difference between ROSENET and RplTrc is that, when applying traces, RplTrc uses traces of end-to-end delay packets taken directly from the emulated packets, while ROSENET characterizes traces for a time interval as network models and uses the network models to generate QoS predictions for each packet in the network emulation.

Similar to ENDE, ROSENET periodically characterizes the data from traces as models and applies the models in the network emulation. It is different from ENDE in that ROSENET collects the traces from high fidelity simulation while ENDE collects data from real networks using ICMP probe packets. Also ROSENET uses a more complicated network modeling technique than ENDE so that a wide variety of network scenarios can be modeled. ENDE is limited to scenarios that can be measured through ICMP packets.

1.2.2.2 Cluster-based Emulators

Cluster-based network emulators are topology-aware emulators that map a target virtual

network to a cluster of machines in which one or multiple virtual nodes may be mapped to one physical node in the cluster. Compared with single-node emulators, cluster-based emulators can improve the scalability or accuracy of emulation as more resources are devoted to the emulation. However, due to the limited number of physical nodes that can be included in a cluster and the limited number of virtual nodes that can be mapped to a single physical node, topology aware network emulations have been typically limited to systems that are relatively small and static. Different methods must be used to improve the scalability and accuracy of emulation on clustered emulation systems as discussed below.

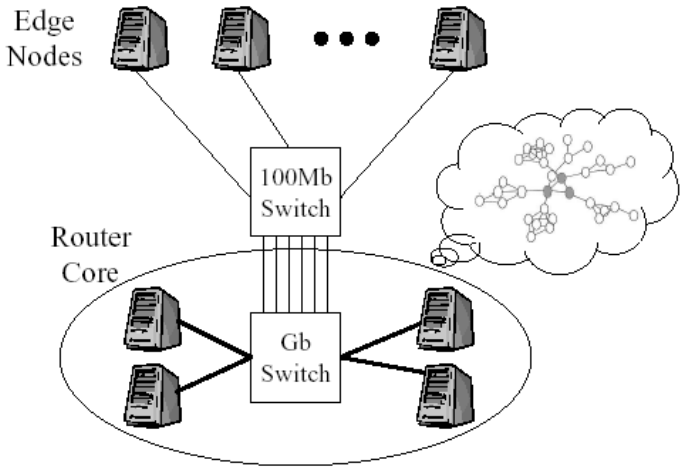

Figure 8: ModelNet Architecture

ModelNet [10] is a cluster-based network emulation system in which *virtual node*s are multiplexed across a set of physical machines. As shown in Figure 8, user applications run on *edge nodes* which route packets through a number of ModelNet *core nodes* connected through gigabit links. The ModelNet core is based on Dummynet [6] and each link in the target network is represented as a pipe with a packet queue. Queuing and link delay/loss is calculated when a packet enters a pipe queue and the packet is forwarded to

the next pipe. Link contention among competing flows in the core is not emulated based on the assumption that bandwidth is constrained only near the edge of the network where edge nodes are connected to the core through 100Mb switches. For background traffic generation, users can dynamically modify pipe parameters (latency, bandwidth, queue size as in an analytical queue model for each impacted link) to represent cross traffic effects in the core nodes on the traversal flow. The pipe queue approach does not capture details of packet dynamics such as slow start and bursty traffic, and synthetic traffic does not respond to congestion.

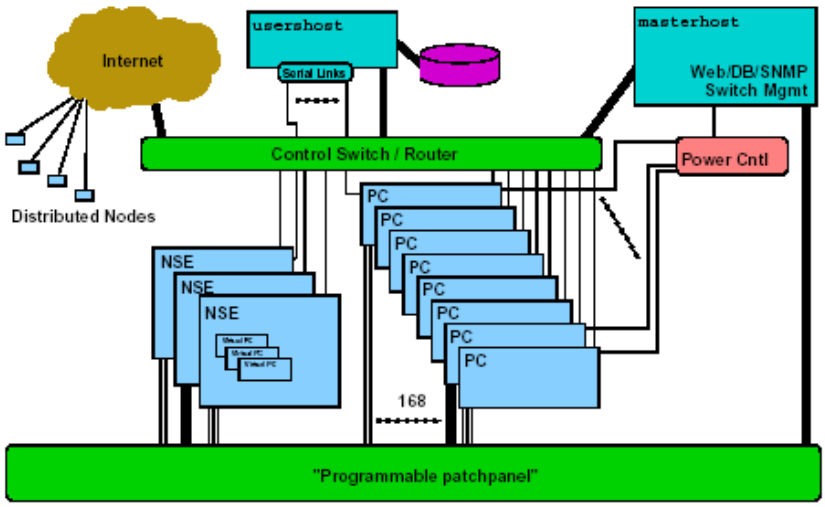

Figure 9: Emulab Architecture

Emulab [11] integrates simulation, emulation, and live networks into a common framework. As shown in Figure 9, a cluster of local resources (shown as 168 PCs in the figure) can be temporarily dedicated to isolated distributed systems and networking experiments for emulation as edge node, traffic generator, or router. Emulab uses virtual machines (virtualization and abstraction) to integrate heterogeneous resources. Dummynet is used to emulate wide-area links within the local-area cluster. Simulation is

integrated into emulation through the NS-simulator-enabled emulator NSE [12]. The major limitations of Emulab are scalability and accessibility. For scalability, each node in the emulated network must be mapped to a machine in the testbed and extra nodes are needed to shape traffic between two connected nodes to provide background traffic. Due to the mapping requirements, the number of nodes that the system can emulate is restricted by the number of machines in the system. With regard to scalability, a local cluster with hundreds of machines is very expensive. Thus the emulation environment is usually not readily accessible to many network researchers.

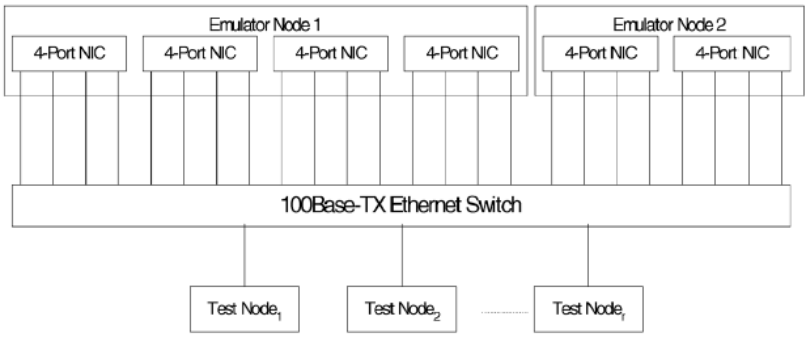

Figure 10: A Sample Layout of EMPOWER

EMPOWER [13] is a clustered network emulation system in which each physical node is equipped with multiple network cards and configured to emulate multiple network nodes (Figure 10). On average each physical node can emulate 4 to 6 virtual nodes in the target network. In EMPOWER the network topology that can be emulated is limited by the number of machines in the cluster, the number of virtual nodes that can be supported on each machine, and the network link bandwidth between the physical machines in the cluster. To solve the network physical link bandwidth limit problem in EMPOWER, multiple network ports can be multiplexed to a physical port when the target network is small, or the bandwidth can be scaled down by sacrificing the accuracy of emulation.

Similarly, background traffic is generated using extra physical nodes.

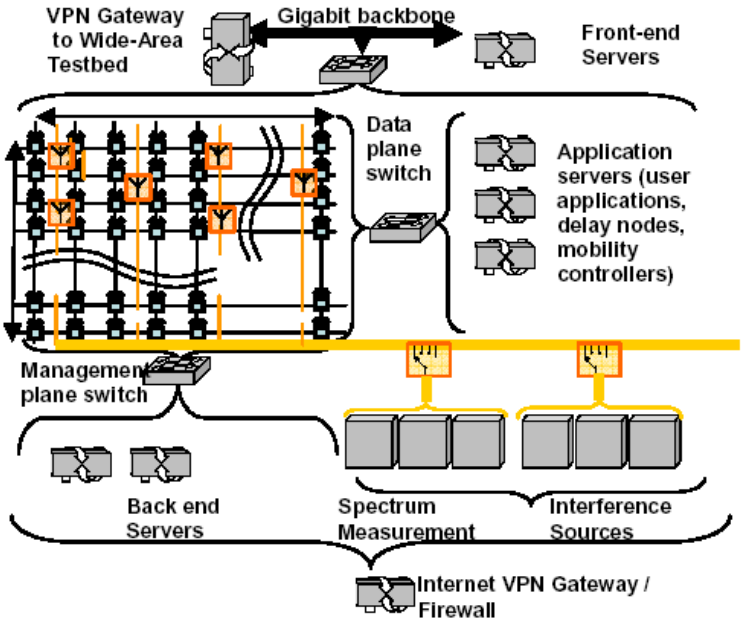

Figure 11: Orbit Architecture

ORBIT [14] (Open Access Research Testbed for Next-Generation Wireless Networks) is a large-scale wireless network testbed which includes a laboratory emulator and a field trial network testbed. As shown in Figure 11, the laboratory-based wireless network emulator is a two-dimensional grid of static and mobile 802.11x radio nodes which can be dynamically interconnected into specified topologies. End-users can download radio links, MAC and network layer protocols to the radio devices to construct a specific networking configuration.

The Network Emulation Testbed (NET) [15] [16] consists of a 64-node cluster system connected by gigabit network switches which establish a number of Virtual LANs for arbitrary network topologies, and also provide fast Ethernet switches which can be used

for administrative purposes. A central administrative node is used to set up and control network experiments. Using node virtualization technology (virtual switch and virtual routing), each physical node can support up to 30 virtual nodes.

1.2.2.3 Limitations of Existing Approaches

Single node hardware emulators abstract the network as a cloud and as such offer limited scalability and accuracy. Cluster based emulator systems improve the scalability of emulation by using a cluster of machines. Virtual network nodes are mapped to physical nodes in the cluster. However they incur a number of problems as discussed below:

- Scalability. The scalability of a cluster-based emulator is still limited to the number of physical nodes in the cluster. Using multiplexing, virtualization, and abstraction, the number of virtual nodes emulated in a cluster can be further improved, but physical resources in the system (such as physical bandwidth and computational power) constrain the number of virtual nodes that can be multiplexed on a physical node.
- Resource competition among multiplexed virtual nodes on one physical node. When multiple virtual nodes are multiplexed on a physical node to improve scalability in a network emulator, performance isolation among virtual nodes running on the same node is not guaranteed due to resource contention. Virtual machines may help solve this problem by isolating the performance of each virtual machine, but the cost of running each virtual node inside a virtual machine is greatly increased. When virtual nodes are not isolated from each other, emulation results may be biased by resource competition among virtual nodes.
- Limited physical bandwidth inside the cluster. The limited physical bandwidth inside

the cluster limits the number of virtual nodes and links that can be supported on a single physical node, which in turn limits the total number of nodes supported in the network topology.

- Background traffic generation. In clustered systems, background traffic consumes system resources that could have been used by the emulation systems. To solve this problem, clustered systems either synthetically limit the bandwidth by using metrics to imitate the effects of cross traffic on the link (ModelNet), or use extra physical nodes to inject background traffic into the system even though synthetic traffic is not responsive to congestion (EMPOWER).
- Mapping. Mapping the virtual network to the physical nodes requires network partitioning and load balancing among the nodes in the cluster, which is not an easy task.
- Configuration. The target network topologies and parameters in an emulation cluster cannot be as easily configured as in a simulation environment. With more machines participating in the emulation, both the setup of machines and the coordination of emulation tools becomes a problem.
- Accuracy. Network models in emulation clusters are often simplified to improve performance at the cost of emulation fidelity.

1.2.3 Software Emulators

Simulation tools are being used for emulation in order to improve the flexibility and scalability of the target network. Emulators that are implemented using simulators that interfere with live traffic are referred to as ***software emulators***. Network simulation tools may be used for emulation provided the network simulator is able to compute the

required network characteristics faster than real time. This simulation-based emulation is also called ***real time network simulation*** because the simulator needs to execute in real time, and because it can be used to evaluate performance of real time network applications or used to study network protocols under real world traffic.

1.2.3.1 Classification of Software Emulators

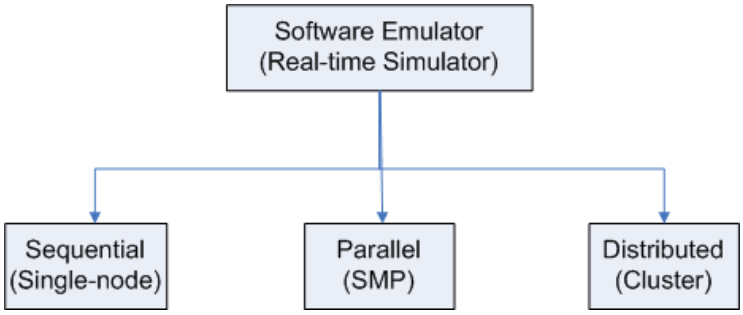

Figure 12: Taxonomy of Software Emulator (Real Time Simulator)

Depending on the platforms where the simulation for software emulators is executed, software emulators can be classified as single-node, parallel, or distributed, as shown in Figure 12. A single node software emulator executes a sequential simulation in real time on a single machine. A parallel software emulator executes the simulation in parallel on a high performance machine such as an SMP. Software emulators can also be distributed over a cluster of machines.

As has been shown in Section 1.2.2, hardware emulators can also be classified as single node or clustered, but the implementations are very different. A single node emulator uses a single machine to emulate a link or an Internet cloud using mathematical tools while a software emulator executes a sequential simulation on a single node and

synchronizes the simulation time with wall clock time, or real time. Similarly, a clustered emulator uses physical resources such as physical bandwidth to emulate a network topology by mapping virtual nodes to the physical nodes in the cluster, while a clustered software emulator executes a parallel/distributed simulation on the cluster platform.

NSE [12], the emulation capability of the Vint/NS simulator, routes live traffic between the simulator and the distributed application. NSE uses a real time scheduler which ties event execution within the simulator to real time. Since ns-2 runs on a single node, the scale of the network topology and the amount of traffic that can be simulated in real time is very limited. To improve the scalability of single node simulator-based emulation, parallel and distributed simulation can be introduced to improve the scale of the network topology in the network emulation. The following section introduces the concept of parallel and discrete event simulation upon which software emulators can be implemented.

1.2.3.2 Parallel Discrete Event Simulation

A ***discrete event simulation*** (DES) models a system where changes in the state of the system occur at discrete points in simulation time. A discrete event simulation is typically used for packet level simulation of networks where events are scheduled for packet departure, arrival, and loss at discrete points in time. This thesis focuses on packet level simulation/emulation, so simulation is assumed to be a discrete event simulation unless specified otherwise.

Parallel Discrete Event Simulation (PDES)[17] [2] refers to the execution of a discrete event simulation program on a parallel computer. Parallel discrete event simulation offers the potential to simulate large scale network topologies at a detailed packet level on parallel and distributed machines, thereby increasing the size of the network and the amount of traffic that is modeled. Using PDES, networks can be partitioned and simulated concurrently on multiple machines.

A number of tools have been developed utilizing parallel computing facilities to improve the scalability and performance of network simulation. These include PDNS [18], GTNetS [19], DaSSF [20] [21], GloMoSim [22] and its commercial successor QualNet [23], and Genesis [24]. Parallel discrete event simulation can run most efficiently on shared memory multiprocessors. As large scale shared memory machines are expensive and the number of processors available on an SMP is still limited, distributed memory machines are often used. [25] studied large scale network simulation on a variety of platforms ranging from workstations to cluster computers to supercomputers and demonstrated the ability to simulate a million web traffic flows in near real time using GTNetS and PDNS.

1.2.3.3 Simulation-Based Emulators

Based on the above mentioned parallel and distributed simulators, a number of simulation-based emulators have been implemented. IP-TNE [26] is a network emulator using the parallel discrete event simulation IP-TN that runs on shared memory multiprocessors. Unlike NSE which ties the events with wall clock time, IP-TNE

synchronizes only the edge of the simulated network interacting with real traffic with wall clock time. [27] [28] report performance results of running real-time emulation on 128 processors for a network model including 20,000 nodes of nearly 50 million packet transmissions per second.

Maya [29] is a hybrid software emulator which combines analytical models, simulation, and interfaces to operational networks and enables emulation for real time applications. It uses QualNet [23], a fluid model for TCP, and a physical network interface implemented on the Linux operating system. The event scheduler in QualNet must synchronize with real time during its execution. The TCP fluid model is tied with the physical network interface and network statistics over a time interval that is calculated periodically.

RINSE [30] is a real-time network simulator for large-scale human/machine-in-the-loop experimentation used for security and training exercises. It uses reader/writer threads to convert between simulation events and packets from real time applications. IN RINSE, it is observed that, the latency of the physical connection between the real world application and the simulated host in the virtual network can be hundreds of microseconds in a local area network. This can have a great impact on applications that are sensitive to such latencies and may increase the number of missed deadlines in the emulation. The solution in RINSE is to hide latency, caused by the physical connection between simulator and real time applications, inside the simulated network. This prioritizes emulation events over regular events in scheduling and sending the emulation packet ahead of its scheduled time from one router to another in the simulation, based on the assumption that

without the physical connection latency the event would have entered the queue much earlier. This priority-based scheduling does not follow the FIFO rule any more for emulation events and the simulation accuracy is thus sacrificed for timeliness.

MaSSF [31] is a network simulator/emulator based on grid computing principles. To solve the scalability issue with detailed packet level emulation, the network emulator is built over a distributed simulation engine DaSSF [32] to exploit the availability of scalable cluster systems. A real time scheduler is implemented and run on a cluster machine using MPI. The scheduler can also run in a scaled-down (slower than real time) mode when the simulation system is too large to be simulated in real time. A CPU controller virtualizes the CPU resources among multiple virtual hosts. This way it can simulate a large number of machines (100's to thousands) on a small number of physical machines. It is more focused on studying performance questions in grid applications instead of implementing a real network emulator.

Communication Effects Server (CES) [33] from Scalable Technology Inc. is a wireless network simulator based on QualNet [23]. In CES, a transaction is defined as the end-to-end transmission of a message from its source to destination. ***Transactional real-time*** execution is defined as one where the wall-clock time to execute an average transaction is less than its simulation time. Transactional real-time is achieved for nearly all messages in a wireless network with thousands of virtual nodes on no more than 16 processors of a cluster, with simulated link bandwidth of 2Mbps under different traffic loads.

The GT backplane [34-36] integrates multiple heterogeneous network simulators in a single environment. The backplane also supports incorporation of actual network applications into the execution over the emulated network by integrating parallel network simulators with an emulation backplane. The end-node applications and the parallel network simulators use the runtime infrastructure [37] to pace their execution. Support for real-time execution is realized by synchronizing all the components in the emulation with real time. Applications interact with the simulation backplane through a library called Veil [36] which intercepts system calls and re-routes the application data to the simulated network.

In [38] real time lookahead is exploited from the interactions between wireless applications and the simulated wireless network in the parallel simulator GloMoSim. In a hybrid component network, simulated components (GloMoSim) receive messages directly from physical components (real-time applications). The simulated components have lookahead of zero as they can not predict when they receive messages from the physical components. Since the maximum throughput of a traffic flow is limited by the link capacity, the data stream transmitted by the mobile node will not change the simulation immediately if the data buffer at the simulator is not empty at time T1 until a later time T2. This allows the simulator to advance its simulation time ahead of real time to time T2.

In addition to the previously mentioned systems that apply parallel simulation techniques to network emulation, PRIME [39] is a system most similar to ROSENET. PRIME aims

to implement an open and scalable network emulation infrastructure to allow a large number of real time applications to dynamically interface with network simulators running on supercomputers. It uses a Virtual Private Network (VPN) to bridge traffic between physical entities and network simulators. Real applications run as VPN clients which automatically forward network packets to VPN servers. VPN is used to circumvent the firewall of the supercomputing center and also serves as a network interface on the client machine.

Sharing the common goal of providing an open network emulation infrastructure to test real time applications by utilizing network simulators running on supercomputers, ROSENET differs from PRIME with its support for remote access. ROSENET allows users to access the high fidelity simulation remotely and also meet real time requirements for QoS predictions. The experimental results in PRIME show that with a simple dumbbell topology, losses and delays experienced by the packets as they travel through the simulation gateway can have a profound impact on emulation accuracy. Good results are achieved when the simulation and applications are in a local area network but this approach has difficulty meeting real time deadlines when packets are sent over a wide area network to the simulator due to wide area network packet loss and end-to-end delay. In addition the simulation gateway's bandwidth and latency could affect the quality of service of the applications. These two problems are identified in [40]. The results in PRIME further confirm that network emulation with remote accessibility is required. In this sense, ROSENET and PRIME are not directly comparable. ROSENET can complement PRIME by integrating PRIME into its client/server framework as the high

fidelity simulation server and providing PRIME users with a remote access capability.

ROSENET and PRIME also address the same problem of large scale network emulation using different approaches. In order to achieve results within the emulation's real time constraints, PRIME integrates a fluid model with simulated packets to improve parallel simulation performance, while ROSENET uses network models and periodic model updates, instead of sending individual packets as in PRIME, to trade time for accuracy. Using fluid model in ROSENET is an area of future work.

1.2.3.4 Limitations of Existing Approaches

As seen from the discussion of existing software emulators, several approaches are used in order to achieve real time performance:

- Scale

  The network topology of the parallel simulation based emulators mentioned above must be relatively small. In the performance evaluation of IP-TNE, the network is either composed of only two end nodes with one router, or sixteen nodes which results in 16% of the packets' arrival after their scheduled delivery time on a 4-processor system.

- Timeliness

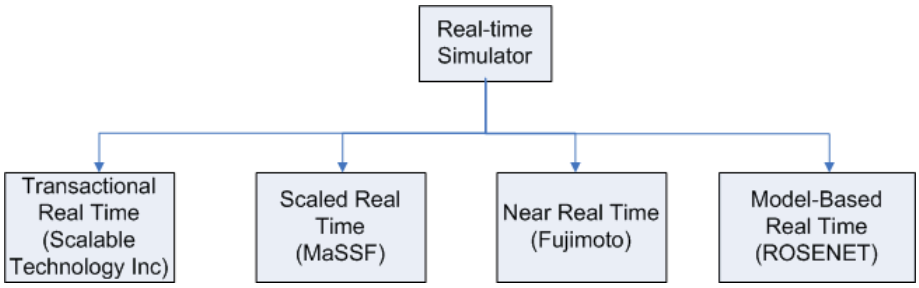

Figure 13: Definition of Real Time in Real Time Simulator

Applying simulation in emulation requires that simulators execute faster than real time. Directly applying parallel simulators for emulation may not be effective because parallel simulators are primarily designed to improve scalability and accuracy rather than meeting specific deadlines. If real time performance is to be achieved for a detailed packet level model, the topology of the system must be kept small as in IP-TNE. The CSE from Scalable Technology Inc. modifies the definition in simulation based emulation as ***transactional real time*** when the wall-clock time to execute an average transaction is less than its simulation time as a solution to the timeliness problem. ***Scaled real-time*** (slower than wall clock time) simulation (MaSSF) is defined as an alternate solution. The results from [25] report that ***near real time*** performance is achieved for detailed packet level million nodes simulation on a supercomputer. ROSENET uses a ***model-based real time*** trading accuracy for timeliness, meaning it can always meet the real time deadline but the accuracy may be traded for timeliness accordingly. Figure 13 summarizes the modified definition of real time in different network emulators.

- Accuracy

To improve performance and scalability of parallel simulation, a less detailed model

of the network is often used, resulting in some loss of simulation accuracy. Abstraction methods such as fluid models, integrated analytical model (MAYA), or simplified models of network nodes as queues (ModelNet) are sometimes used.

- Accessibility

  All of the above approaches attempt to apply parallel simulation to network emulation and also try to meet the real time requirement. But they all fail to meet another requirement: ***accessibility***. Parallel computing facilities required in these network emulators may not be locally available, and co-locating application code with a remote high performance computing facility may be cumbersome and inconvenient. If a user attempts to use high performance computing facilities remotely, the real time constraints for emulation may not be met since latency between the application and remote simulation servers may exceed predicted delay. Although many research efforts have been designed to realize an emulation framework that is remotely accessible, they require participants to upload models to the test-bed remotely and observe the system performance without interactions during the testing.

In summary, large-scale simulation based emulation systems require simulation to be able to characterize the behavior of a network with a large number of nodes and with realistic traffic loads. Approaches have been developed but no single method exists that can simultaneously address the issues of scalability, timeliness and accuracy while being accessible to general users.

1.2.4 Summary

An ideal network experimentation tool for evaluating large-scale distributed services

should support direct execution of applications, a broad range of network topologies and dynamically changing network characteristics with a wide spectrum of network behaviors, and a sufficiently large number of nodes with realistic models of cross traffic. The scalability of network topology in hardware emulators are constrained by the physical resources available and it also may be difficult to set up and configure the system compared to simulation. Software emulators using parallel and distributed simulation techniques have greatly improved the scale of the simulated network but achieving real time performance at detailed packet level for large scale network is still difficult.

1.3 The ROSENET Approach

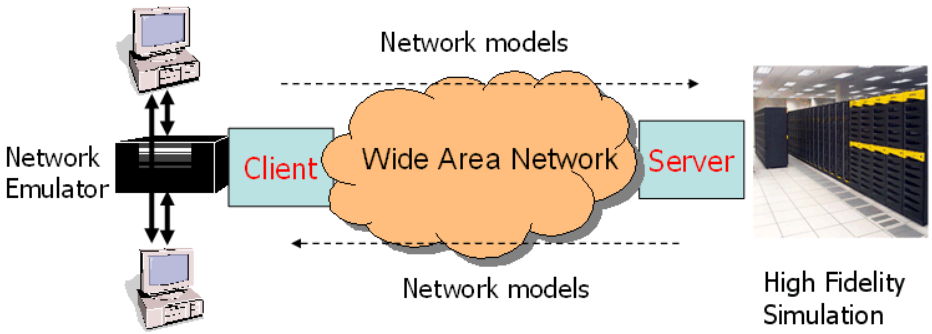

Figure 14: ROSENET Approach

The ROSENET approach proposed in this thesis attempts to address the problems of scale, accuracy, and timeliness simultaneously. As shown in Figure 14, in ROSENET, a packet-level simulation is used to provide accuracy. Mapping this simulation to parallel/distributed computing facilities provides scale. The local emulator provides timeliness. While this approach focuses on addressing all three problems, tradeoffs among the three requirements can be dynamically adjusted using different network

models. Compared with traditional hardware and software emulation systems, the ROSENET approach has the following advantages:

- **Accessibility**. In order to achieve scalability, clustered emulation systems are usually composed of tens or hundreds of machines which may not be likely available. This limits the opportunity to directly execute real world applications on the emulators. The remote server-based approach in ROSENET makes it complementary to existing emulation systems. For example, clustered hardware emulation systems will be more accessible to general users if these clusters are integrated into ROSENET to allow users to test their applications remotely. Another example is ORBIT, which is a wireless network testbed composed of a two-dimensional grid of 400 802.11 radio nodes. ORBIT has a similar problem that the access to geographically remote users is limited for emulation users. The ROSENET approach can similarly be utilized to address this problem.
- **Flexibility**. The ROSENET approach provides the flexibility to allow users to test networks with a diverse set of network topologies and traffic loads. A cluster emulator designed for wired network emulation will not be applicable for a wireless network. The ROSENET architecture provides the ability to easily switch users to wireless emulation.
- **Security**. Applications such as Internet simulations are motivating the use of high-end computing for parallel discrete event simulation. Supercomputing facilities can provide tens to over 100 thousand processors [41]. Since these high end computing facilities are usually shared among users, up-loading user applications to a remote high performance facilities may not be preferred for users who are concerned with

privacy. The ROSENET approach separates the emulation task into a local low fidelity emulator and remote high fidelity simulator, which allows users to protect the security of their applications without the need to directly execute them in remote facilities.

- **Address scale, accuracy and timeliness simultaneously**. Highly accurate simulations at large scale are always desirable for network researchers but the limited availability of physical resources may force users to trade accuracy against time. A single node emulator using a single machine may not be able to simulate the network with much detail. Trace-based methods try to make up for this inaccuracy by using real world data but are still limited to the specific network scenarios corresponding to the traces. Clustered network emulators can improve the scale of network topology, but clustered machines may not be readily available. The ROSENET approach addresses the three requirements simultaneously and allows users to tradeoff these concerns according to their specific requirements.

1.4 Research Challenges

Remote network emulation must successfully address several research challenges for it to effectively meet our requirements of accuracy, scale, and timeliness:

- **Multi-resolution network modeling**. Remote network emulation requires a suitable combination of a high fidelity model residing at the remote server, and a low fidelity model operating locally at the client. This raises several important research questions, especially with respect to the low fidelity model. Specifically, what model should be used at the client to ensure accurate QoS predictions are produced in a timely fashion that do not impose large computation and communication burdens on the system?

Can a single model be used for this purpose that can span the broad range of network configurations and traffic that one might expect? If one model is not sufficient, how can multiple models be integrated and utilized in a seamless fashion, and what strategies are needed to control and manage the use of these models? What is the relationship between the high and low-fidelity models?

- **Model update protocols**. Remote network emulation requires that the low fidelity model at the client be periodically updated to so that QoS predictions reflect the current network status from the high fidelity simulation at the remote server. This raises important research questions with respect to low fidelity model update protocols. Specifically, what needs to be updated and what triggers model updates? What needs to be transmitted to update a network model to reduce communication cost while achieving optimal performance? What effects will the model update protocol have on the system performance, in particular emulation accuracy, and how should one measure these effects quantitatively? Using these measurement metrics, how should one design and choose among different model update protocols under operating conditions?
- **Performance evaluation**. Remote network emulation differs from existing emulation approaches in that traditional emulation tools are composed of either a single node machine or a homogeneous cluster while the remote emulation approach integrates general distributed emulation and simulation tools into a heterogeneous environment over a wide area network. Measuring and evaluating the emulation performance with such a complex system raises several research questions. In particular, what metrics should be used in measuring the performance of a remote network emulation system

with regard to scale, accuracy and timeliness? What mathematical and statistical tools should be used to measured emulation accuracy? What system factors can affect emulation accuracy and to what extent do these factors affect accuracy? How should one measure timeliness and what are the factors that can prevent an emulated packet from being delivered on time? What is the relationship between scale, accuracy and timeliness and is there any way this relationship can be quantified so that users can easily define their requirements on these parameters and make tradeoffs among them? What applications are suitable to be evaluated in the remote network emulation platform and how should one integrate a real world application using the remote network emulation system?

- **System architecture design**. Remote network emulation uses a remote high fidelity simulation server to achieve scale and accuracy and a local low fidelity emulation client to achieve timeliness. Designing a general system that can meet the requirements for scale, accuracy, timeliness and accessibility raises several research problems. Specifically, how should one integrate heterogeneous simulators and emulators so that it is flexible enough to support different types of simulators and emulators, under different experimental settings and applications? What functionalities are needed in the clients and server to meet the emulation requirements? If large scale parallel and distributed simulation is executed at the remote server, what specific functionalities are needed to seamlessly integrate the distributed client/server architecture over a wide area network with the distributed high fidelity simulation in a local network environment? How much overhead does each component introduce into the system and how can one fine-tune the system

configuration to achieve optimal performance?

1.5 Research Contributions

This thesis addresses the challenges in remote network emulation and the contributions are summarized as follows:

- **Remote Network Emulation Architecture.** A client-server architecture for remote network emulation is proposed, implemented and evaluated. Standard interfaces are defined for high fidelity simulation, low fidelity emulation, and network models. This allows users to integrate their specific applications, simulators, and emulators into the system while maintaining acceptable performance and accuracy. A time management and synchronization protocol for time advance in the emulation client and simulation server is proposed. An approach to address the challenges of parallel discrete event synchronization is described. Finally an analytical model is examined to estimate the amount of delay introduced by each component in the system and the remote emulation delay as a whole.
- **Network Modeling Techniques**. In order to address the network modeling challenges in remote network emulation as discussed in previous section, the network constancy properties and the constancy scale are first studied to justify using network models, instead of individual packets, to describe network traffic within a time interval. Then different network modeling techniques are explored and a library-based modeling technique is proposed for remote network emulation. A state divided system identification approach to model the end-to-end delay is proposed in ROSENET. Using a large scale worm propagation simulation, it is shown that the end-to-end delay in the server-based architecture can be accurately modeled using the

system identification approach.

- **Performance Evaluation and Case Studies**. In order to verify that ROSENET can generate accurate emulation results while meeting real-time constraints for network emulation, a series of experiments are conducted to exam the accuracy of end-to-end delay and loss prediction. The results show that ROSENET can accurately predict QoS parameters while meeting real time constraints for network emulation, thus providing a promising approach to network emulation supporting accuracy and scale while meeting real-time constraints. Two case studies are performed to demonstrate the usage of ROSENET including testing a commercial VoIP application Skype on ROSENET and evaluating ROSENET's performance with synthetic traffic workloads over DARPA's NMS network topology for a large scale simulation.

1.6 Thesis Organization

The remainder of the thesis is organized as follows. Chapter 2 describes the design of the client-server architecture for remote network emulation. Chapter 3 discusses network modeling issues. Chapter 4 provides a performance evaluation of ROSENET and presents two case studies illustrating its use. Chapter 5 summarizes the thesis and suggests future research directions.

2 REMOTE NETWORK EMULATION ARCHITECTURE

In the remote network emulation approach, low fidelity emulator and high fidelity simulator execute in a distributed manner over a wide area network to support different types of simulator, emulator, experimental settings and testing applications. Designing a general emulation system that is flexible enough to meet the requirements for scale, accuracy, timeliness and accessibility raises several research questions. This chapter addresses the research problems in remote emulation architecture design and the rest of this chapter is organized as follows. Section 2.1 describes the system architecture and discusses specific design issues. Section 2.2 explains research issues in extending the system to support parallel discrete event simulation. Section 2.3 introduces an analytical model to measure remote emulation delay, a metric which can be used by users to estimate the network emulation accuracy.

2.1 System Architecture

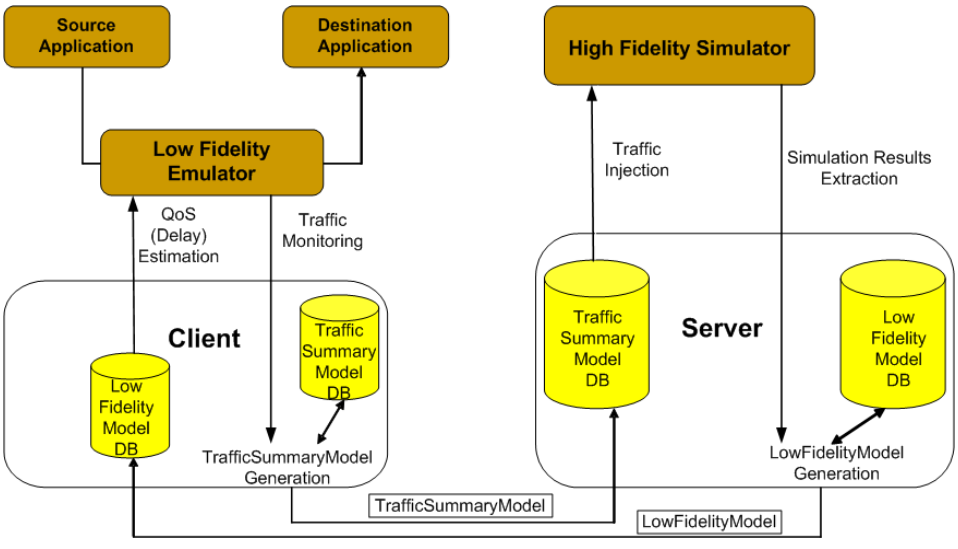

Figure 15: ROSENET System Architecture

The system architecture of ROSENET is shown in Figure 15. The client and server are the principle components of the system. The low fidelity emulator at the client routes traffic between distributed applications, provides rapid QoS estimation, and monitors application traffic. The remote server controls the high fidelity simulation by injecting traffic and extracting simulation results such as end-to-end delay from the simulation. Simulation time is partitioned into time intervals with the assumption that traffic characteristics change little within each time interval [42]. Clients and servers exchange their status through periodic updates of network models at the end of each interval. The update frequency can be dynamically adjusted according to the required accuracy and capacity of the simulation as well as the available bandwidth between the client and server.

This client/server design allows ROSENET to achieve timeliness through the low fidelity network emulation and improves network modeling scale and accuracy through the high fidelity network simulation. Flexibility is obtained by integrating different simulators and emulators through the High Level Architecture (HLA) [43] as well as by defining standard interfaces to hide internal implementation details of simulation and emulation. Periodic network model exchanges between simulators and emulators can dynamically trade simulation accuracy for time, and can reduce large bandwidth consumption and avoid the wide area round-trip delay required if each packet is to be sent to the remote high fidelity simulation to simulate.

2.1.1 Time Interval

In ROSENET, a time interval can last seconds, minutes or even longer depending on

user's emulation accuracy requirements. Two special scenarios illustrate some of the tradeoffs that are possible in ROSENET. In the first scenario every packet generated by the application is sent to the remote simulator in order to get the actual end-to-end delay and loss for that particular packet. In the other scenario all packets generated by the application during the entire execution are collected and sent to the simulator. Then the simulation results will be used by the emulator, either as static parameters, or as traces, to generate QoS predictions for emulated packets and deliver to the destination application. The accuracy in these two examples represents two extremes that a ROSENET user should expect. The time interval can be dynamically adjusted according to the required accuracy and capacity of the simulation as well as the available bandwidth between client and server. The client and server exchange their status through periodic updates of network models at the end of each interval.

2.1.2 Application Traffic Monitoring

In a typical network emulation, applications transmit traffic using standard communication primitives such as send() and recv(). The network emulator routes the packets from source application host to the destination host while introducing QoS properties (e.g., delay, loss) to the packet. The following techniques are usually used to intercept packets in network emulation:

- Integrate interception function into kernel. Several network emulators implement a kernel model that intercepts IP packets and applies network dynamics to the packets. Since the packets are intercepted at the IP level, this kernel module can affect all networking applications regardless of their higher level network protocol type, and the implementation is transparent to applications or network protocols.

The disadvantage of this technique is that kernel code has to be modified.

- Use shared libraries that provide alternative versions of system call routines or compile time switches to force applications to use alternative header files. Using a shared library requires that the system supports shared or dynamic libraries. Using alternative header files redefines system call entry routines to emulation functions and it requires modifications of the application's source code. The advantage of these two approaches is that they do not need to change kernel code, but they may only be able to support a specific type of packet such as TCP packet.

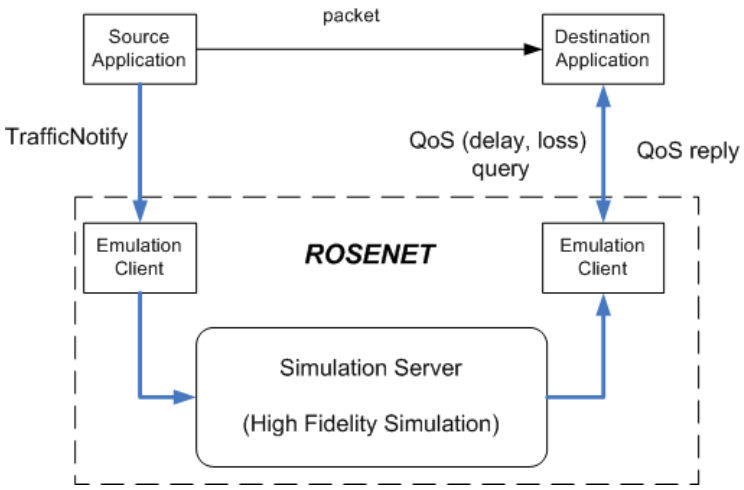

Figure 16: Non-Intrusive Clients

Sometimes application traffic monitoring through packet interception may not be allowed due to security concerns. Therefore, in addition to the normal "intrusive" emulation mode defined in Figure 15, a "non-intrusive" emulation mode is also introduced as shown in Figure 16. In a non-intrusive mode, the system only provides QoS estimations upon request from applications, and it is the application's responsibility to route and deliver messages to the appropriate destination. The application also notifies the emulation client

of the generated traffic characteristics. The non-intrusive mode provides a means to easily "turn off" the emulator and avoids routing sensitive data through an un-trusted emulation device.

2.1.3 High Fidelity Simulation

```
class HFSimulation
{
  public:

    virtual void initSimulation() = 0;

    virtual bool mapApplicationToNode(SimId) = 0;
    virtual bool unmapApplicationToNode(SimId) = 0;

    virtual void runSimulation(double endTime) = 0;

    virtual void addTraffic(SimId from, SimId to, TrafficType tt,
          TrafficParameters *tp, double stopTime) = 0;

    virtual map<SimId, RawData *> * getSimulatedResults() = 0;
};
```

Figure 17: High Fidelity Simulation Interface

The high fidelity simulation interface defined in Figure 17 allows the ROSENET server to interact with the high fidelity simulation without the need to know the implementation details of the simulator. The server uses this interface to initialize the high fidelity simulation, map or un-map applications to virtual nodes in the simulation, execute the simulation until a specified simulation time, update traffic on a source virtual node during the simulation, and extract simulation results at the end of a simulation interval. It is assumed that the high fidelity simulation runs as a discrete event simulation and can pause at certain simulation times, update traffic models at a source virtual node, and collect end-to-end delay and loss statistics for a source-destination pair during the simulation execution.

Two types of traffic flows exist in the high fidelity simulation: background traffic and injected live traffic. The virtual nodes in the simulation, mapped for real time applications, can have traffic injected into the high fidelity simulation through periodic update of *TrafficSummaryModel* based on the current traffic generated by the source application being tested.

2.1.4 Network Model Interface

Since network simulators are able to simulate a wide spectrum of network dynamics, a few specific modeling techniques may not be able to adequately characterize network dynamics. Therefore in ROSENET a library of network modeling techniques is provided and specific network modeling techniques are explored which are sufficiently general to be used in different applications. In addition, users are allowed to integrate network modeling libraries specific to their real time applications into the ROSENET system. The network modeling details are discussed in Chapter 3 and here only the network model interfaces are presented.

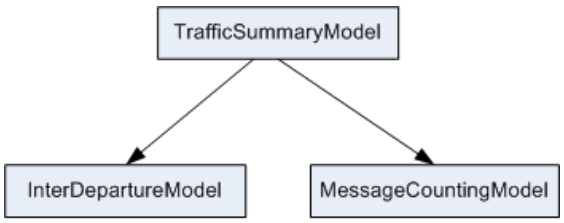

Figure 18: *TrafficSummaryModel* Object Hierarchy

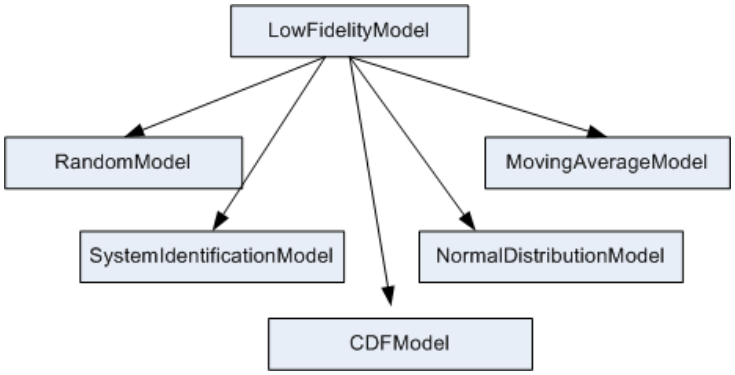

Figure 19: *LowFidelityModel* Object Hierarchy

To hide the implementation details of network models from the emulation client and simulation server, two network model interfaces are defined in ROSENET. *TrafficSummaryModel*, generated by the client and used by the server to update the traffic in the high fidelity simulation, characterizes the traffic generated by the application. *LowFidelityModel*, generated by the server and used by the client to generate QoS predictions quickly on demand, describes measurements from the high fidelity simulation such as end-to-end delay and loss. As shown in Figure 18, *TrafficSummaryModel* can be described using packet inter-departure time or the number of packets generated in a time interval. Figure 19 shows that a number of mathematical and statistical techniques have been used to model the end-to-end delay. In particular, a System Identification model is proposed in [40].

```
class TrafficSummaryModel
{
   public:
      // update model parameters or lifetime
      virtual void update(void * parameters, double fromTime, double toTime) = 0;

      // extend model's life time
      virtual bool extendTo(double toTime) = 0;

      // compare two models
      virtual bool differFrom(TrafficSummaryModel *trafficSummaryModel ) = 0;

      // convert model parameters to a string
      virtual void * getParameters(int &size) = 0
};
```

Figure 20: *TrafficSummaryModel* Interface

Users can implement a customized *TrafficSummaryModel* as long as it follows the standard interface as defined in Figure 20. The functions required in an extended *TrafficSummaryModel* include updating the current model parameters or expiration time, extending a model's expiration time, comparing two models to see if they are the same so that unnecessary updates can be avoided, and converting the model parameters to strings when the models are transmitted to the server. The interface of *LowFidelityModel* is similarly defined and will not be listed here.

2.1.5 HLA/RTI

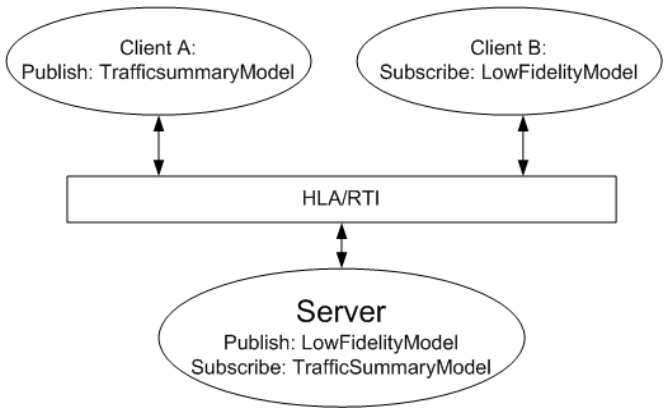

Figure 21: HLA System Architecture

The High Level Architecture (HLA) [43] has been used by the simulation community to integrate (federate) computer simulations in distributed computing environments. Since ROSENET integrates autonomous network simulations, HLA provides a suitable vehicle to hide the internal details of the underlying simulation models and facilitate future extension. Therefore HLA is used to define the interactions between the client and server. Each client, as well as the server, is defined as an HLA federate. *LowFidelityModel* and *TrafficSummaryModel* are implemented as Federation Object Model (FOM) object classes, and the entire system is executed as an HLA federation. Several HLA Interaction classes are defined to exchange information between the clients and server such as client join or leave. Figure 21 depicts the system architecture in the context of HLA.

2.1.6 Time Management and Synchronization

In a sequential simulation-based emulators such as nse [44] the processing of an event is delayed until wall-clock time reaches the timestamp of that event. If the simulator executes ahead of wall clock time, causality errors may occur if a packet is generated externally with a timestamp that is less than the current simulation time of the simulation. In IP-TNE [26], the parallel simulation supported emulation, the node that interacts with a real-time device timestamps an external packet and does not dispatch the packet until the wall-clock time reaches the event's time stamp. In ROSENET, the emulator and application execute at real time (or wall clock time) while the simulator executes at simulation time. Because the emulation cannot be rolled back as in optimistic interactive simulations, the high fidelity simulation must lag behind the low fidelity emulation to avoid causality errors. An *optimistic emulation* approach, in which the emulation executes faster than real time, is impossible because events are generated at real time and

can not be emulated ahead of real time.

```
while( true)
  {
     double endTime = getMinSimulationEpoch();

     updateSimulationTraffic(endTime);

     runSimulation(serverTime_.convertToSimTime(endTime));

     generateLowFidelityModel(endTime);

  }
```

Figure 22: Server Execution Loop

Following this rule, a model-driven synchronization protocol is defined between the client and server. Time information is piggy-backed on network models exchanged between client and server. The client time-stamps the *TrafficSummaryModel* using the first packet's arrival time and the monitoring interval, both in wall-clock time. Figure 22 shows the server execution loop in the simulation process. The server first computes the minimum time from all received *TrafficSummaryModels'* timestamps to get the new simulation end time. It then applies these *TrafficSummaryModels* in the high fidelity simulation and runs the simulation until the simulation end time. After that a *LowFidelityModel* is generated based on the collected measurements from the high fidelity simulation. The *LowFidelityModel* is time-stamped with the first packet's arrival time as well as the last packet's arrival time, which roughly corresponds to the *TrafficSummaryModel*'s time interval. When the client receives a *LowFidelityModel*, it applies the model to the network emulation. As long as the server and client are in the same time interval, they are regarded as synchronized. If during the process the high fidelity simulation lags, it is safer for the emulator to use an old *LowFidelityModel* model than miss the deadline delivering the packet. In this case, however, users should be

informed of the estimation inaccuracy.

2.2 Support for Parallel Discrete Event Simulation

2.2.1 Challenges

In extending ROSENET's architecture in Figure 15 to support parallel discrete event simulation, an intuitive approach will be to allow the emulator to directly interact with each distributed simulator in the federation. However, this approach will cause several problems:

- **Time management**. The emulator executes in real time while the distributed simulators execute in simulation time. In a parallel discrete event simulation, simulation time is maintained locally on each simulator and specific algorithms are used to manage time advances in each simulator. Thus it is rather difficult for the emulator to directly interact with distributed simulators carrying different simulation time clocks.
- **Communication latency.** In ROSENET the simulators and emulators are distributed over a wide area network. If the emulator joins the federation of the simulators, the synchronization among these federates may not be very efficient since they have to wait for the slowest federate with very limited look-ahead to exploit before they can advance.
- **Security.** Distributed simulators running on high performance computing platforms are usually protected by firewalls and provide only limited access to users. Allowing the remote emulator to frequently access each node in the parallel and distributed machines behind a firewall and over a wide area network may

cause difficulties in control and management if security of these machines is not to be compromised.

- **Locality.** Since the target network is partitioned and modeled by different simulators, the source and destination applications to be tested by the network emulation may be mapped as virtual nodes modeled by different simulators. Information about the locality of these mapped nodes in the distributed machines should not have to be maintained by the emulator. A thin client is desirable in this case to reduce the computing resources needed at the client emulator.

2.2.2 Two Federation Design

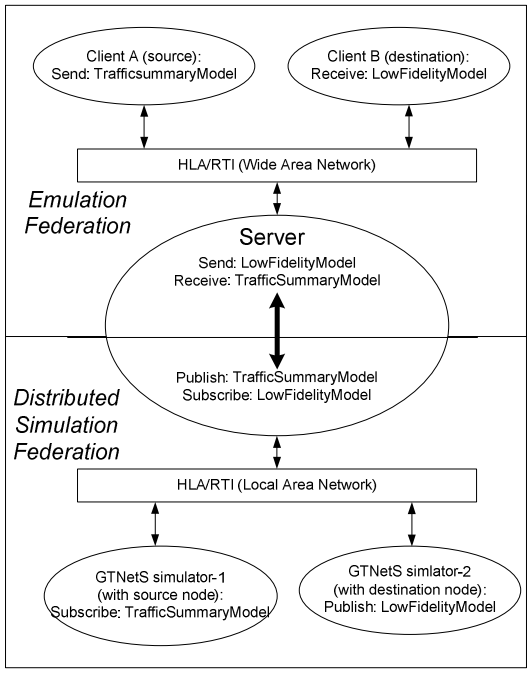

Figure 23: Two Federation Design

An extension of the ROSENET approach with support for parallel and distributed simulation has been developed as shown in Figure 23. It is composed of two federations: the emulation federation and the distributed simulation federation. The server is involved

in both federations and acts as a proxy to provide the following services:

- **Network model exchange.** Two types of network models are exchanged in both federations: *TrafficSummaryModel* and *LowFidelityModel*. The distributed simulation event scheduler on each simulator receives the *TrafficSummaryModel* generated by the remote emulation client and forwarded by the ROSENET Server, updates the corresponding virtual node's traffic patterns in the simulation, and executes the simulation until the end time of all the current *TrafficSummaryModels* in this simulation. Compared with the previous version of the ROSENET system, which had a sequential event simulation on each node, this parallel and distributed version allows source and destination applications to be modeled on different physical nodes that can execute at their own pace. This requires traffic models to be applied at the nodes at the same simulation time, which means they must execute the simulation in the same time interval in ROSENET.
- **Time translation.** In ROSENET, the emulator and test applications execute at real time (or wall clock time) while the simulator executes at simulation time. Since the execution of the emulation cannot be reversed as in optimistic interactive simulations, the high fidelity simulation must follow the low fidelity emulation in order to avoid a causality error that may occur when traffic with time-stamps less than the current simulation time is injected into the simulation. This is achieved by piggybacking time information on network models exchanged between the simulators and emulator. When forwarding network models between the simulation federation and emulation federation, the proxy server also

translates the timestamp on network models from real time to simulation time.

2.2.3 Time Management Protocols

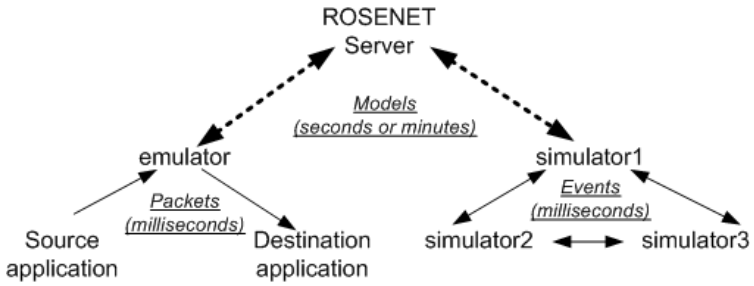

Figure 24: Time Management in ROSENET

In a distributed version of ROSENET, the meaning of time management is two-fold: time management among the distributed simulators and time management between the emulator and simulators. In a distributed simulation, the simulator advances its simulation time through time management services defined in the HLA RTI. Each simulation federate processes events and makes time advance requests to the RTI which grants these requests to ensure no causality error occurs. In this sense, time advance within simulators is fine-grained as the time advance is event-driven, and can be on the order of milliseconds. On the other hand, time management between emulator and simulators is more coarse-grained, meaning that time advance for the emulator and simulators is by time intervals, usually on the order of seconds or minutes. The two-level time management in ROSENET is illustrated in Figure 24.

2.3 Analytical Model for Remote Emulation Delay

Although ROSENET can achieve timeliness by using low fidelity network models at the client, this does not guarantee that the results are accurate because network models

generated based on historical data may not accurately predict future QoS. If traffic patterns remain the same during this period, the emulation results should be accurate. Otherwise, the emulation results will be inaccurate until the network model is updated. Hence a new metric is introduced to measure network emulation performance. This metric is called ***remote emulation delay***. This is the delay incurred when there is a change in the source application's traffic until the time it is reflected in the QoS predictions of the low fidelity emulator. As has been discussed in [40], the overhead in each part of the ROSENET system might result in inaccuracies in the emulation results because the emulator produces end-to-end delay and loss estimates for a packet based on the source application's traffic patterns existing at an earlier point in time. The correctness of the emulation results is directly related to remote emulation delay and an upper bound on remote emulation delay in ROSENET is of particular importance to ROSENET users.

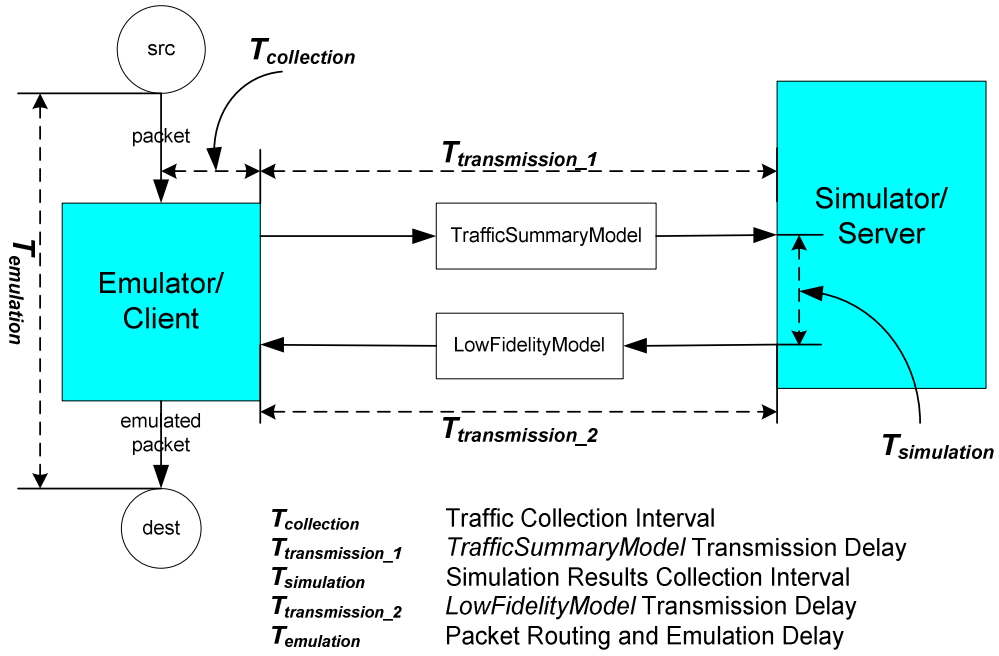

Figure 25: Overhead in ROSENET

Figure 25 shows the time overhead incurred by each component of the ROSENET system. $T_{collection,}$ the data collection interval or model update interval, is the time required to collect source traffic to generate *TrafficSummaryModel* and can be on the order of seconds or even minutes. $T_{simulation}$ is the wall clock time required to execute the simulation for one model update interval in simulation time and can be on the order of milliseconds, seconds or even minutes depending on the model update interval length and the speed of the simulator. Network transmission delays $T_{transmission\_1}$ and $T_{transmission\_2}$ represent the end-to-end delay for network model transmission between the client and server over a local or a wide area network, and usually range between a few to several hundred milliseconds. The packet routing and emulation delay $T_{emulation}$ represents the time needed for a packet to travel from the source application host, through the emulator, to the destination application host. This is expected to happen according to the estimated end-to-end delay and can vary from a few to several hundred milliseconds.

Using the time spent in each component of the system, the remote emulation delay and obsoleteness can be estimated using a simple analysis. Ideally the measurement would be the difference in time from when a packet p_i is generated by the application to the time when the client's *LowFidelityModel* is updated by the high fidelity simulator and based on simulation results which include packet p_i in the simulation. Since network models, instead of individual packets, travel through the system, for ease of measurement, remote emulation delay is measured in the unit of network models instead of a single packet. Hence remote emulation delay is defined as $T_{remote\_emulation\_delay} = T_{collection} + T_{transmission\_1} + T_{simulation} + T_{transmission\_2}$ and obsoleteness as $T_{obsoleteness} = T_{remote\_emulation\_delay} - T_{emulation}$.

Remote emulation delay can be used to predict the performance or accuracy of the ROSENET system, since the accuracy of a packet's QoS predictions can now be quantified using the remote emulation delay value of the *LowFidelityModel* generating the prediction. If $T_{remote\_emulation\_delay}$ is smaller than or equal to $T_{emulation}$, $T_{obsoleteness}$ is zero and users get emulation results that are accurate. However, this requires that the *LowFidelityModel,* based on simulation results that include packet p_i, be available for QoS predictions before packet p_i is delivered to the destination application, This is only an ideal scenario since $T_{remote\_emulation\_delay}$ is usually on the order of seconds while $T_{emulation}$ is on the order of milliseconds.

The existence of remote emulation delay and obsoleteness in ROSENET means that ROSENET is only applicable to network testing scenarios where users can tolerate a certain range of remote emulation delay and obsoleteness in order to gain advantages such as scale, accuracy, timeliness, flexibility, and remote access. ROSENET is not applicable if users require zero obsoleteness in their emulation results. In fact, all testing using existing emulation tools require users to make tradeoffs, such as using abstract network models, testing with small scale network topology, static network status, or inability to remotely access experimental environments. The advantage of ROSENET is that the tradeoff in accuracy can be quantified using remote emulation delay which can be adjusted easily using a few system parameter settings. The experimental results from Section 4.2.4 shows that remote emulation delay is bounded and easily predictable so that users can choose experimental settings accordingly to meet their specific needs for

emulation accuracy.

2.4 Conclusion

The design goal was to allow users to "connect" their simulator, emulator, and applications to ROSENET in order to perform remote network emulation. In order to achieve this, a client-server architecture with support for standard interfaces, network models and HLA/RTI has been designed to support low fidelity emulation and high fidelity simulation. Issues extending this client-server model so that it can support parallel and distributed simulation are discussed. Finally an analytical model is introduced to estimate the remote emulation delay in ROSENET.

3 NETWORK MODELING ISSUES

In the ROSENET framework, simplified light-weight models, instead of single packets, are exchanged between the high fidelity simulator and low fidelity emulator to keep them properly synchronized. The goal of using network models in remote network emulation is to reduce communication overhead without incurring too much cost in accuracy. Due to the Internet's distributed topology and its support for dramatically heterogeneous mixtures of protocols, services and applications, it is very difficult to use a single static model to accurately represent a wide spectrum of network dynamics.

This chapter focuses on the network modeling issues [40], meaning how to create communication and computation efficient network models to characterize the dynamics measured in the high fidelity simulator and regenerate these data in the low fidelity emulator. Section 3.1 discusses background information on network traffic constancy based on which the use of network models in remote network emulation is justified. Then related work on network traffic modeling techniques is explored. Section 3.2 illustrates the flow of data and network models in ROSENET. Section 3.3 describes a library-based modeling methodology for remote network emulation. Section 3.4 presents a network traffic modeling and update approach using system identification. Section 3 discusses experimental results from a network worm propagation simulation. Section 3.6 presents conclusion.

3.1 Background and Related Work

3.1.1 Network Traffic Constancy

The periodic network model update approach used in the remote network emulation is based on the assumption that network traffic remains relatively constant within the update interval and thus can be described by a network model. With this assumption communication overhead between the low fidelity emulation and the remote high fidelity simulation can be reduced without incurring too much cost in accuracy. The validity of this assumption is contingent on the question of the time scales on which network traffic is *constant* (defined below).

According to [42], network measurement can be used to predict the future if the relevant network property remains stable, or exhibits *constancy*. Time scales of three notions of constancy (mathematical, operational, and predictive) are explored for three key network properties: loss, delay and throughput. Mathematical constancy measures whether the network property can be described with a single time-invariant mathematical model. The simplest form of mathematical constancy is the dataset can be described using a single independent and identically distributed (IID) random variable. In general, if the dataset can be describe by a mathematical model with a certain set of parameters, the dataset is consistent with that set of parameters for mathematical constancy. Change-points are identified and a time series of the measured data is partitioned into change-free regions (CFR) for mathematical constancy. Operational constancy evaluates whether the measurement remains within bounds considered operationally equivalent, meaning whether users care about these changes. Predictive constancy checks whether past

measurements allow one to reasonably predict future characteristics or track the changes. For instance, RTT (Round Trip Delay Time) is neither mathematically nor operationally steady but is highly predictable.

The findings from [42] show that the three constancy notions sometimes differ so it is essential to determine which notion of constancy is relevant. Almost all predictors frequently used in networking produce very similar prediction error levels. The steadiness of the Internet depends on the constancy notion and the dataset. By statistically studying the constancy time scale for the three constancy notions, the authors conclude that for packet loss, delay and throughput, one can generally rely on constancy on at least the time scale of minutes. This constancy time scale justifies the assumption in the ROSENET design that network dynamics remain constant within a time interval and that network models, instead of individual packets, can be used to characteristic network dynamics within that interval without incurring too much loss in accuracy.

3.1.2 Network Traffic Modeling Techniques

Network traffic modeling uses parameters to capture and summarize important characteristics of network traffic. Network traffic modeling has been used in many different areas such as congestion control [45], network security [46], and network simulation [47]. The validity of underlying network models is of critical importance as the factors used to evaluate a system are taken directly from the underlying traffic model.

Network traffic modeling can be classified as input traffic modeling and measurement or

output traffic modeling. Input traffic modeling describes traffic generation parameters such as packet size and inter-departure time. Measurement traffic modeling describes traffic properties measured from the network such as packet loss rate, delay and throughput. Network measurements are used to report the current status of the network and can be used to predict future behavior. For instance, the measurement of packet delay and loss has been used for data transmission rate control, and TCP uses end-to-end delay to time-out dropped packets. In this section the focus is on measurement traffic modeling including packet delay and loss.

Poisson processes have been frequently used to model network traffic [48, 49]. The Poisson process fits most network traffic traces reasonably well for short periods. Poisson models are popular in queuing theory because they are memoryless, which means future behavior is not dependent on past behavior and aggregating multiple Poisson streams generates a new Poisson stream. However this model is not very satisfactory because real network traffic exhibits bursty behavior and long range dependences that cannot be modeled by Poisson process. Packet train models [50] and fluid models [51] attempt to add burstiness to Poisson distributions. As pointed out in [52], wide area network traffic is bursty and heavy tailed, which violates the Poisson assumption. The dynamic behavior of the Internet is one of the main reasons that queuing theory and associated statistical analysis approaches have achieved limited success.

Some recent research [52-54] has shown that local and wide area network traffic exhibit variability at a wide range of time scales and that network packet traffic is self-similar.

Self-similarity refers to distributions that exhibit the same characteristics at all time scales and is also called long range dependence (LRD). Several models have been developed to generate self-similar traffic [55, 56] and systems have been created to compute realistic parameters for self-similar models [57]. Although self-similarity applies over large time scales, small-scale correlations may be very difficult. For extremely short samples even Poisson models are accurate. Swing [58] examines characteristics of a range of applications, and use a structural model to reproduce burstiness across a wide range of time scales.

If the details of the applications are known, such as in Swing, network models can use that information to build application-level network models. Application level modeling must consider application-dependent, protocol-dependent, and network-dependent characteristics. RAMP [59] is a tool that uses traces to estimate end-user behavior and network conditions to generate application-level simulation models. The trace file is processed off-line and the application of the model requires knowledge of the type of network applications and the physical structure of the network. The output is statistically represented as an empirical distribution with no correlation information included.

3.1.3 End-to-End Delay Modeling Techniques

Network traffic models have been developed to study how video codec cell arrivals affect network performance. Linear and nonlinear models have been used to predict network traffic to evaluate network QoS and intrusion detection systems. End-to-end delay prediction has been used in real time network applications, especially for multimedia

applications such as VoIP applications to ensure end-to-end QoS.

Queuing theory [60] can be applied to model mean end-to-end delay if the distribution at each individual link is known. This assumption might hold in a small network with a few interconnected servers, but usually not for large networks. [61] uses statistical modeling for end-to-end delay in wide area network. It is found that the delay distribution does not follow normal nor lognormal distribution and a Pareto distribution is used to model the end-to-end delay. [62] uses a multiple model approach to model Internet end-to-end delay. This approach uses AR models and an off-line model set design procedure based on vector quantization and short-term time series analysis. [63] models end-to-end delay in a wide area network using time series. NIST Net [4] uses a multifractal wavelet model (MWM) [64] to model end-to-end network delay patterns. Wavelet models can provide multi-resolution analysis but are costly to use.

[65] proposed a time varying exponential distribution to model end-to-end network delay based on the assumption that network delay can be divided into several states. Within each state, the probability density function of network delay follows an exponential distribution with a determined offset. [66] treated the network seen by specific source-destination host pairs as a black-box, and modeled the end-to-end packet delay dynamics using a system identification technique that is widely used in control engineering.

3.2 Data Flow in ROSENET

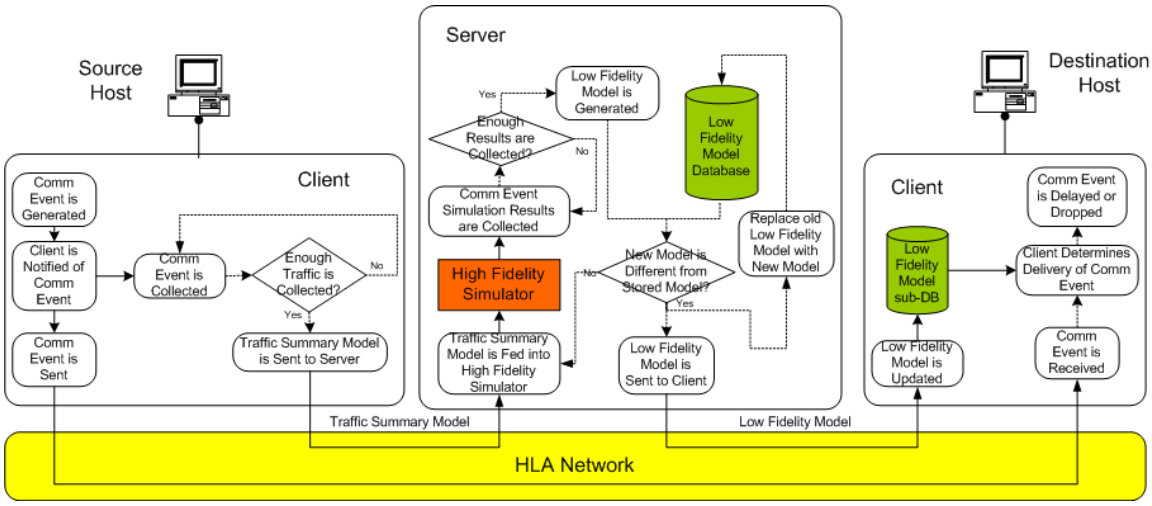

Figure 26: Data Flow in Remote Network Emulation

Figure 26 illustrates the data and model flow in ROSENET using an example of a source host sending a communication event to a destination host. Each emulation client supports one or more applications that can be a message source or destination. The client is composed of several components: a low fidelity simulator, an input traffic collector and *TrafficSummaryModel* generator, an event routing table, a set of *TrafficSummaryModel*s for local applications that are message sources, and a set of *LowFidelityModel*s for local applications that are message destinations. With these components, the client intercepts communication events (such as IP packet or application level messages) from the application and routes them to the corresponding destination application, monitors the input traffic to build *TrafficSummaryModel*, passes the model to the simulation server, updates the *LowFidelityModel* as directed by the server, and delivers or drops communication events according to QoS predictions of *LowFidelityModel* for local applications.

As also shown in Figure 26, the remote simulation server executes a large-scale high fidelity simulation supporting the emulation of the clients. In addition to the high fidelity simulator, the server includes other components: a *TrafficSummaryModel* database, a *LowFidelityModel* database, a configuration management database managing all clients and mapped applications, a *LowFidelityModel* generator, and traffic injector and simulation data set collector attached to the high fidelity simulator. After the server collects all the data for one time interval on end-to-end network performance from the high fidelity simulation, it generates a new *LowFidelityModel*. The *LowFidelityModel* generated by the server during each interval needs to be sent to the corresponding client to update the client's knowledge about the current network status in the simulation. To reduce the amount of network communication, the server does not send the new *LowFidelityModel* to the client unless it decides that this new model is statistically different from the previous one. Its comparison is based on a Kolmogorov-Smirnov statistical test, and could also be performed using other statistical test. The details of the network modeling technique and statistical test for network models are discussed in the following sections.

3.3 Library-based Modeling Methodology

Since network simulators model a wide spectrum of network dynamics, a single technique will not be able to adequately characterize network dynamics. Therefore it is necessary to provide a library of network modeling techniques in the remote network emulation system. *LowFidelityModel* and *TrafficSummaryModel* are the two required network models. *TrafficSummaryModel* is an input model that describes traffic generation patterns while *LowFidelityModel* is a measurement traffic model that

describes network dynamics such as end-to-end delay and loss. Section 2.1.4 describes the network model interfaces defined for these two models and list sample modeling techniques for each model.

In this section a library-based modeling approach focusing on the *LowFidelityModel* is presented. The measurement metrics used are end-to-end delay and loss, which can be directly applied by the network emulator. Since the network simulator periodically updates the packet inter-arrival time and packet size using *TrafficSummaryModel*, models characterizing end-to-end delay are directly used by the network emulator for end-to-end delay prediction.

The Internet transmits best effort data and provides no guarantee concerning the end-to-end transmission delay. Two issues involved in end-to-end delay prediction are the modeling/prediction method and prediction interval. The prediction interval refers to how far in the future the network packet delay can be predicted with a certain confidence interval and error bound. According to [67], Internet delay variations occur primarily on time scales of 0.1 to 1 second, but extend out to quite larger times. [42] has found out that for Internet loss, delay, and throughput one can generally count on constancy (including mathematical, operational, and predictive constancy) on at least a time scale of minutes. The trade off between prediction accuracy and cost is made when choosing a prediction model. Since network emulation has to deal with real time applications, a prediction model with low computation and communication cost is preferred.

When modeling the end-to-end delay in remote network emulation, it is assumed that the high fidelity simulation provides valid representation of the network traffic and is used as a means of comparison with network models for end-to-end delay. When end-to-end delay is collected from the high fidelity simulation, clock synchronization is not required because simulation time is used in high fidelity simulation and the advance of time is managed by distributed simulation protocols.

Findings from large scale studies of Internet packet dynamics [67, 68] indicate that a flexible model that can adapt to a wide range of network behaviors, instead of treating any aspect of packet dynamics as typical, is desirable in network traffic modeling where no assumptions are made about the properties of the network. Due to the Internet's distributed structure and its support for dramatically heterogeneous mixtures of services and applications, it is difficult to describe the Internet properties using any single linear model. As discussed in Section 3.1.1, the Internet has constancy at certain time scales. Thus the network traffic within a time interval should be modeled using a network model selected from a network model library according to traffic patterns and application requirements.

The main challenge in the library-based modeling approach is to design the set of models that are required to cover all traffic delay patterns at different time scales. Therefore this library includes different modeling techniques such as queuing models, statistical models, empirical models, time series models, wavelet models, and application-level models. Users can also introduce their own modeling techniques into the library as model

candidates. The model is then selected according to criteria as follows:

- **Accuracy.** The model accuracy is obtained by using training and test data to generate models and the prediction errors (such as minimum mean square error) are then compared among different models to choose the model with the least prediction error. Users can specify accuracy thresholds so that only models that meet the accuracy requirements can be used.
- **Measured data properties**. By analyzing the input data, linear or non-linear models may be selected. Models that are suitable to describe short time scale constancy or long time scale constancy are selected accordingly depending on the constancy of the data set and user's constancy requirements.
- **Computational or communication cost**. If two models are similar in their prediction accuracy, the model that requires less computation to generate and to predict or takes less space to store and to transmitted is preferred.

The model selection process works as follows. First the measured data is pre-processed and analyzed to find the types of models that best fit the data's properties. Then different models and parameters are tested on the data using pre-defined accuracy measurement such as prediction error within certain range. The computational and communication cost may also be considered in choosing the model, in addition to the model accuracy criteria.

After the model is selected, the model is then shipped to the remote emulation client to generate predictions. The life time of the model is first decided by the model's constancy range as discussed earlier. When new measured data is collected, statistical tests such as

Kolmogorov-Smirnov Test, F-Test, or T-Test can be used to check if the last model is still valid in describing the new data, meaning whether the model's parameters need to be updated, or new model type needs to be selected for changed traffic patterns.

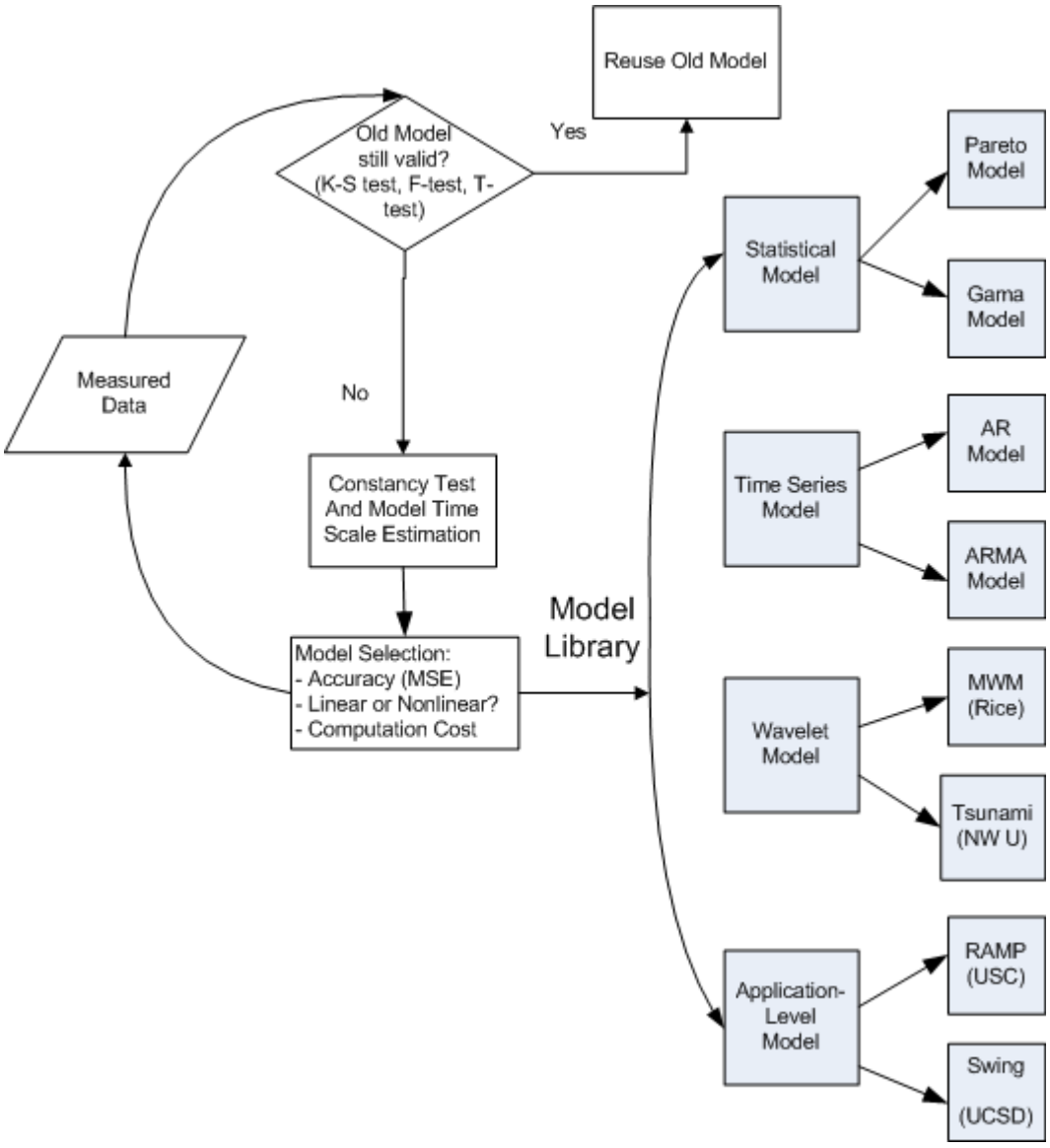

Figure 27: Library-based Modeling Approach

To reduce computation costs, model selection is triggered only if a pre-set error threshold is reached. A model is first selected at the beginning with a full-scale search, meaning

every model in the library is examined to find the model achieving the best fit. When a first-choice model is selected, a sub-set of secondary choice model candidates are also marked with priority scores so that later if the model needs to be reselected, the secondary model subset may be searched first to see if they meet the accuracy requirements. Figure 27 illustrates the library-based modeling approach. Given the complicated and time-varying nature of network dynamics and the limits and strength of different modeling techniques, it may be reasonable to use a hybrid modeling approach where the Internet is modeled using a combination of several network models at different time scales as was done in [69].

3.4 Traffic Modeling Using System Identification

3.4.1 Black-box ARX Model

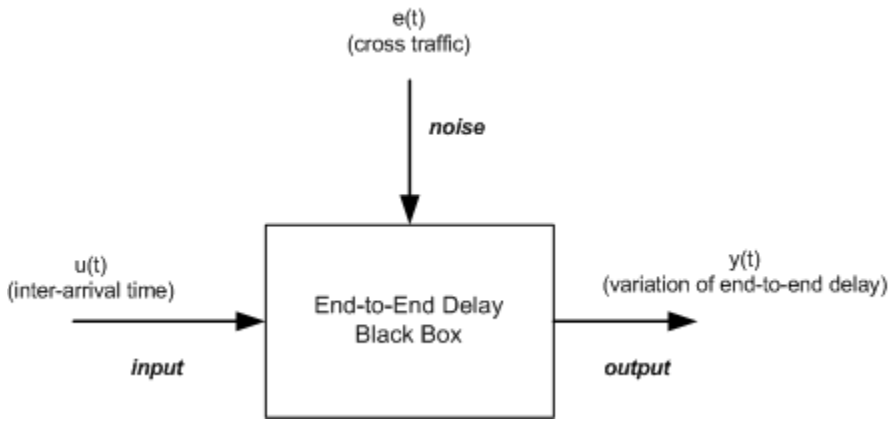

Figure 28: End-to-End Delay Black Box Model

As shown in Figure 28, in the server-based emulation system, each source-destination host pair views the rest of the network as a black box. The traffic through this black box is described by the *LowFidelityModel*, which is generated by the simulation server based on data measured in the high fidelity simulation. A black-box model describes the end-

to-end delay between a source-destination host pair in which the input (represented as *u(t)*) is the inter-arrival time measured from the source host, as inter-arrival time directly affects the end-to-end packet delay if the network bandwidth is not unlimited and shared by multiple source-destination pairs as in the system. The output of the system (represented as *y(t)*) is the difference in end-to-end packet delay between two consecutive delays. [42] has shown that at a packet-level time scale the aggregated network traffic is not stationary. Thus by using the difference of consecutive delays as output, instead of the delay itself, the non-stationary effects from the disturbance of the unpredictable background cross-traffic on the measured end-to-end packet delay are cancelled.

This black-box model can be described using System Identification, which constructs mathematical models of a dynamic system from observed data. System identification is widely used in control engineering. It applies to very general models, among which the most basic dynamic models are linear difference equation descriptions. Equation 1 is a linear ARX (Auto-Regression with eXternal signal) model, in which *y(t)* represents the output at time *t*, *u(t)* as the input at time t, and *e(t)* as the noise at time *t*.

Equation 1

$$y(t) + a_1 y(t-1) + \ldots\ldots + a_{na} y(t-na) = b_1 u(t-nk) + b_2 u(t-nk-1) + \ldots\ldots + b_{nb} u(t-nk-nb+1) + e(t)$$

Other no-input black box models are available such as the AR (Auto Regressive) model. However, here input is needed because the simulation results are affected not only by network traffic history, but also by end-user behaviors such as changes in packet inter-

arrival time which may be different at the time of network dynamics synthesis. For the same reason, curve-fitting cannot be used as it does not consider the effects of input. It can be predicted that the black-box model accuracy may not be as good as those models whose internal structures are known in advance as is the case in a queuing model. Compared with other approaches in network traffic modeling, the system identification approach is topology independent and thus is well suited for the client-server simulation architecture.

The basic system identification procedure includes six steps: (1) collect input-output data, (2) examine the data to remove outliers, (3) select and define a model structure to use such as the linear model ARX and choose the model order range (parameters such as na and nb in equation 1), (4) compute the best model from the model structure using a criteria of fit, (5) validate the model, and (6) examine the properties of the model. In the current implementation, the model is built using the ARX model as the model structure. A model order range is selected within which a best fit is found. The first half of the collected data is used to generate the model, and the model is then validated using the second half. Here, the MATLAB system identification toolbox [70] is used to execute the system identification functions.

3.4.2 Model Generation and Update Algorithm

Using the difference equations in the system identification approach, the end-to-end delay of a packet can be predicted in the near future, but not in the far future because the end-to-end delay may be disturbed by unpredictable cross-traffic. To capture the network

dynamics as closely as possible, the measured data is divided into "chunks" and within each chunk a black box model is generated to describe this chunk's network dynamics such as end-to-end delay. At the end of each chunk an ARX model describing the end-to-end delay difference and the first end-to-end delay are used together as the *LowFidelityModel* for the simulation client to synthesize end-to-end delay using the current packet inter-arrival time as input. To further reduce network traffic between clients and server, *LowFidelityModel* will be updated only if the server decides that the old model cannot represent the new measured data. The model generation and update algorithm is shown below:

Simulation server:

1. *collect all the data in chunk 1, generate a LowFidelityModel, and pass it to the client*
2. *collect all the data in chunk i (i>1), test to see if model i-1 is still valid for data in chunk i. If model i-1 is valid, repeat step 2 for chunk i+1. Otherwise, go to step 3*
3. *generate a LowFidelityModel based on data chunk i, and pass it to client*

Simulation client:

1. *obtain a LowFidelityModel from server*
2. *generate end-to-end delay with packet arrival event (inter-arrival time as input to LowFidelityModel)*
3. if new LowFidelityModel is received, the old one is replaced with the new one in step 2

3.5 Experiments and Performance

This section presents the results of experiments evaluating the accuracy of the client-server system using a worm propagation simulation. A packet-level simulator called GTNetS [19] is used for the high fidelity model. The purpose of these experiments is to evaluate the accuracy of the client emulations. The end-to-end delay is the principal metric of interest. The end-to-end delay generated from the low fidelity emulation at the client is compared with that generated directly from the high fidelity simulation in the simulation server, in order to justify that by using the modeling methods, the simulation results from the local client is statistically as good as those from the server (high fidelity simulation).

3.5.1 Experimental Setup

The Georgia Tech Network Simulator (GTNetS) is a network simulation environment designed to study the behavior of moderate to large-scale networks. GTNetS uses a federated approach to create parallel simulations. Using GTNetS, a worm propagation simulation was developed to demonstrate the effect on web traffic [71]. The worm propagates as a shooting agent in an infected node, makes a connection to a vulnerable server on a randomly selected host, and infects it. During the worm propagation, normal background traffic is injected into the network. The infected node follows the preceding steps to propagate to other hosts in the network. The worm connection is TCP-based.

In these experiments, the network topology is a clique with 10,240 nodes (routers and end hosts) spread across 10 processors (1024 nodes on each processor). On each processor

there are two web servers and twenty web browsers (total of 20 web servers and 200 web browsers). In all the experiments, the worm, after being set loose, propagates at the rate of one iteration every second. Experimental measurements were performed on a cluster-computing platform at Georgia Tech. This cluster is a Linux cluster consisting of 17 machines. Each machine is a Symmetric Multi-Processor (SMP) machine with eight 550MHz Pentium III XEON processors.

3.5.2 Measured Data Analysis

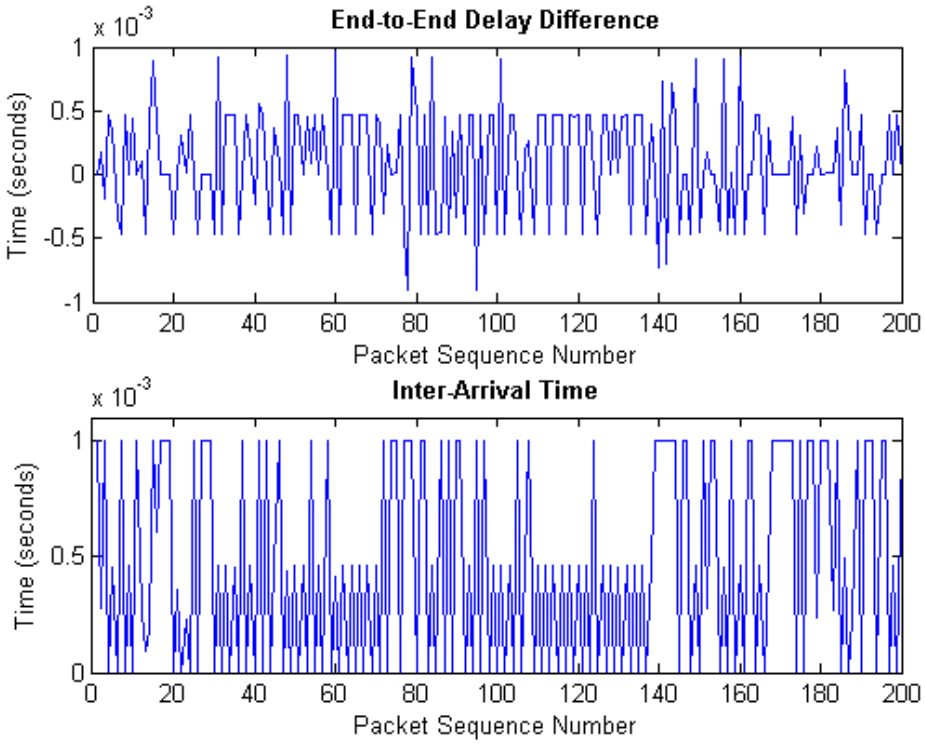

Figure 29: Measured inter-arrival time and end-to-end delay difference

Figure 29 gives a snapshot of the measured data obtained from the TCP-version packet-level (high fidelity) worm propagation simulation. Three thousand packets are collected at one of the hosts in the simulation. For clarity, only the first 200 packets are depicted

here. The inter-arrival time is obtained by taking the difference between consecutive packet send times, the end-to-end delay is calculated as each packet's send time subtracted from its receive time, and the end-to-end delay difference is the difference in consecutive packets' end-to-end delays.

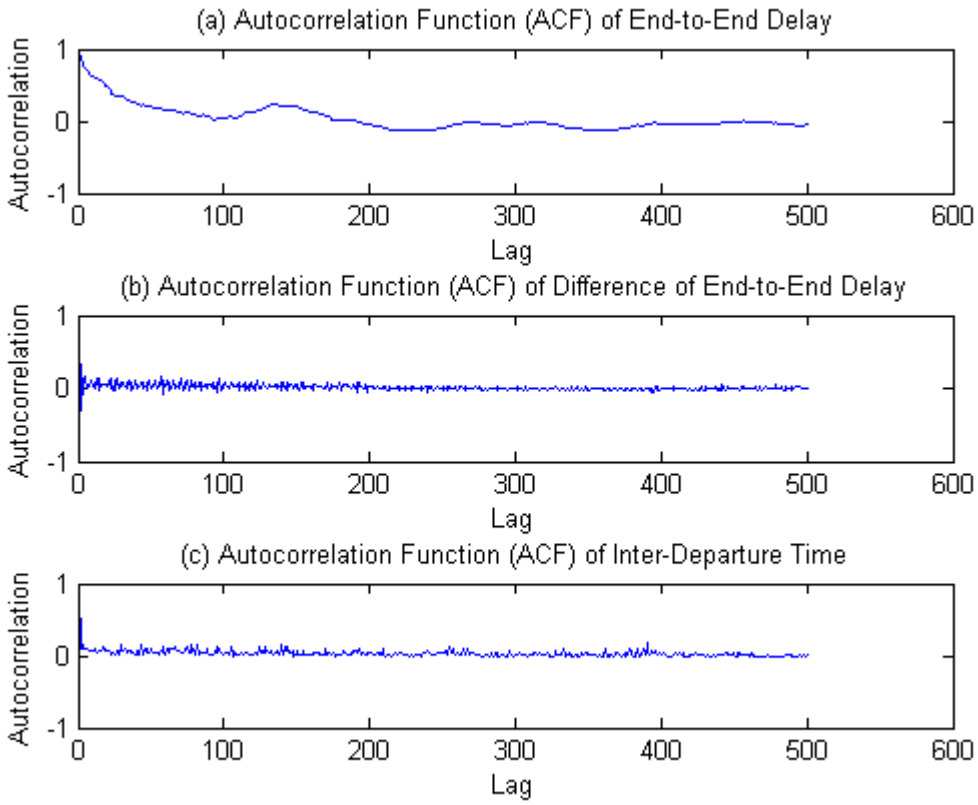

Figure 30: Autocorrelation Function of Measured Data

The constancy of the data is checked before it can be modeled by a network model. Figure 30 shows the autocorrelation function of the data collected from the GTNetS simulation against corresponding lags. The autocorrelation function measures the dependence between observations as a function of their time differences or lag. A stationary series exhibits statistical properties that are unchanged as the period of

observation is moved forward or backward in time. Figure (a) shows that end-to-end delay has a significant correlation and is non-stationary. Figure (b) shows that after taking the difference of consecutive end-to-end delay, the time series becomes stationary. Figure(c) shows that the packet inter-departure time is stationary.

3.5.3 Experimental Results

The model generation and data prediction procedures are as follows. The data in Figure 29 is divided into two chunks, each containing 100 packets. The first 100 packets (inter-arrival time as input and end-to-end delay difference as output) are used in generating the ARX model. The parameters of *na*, *nb*, and *nc* in Equation 1 are set at 10, 20, 1. The coefficients in Equation 1 are estimated using the first 100 input/output points and the least square methods. Then the ARX model is used to predict the next 100 end-to-end delay differences, using the next 100 inter-arrival times as input.

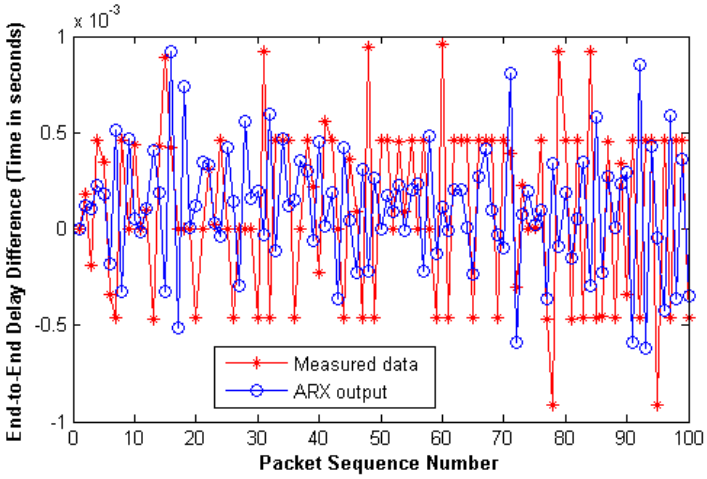

Figure 31: Measured Data from Simulation [101, 200] and Output from ARX Model Generated from Data [1, 100]

Figure 31 shows the measured end-to-end delay difference of the next 100 packets for the

worm propagation simulation, and the predicted end-to-end delay difference using the ARX model generated from the first 100 input-output pairs. The comparison verifies that the accuracy of predicted end-to-end delay difference using ARX is close to that of the collected data from the high fidelity worm propagation simulation. This suggests that it is possible to use the system identification method to define *LowFidelityModel* for the network traffic within a certain time interval. The advantage of using system identification is the small amount of information needed to be transmitted in the network, which is composed of the parameters in Equation (1). This is much more efficient than a CDF table and more precise than pure mean and standard deviation methods.

As stated earlier, in order to reduce the amount of traffic between emulation clients and the simulation server, the *LowFidelityModel* is not updated at the end of every time interval as long as the model can still be used to predict future end-to-end delay. Another experiment is performed to determine to what extent the ARX model can predict traffic in the worm propagation simulation. In the previous experiment shown in Figure 31, input-output data indexed from 1 to 100 is used to generate an ARX model, predict results for data pair 101 to 200, and compare the predicted results with measured data from 101 to 200. In this experiment, the ARX model is generated from data pair [1, 100] and then used to predict data pair [201, 300], [301, 400], and [401, 500]. The results show that the accuracy of the prediction using the ARX model decreases as the input-output data pair is farther away from the data pair that generates the model. This indicates that *LowFidelityModel* built using the system identification method can be reused when the network dynamics do not change too much, but need to be updated after a certain amount

of time since it is possible the network status changes with time.

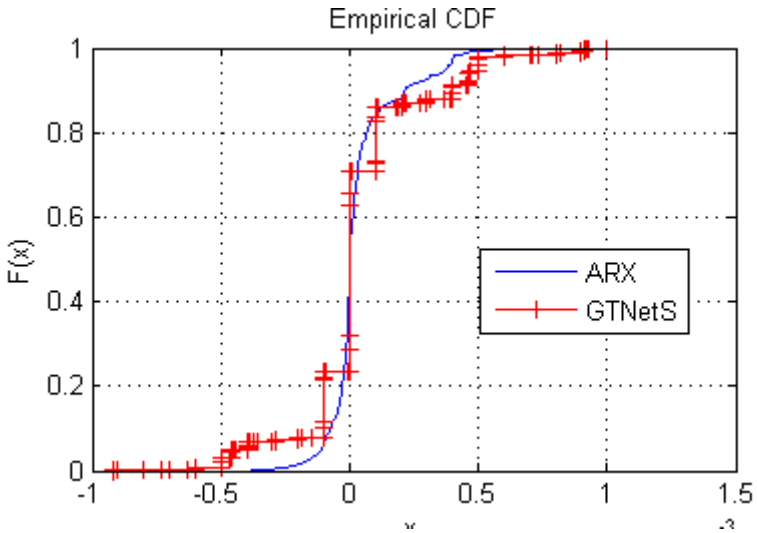

Figure 32: CDF Plot of measured data from GTNetS and prediction from ARX

Figure 32 is the Cumulative Distribution Function (CDF) plot of all the measured data (2800 packets) from GTNetS simulation and predicted end-to-end delay difference using ARX model predictions. With these data, the Jarque-Bera test is used for goodness-of-fit to a normal distribution test with the hypothesis H0 that GTNetS/ARX has normal distribution at level $\alpha = 0.05$. Both null hypothesis were rejected at the 5% level. This means tests based on normal assumptions such as t-test, f-test cannot be used to see whether the mean of the two groups of data are the same. Wilcoxcon rank sum test with the hypothesis H0 that $\mu\_GTNetS = \mu\_ARX$ at level $\alpha = 0.05$ is accepted, which means the two populations are identical with the same mean. The Kolmogorov-Smirnov test of these data at level $\alpha = 0.05$ is accepted with the assumption that the measured and predicted data have the same distribution. All the CDF plots and statistical tests reveal that the generated data using ARX model are statistically similar to the original measurements from GTNetS simulation.

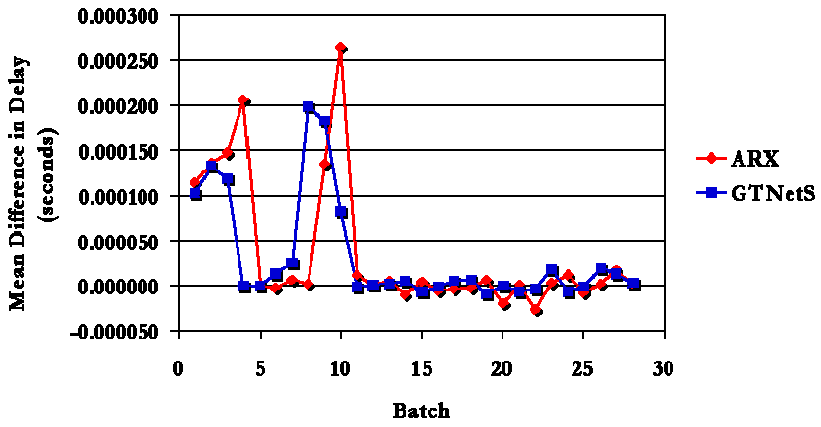

Figure 33: Batch Mean of Delay Difference

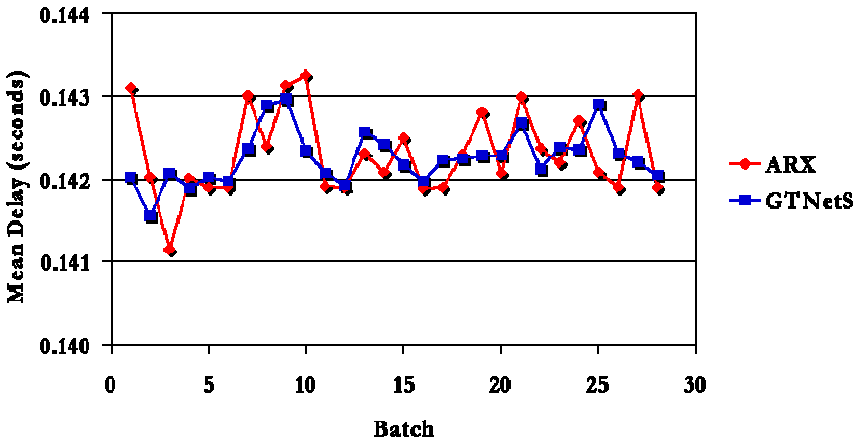

Figure 34: Batch Mean of Delay

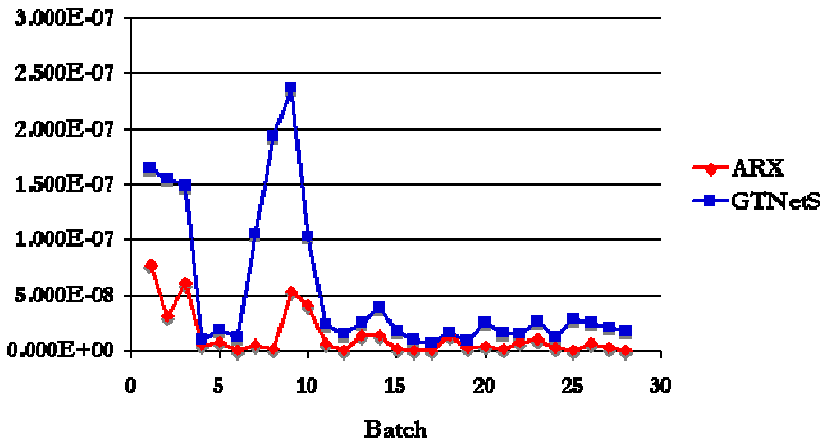

Figure 35: Batch Variance of Delay Difference

Since the data is modeled in batches, the measured data from GTNetS and predicted from ARX model are compared for each batch containing 100 data points, rather than the entire data series. Figure 33 shows the batch mean for delay difference. Figure 34 shows batch mean for delay, which is calculated from the delay difference and the delay of the first point in each batch. Figure 35 shows the batch variance of delay difference. In all the figures, the predictions from ARX model track the changes in the GTNetS simulation results very well.

3.5.4 Discussion

Although the results from the worm propagation simulation are good, system identification does not apply to all traffic scenarios. As seen from Equation 1, there is a limit on the number of previous inputs and outputs that can be used to predict the future output. When the variance is sharp and large, it cannot be captured through Equation 1. Another requirement is a not-too-weak correlation between input and output, since a model generated using the history data is used to predict future data, with only the input

to provide the information about the current status. As indicated in [70], exponential input is an ideal case for the system identification method to apply.

The overhead introduced by the network metrics modeling using system identification is mainly due to the invocation of the MATLAB engine used to compute ARX model coefficients in Equation 1, and communication with the MATLAB engine to import collected data and export computed coefficients. The MATLAB engine invocation is a one-time cost paid when the first model is generated and will not occur for future model generations. Data imports and exports happen every time a new model is to be computed by the MATLAB engine. The imported data are a relatively large data array in the size of tens to hundreds of inter-departure times and end-to-end delay measurements. The export from the MATLAB engine is the coefficients in Equation (1) with varied size ranging from single to double digits. Given the fact that the MATLAB engine runs as a process local to the Simulation Server, the invocation and communication costs are on the order of microseconds.

3.6 Conclusion

In ROSENET, a remote high fidelity simulation continuous calibrates a local low fidelity emulator through network models generated from online simulation data collection. The validity of the design of using periodically updated network models, instead of individual packets, to keep the simulation and emulation consistent is justified, based on studies on network constancy time scale. Existing network modeling techniques are explored and a library-based modeling approach is proposed to model the wide spectrum of network

dynamics which is difficult to describe using a single static network model. The procedures of using a system identification tool and state-divided black-box model to model end-to-end delay in remote network emulation are illustrated. Using a worm propagation simulation, it is shown that the end-to-end delay in the server-based architecture can be accurately modeled using the system identification tool but note that it does have limitations in its application to network traffic modeling.

4 PERFORMANCE EVALUATION AND CASE STUDIES

4.1 Baseline Emulation Performance Evaluation

The performance of traditional network emulation systems are straightforward to evaluate because traditional emulation tools are composed of either a single node machine or a homogeneous cluster and support only a limited number of application types and network scenarios. By comparison, the performance of a ROSENET system is much more complicated to predict as it integrates general distributed emulation and simulation tools into a heterogeneous environment. Hence its performance depends on many factors such as the specific simulators and emulators, application and simulation settings, and the distributed heterogeneous computing environment.

In order to verify that ROSENET can generate accurate emulation results while meeting real-time constraints for network emulation, a series of experiments [72, 73] were performed to examine ROSENET's performance as a network emulator, through a specific implementation of ROSENET using the NIST Net [4] emulator and GTNetS [19] simulator. Experimental results [72] examining end-to-end delay and loss show that ROSENET can deliver timely results and accurately predict the QoS parameters of network traffic.

4.1.1 Experiment Setup

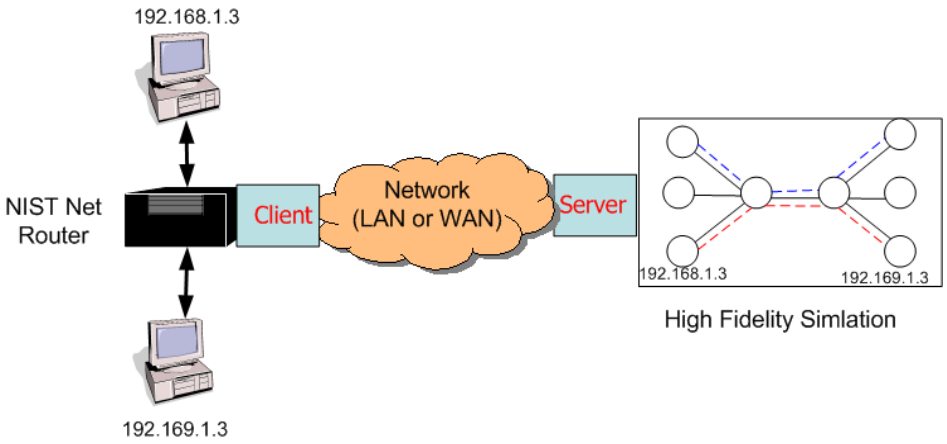

Figure 36: Experiment Setup

The overall experimental setup is shown in Figure 36. GTNetS [19] is used as the high fidelity simulator due to its large scale distributed simulation capability and its support for simulation in C++. In these experiments the GTNetS simulator runs on a single node machine since the performance of parallel and distributed version of GTNetS has been examined in [25] and will not be explored here. NIST Net [4] is used as the low fidelity emulator. NIST Net is a link-level emulator which can apply delay, jitter, and loss to all individual IP packets. The NIST Net kernel module is modified so that it can monitor specific source-destination traffic pairs and pass the statistics to the emulation client, which runs as a user-level process. The emulation client communicates with the remote server and dynamically updates the models NIST Net uses to generate emulation QoS predictions.

The modified NIST Net emulator also serves as a router and forms a private network with

the two test machines. In Figure 36 the source test machine marked with IP address 192.168.1.3 sends UDP packets to the destination test machine with IP address 192.169.1.3. NTP protocols are used to synchronize the clocks and use ethereal [74] to collect the traces on both machines. The precision of the NTP protocol is half the round-trip time between the two synchronized machines, which yields the clock precision to be within 1 millisecond.

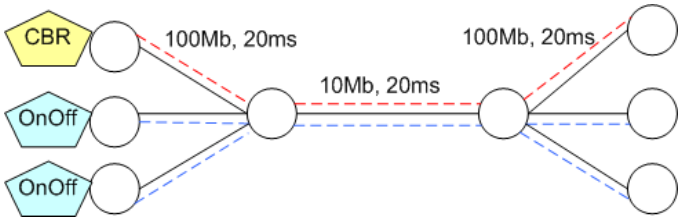

Figure 37: Simulation Network Topology Setup

A simple dumbbell topology is used in the initial experiments as shown in Figure 37. The constant bit rate (CBR) application is mapped to test applications on the two test nodes. As background traffic, more node pairs with On/Off applications are introduced. The test application CBR traffic and background On/Off traffic share a bottleneck link. The On/Off applications have exponential distribution with the On/Off time duration set at a 2.5 seconds interval time. During the On period, the On/Off application sends a constant bit rate stream of 10Mb/s. The simulation results such as end-to-end delay and loss are collected on the run at the end of each simulation time interval and generalized as model updates to the NIST Net emulator.

A major concern in validating ROSENET as a network emulator is to ensure that packets are subject to the right end-to-end delay and loss. For this purpose, the end-to-end delay from the GTNetS simulation is logged, together with the end-to-end delay collected from

the destination test machine with IP address 192.169.1.3. Since this emulation system is based on time-intervals, the end-to-end delay cannot be compared packet by packet between emulation and simulation to measure emulation inaccuracy. Therefore the focus is on how well the emulation responds to changes in simulation results.

4.1.2 End-to-End Delay

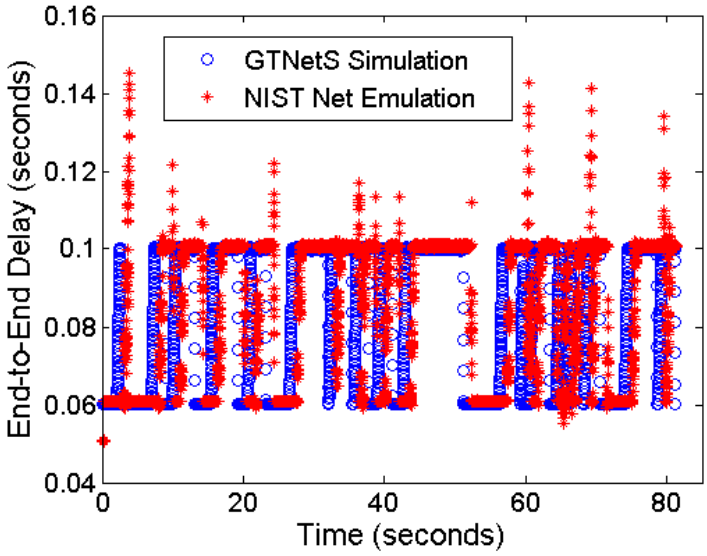

Figure 38: End-to-End Delay

Figure 38 shows the end-to-end delay for 10,000 UDP packets of size 64 bytes at a sending rate of 180 pkts/second. Models between client and server are updated at 20 pkts/interval. The circle represents the results collected in the GTNetS simulation and the star represents the results from NIST Net emulation, collected at the destination host (192.169.1.3).

Three major observations can be obtained from this figure. First, the emulation traces the changes in the simulation results closely with a small lag. This lag is caused by the fact

that the emulator is using *LowFidelityModel* from the previous interval to predict end-to-end delay in the current interval. Therefore the changes in simulation will be reflected in emulation with at least one update interval delay. The second observation is that the delay of emulation is always slightly higher than that of simulation. The higher delay in emulation is caused by the fact that the estimated end-to-end delay from *LowFidelityModel* is applied to the NIST Net router without compensating for the overhead outside NIST Net such as the transmission delay from the source node to the destination node. This overhead is a small amount and is mostly constant. Therefore, this overhead should be compensated when NIST Net does the delay estimation for the test machines. For this reason, another set of experiments were conducted to measure the overhead in NIST Net emulator in the following section.

The last observation is that when a sudden change in end-to-end delay takes place, the simulation results only have a small variance while the emulation results have a large variance. The sharp change in simulation happens when the background traffic of the On/Off application has an On period so that the end-to-end delay suddenly increases in the simulation due to queuing at the node near the bottleneck link. This traffic with sharp change is then modeled by the server using *LowFidelityModel* as a Normal distribution with a large variance. When NIST Net uses this *LowFidelityModel* with large variance to estimate delay, it does not know that this variance is actually caused by one large increase or one large decrease. Instead it will regenerate the delays with many positive and negative variations in that time interval. After that sharp change interval, the server will generate constant delay with almost zero variance and the emulator will accordingly

regenerate delay with little variance.

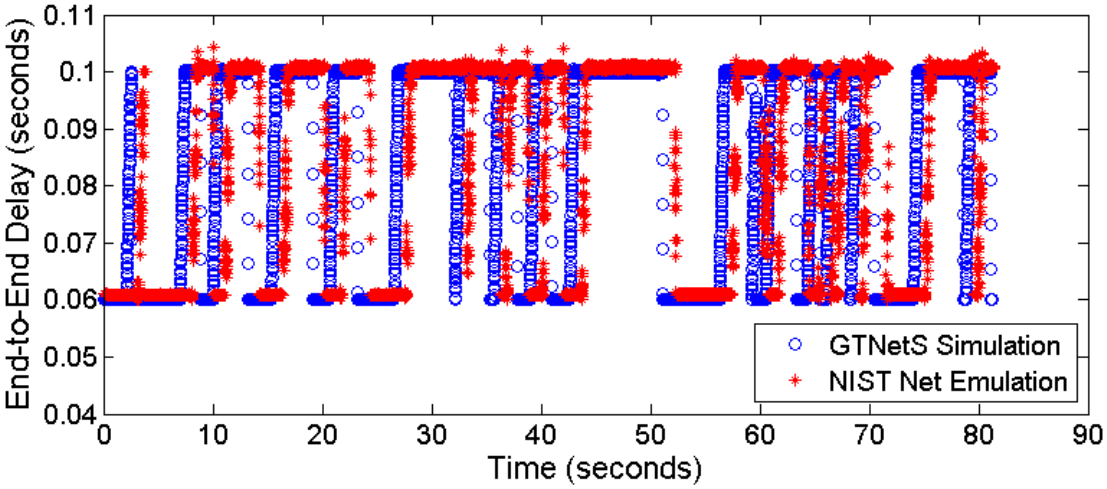

Figure 39: End-to-End Delay with Truncated Normal Distribution

This prediction inaccuracy caused by a sharp change in end-to-end delay suggests directions for improvement. For example, in this particular case, a *LowFidelityModel* can be created which regenerates only one large positive or negative change instead of many positive and negative changes in one time interval. However, this solution requires users to visually compare the original data from simulation with predictions from emulation to identify points that may not be meaningful in a particular network traffic scenario, and thus may not apply to other network traffic patterns. A more general solution is to use a *LowFidelityModel* with a truncated normal distribution, meaning to use the maximum and minimum values obtained from the original data from the simulation as parameters in the *LowFidelityModel* to limit the predicted data value range. Figure 39 illustrates the same end-to-end delay results from Figure 38 using a truncated normal distribution for the *LowFidelityModel.* It can be found that these out of range predictions, although quite obvious in the figure, only accounts for a small percentage of the predictions (167 packets out of 10,000 packets), which shows that the predictions of end-to-end delay is

quite accurate.

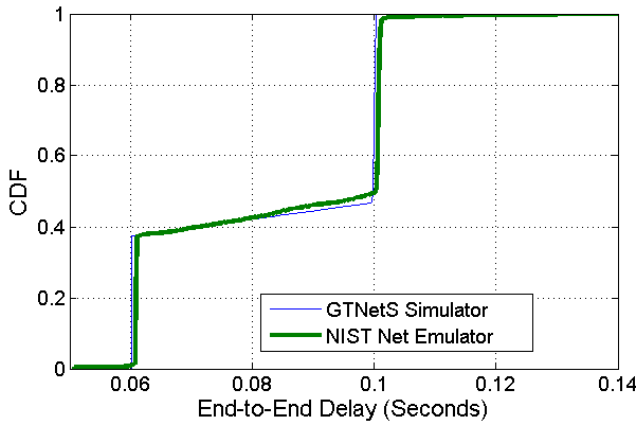

Figure 40: CDF of End-to-End Delay

In addition to the plot of end-to-end delay over time as in Figure 38, the end-to-end delay's cumulative distribution functions for both simulation and emulation are compared as shown in Figure 40. The CDF plot of the end-to-end delay further confirms the analysis of the three observations from Figure 38. It can be seen that the CDF of the simulation's end-to-end delay closely matches that of the emulation, the simulation's end-to-end delay value is slightly smaller than that of emulation which is caused by the unconsidered overhead outside NIST Net's emulation module, and the emulation has delay values larger than 0.1 or smaller than 0.06 because one sharp change of end-to-end delay in the simulation causes several positive and negative changes in the emulation results.

4.1.3 Overhead

As seen from Figure 38, the NIST Net emulation always has a slightly higher estimation of the end-to-end delay than the simulation. This is because the results from simulation are directly applied to NIST Net delay with no compensation for the transmission delay

between the test machines. Therefore another group of experiments are performed to see how much extra overhead the ROSENET system introduces into the emulation. In this set of experiments, the CBR UDP application was run on the two machines through NIST Net and ethereal collecting the end-to-end delay statistics. As comparison, ping is used to collect the RTT between the two machines under different conditions. We then compared the overhead in the following test cases:

1. NIST Net is not running on the router
2. NIST Net is running with no parameters such as delay or loss set
3. NIST Net is running with the end-to-end delay set as zero and the kernel monitoring of packets for the client to generate *LowFidelityModel* turned on
4. NIST Net is running with delay set as 60 milliseconds and the kernel monitoring of packets for the client to generate *LowFidelityModel* turned on

Time (msec)	**Ping (RTT)**	**Client/Server (end-to-end delay)**
1. No NIST Net	1.131	0.606
2. NIST Net, no delay, etc.	1.181	0.669
3. NIST Net, delay 0, monitoring	1.187	0.630
4. NIST Net, delay 60, monitoring	61.347	60.7

Figure 41: Emulation Overhead

Figure 41 shows the round trip time (RTT) measured using ping and end-to-end delay measured from the UDP application traffic using ethereal. Since the NTP protocol is used to synchronize the clocks on the two test machines, the time precision of the clock is half the round-trip time. Therefore from the figure the precision for one-way delay is within 1 millisecond in this case. These results show that very little overhead is introduced into the NIST Net emulation by ROSENET.

Another group of data might also be included, which is the overhead for NIST Net with delay 60 milliseconds with traffic monitoring and model updates from the server. However, the model update strategy is optimized so that if there is very little change in the model parameters, no model will be sent over the network for update. Therefore this case is not studied here. From this figure the compensation in NIST Net delay estimation is estimated to be around 0.7 milliseconds, which is the difference in end-to-end delay between simulation and emulation in Figure 38.

4.1.4 Loss

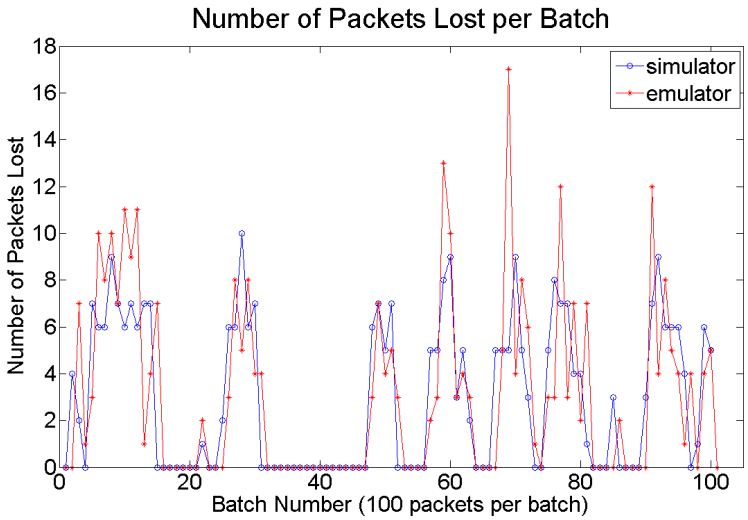

Figure 42: Packet Loss Plot

To investigate how accurately the loss rate can be predicted in ROSENET, the same settings are used to collect the packet traces from both emulation and simulation as was done previously. Then the packets are grouped into batches according to their packet id and the number of packets lost in each batch is calculated and shown in Figure 42. The x axis is the batch index and y axis is the number of packets lost in that batch. In total

10,000 packets are collected and divided into 100 batches with 100 packets in each batch.

Similar to the results from end-to-end delay in Section 4.1.2, it can be seen that the change in loss rate in emulation lags slightly behind that in simulation, which is caused by the fact that the emulator uses the simulation results from the previous interval to predict packet loss. The emulation also seems to have a larger loss rate than the simulation. To make sure this loss rate inaccuracy is not caused by NIST Net itself, the same CBR application is run without the server as the loss rate of NIST Net is set at 5% and the end-to-end delay at 60 milliseconds. It is found out that NIST Net can correctly drop packets at the 5% loss rate.

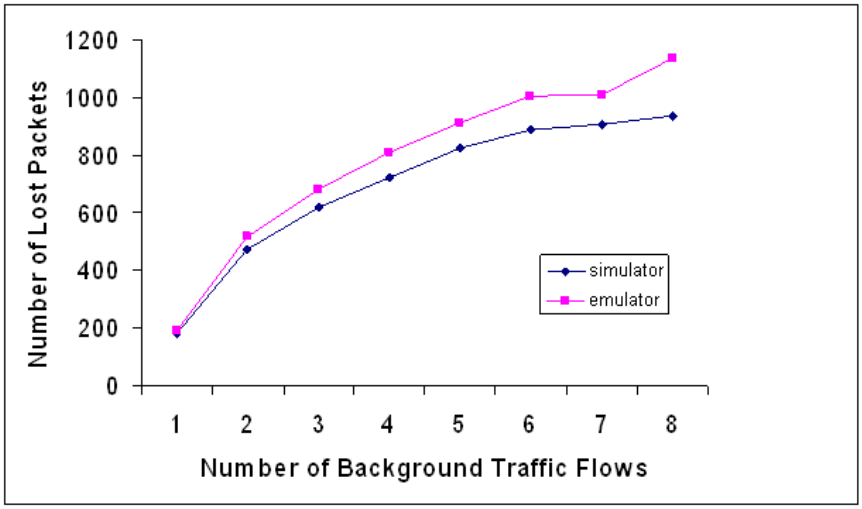

Figure 43: Packet Loss over Background Traffic

In order to determine the real reason why the loss rate from simulation and the measured loss rate from emulation are different, another group of experiments are performed by increasing the number of background node pairs in the dumbbell topology from 1 to 8. Then the total number of lost packets are collected from simulation and emulation with different number of background traffic flows (background node pairs) when 10000

packets are sent from the source to destination. As shown in Figure 43, the emulation consistently loses more packets than the simulation. As more traffic flows are introduced into the simulation network topology, this difference in loss rate becomes larger.

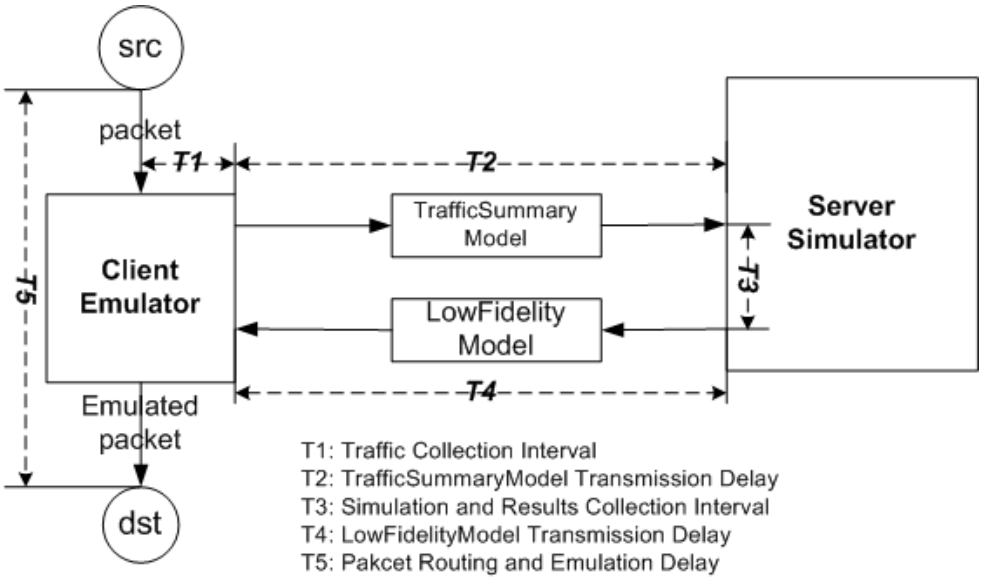

Figure 44: Update Interval Delay in the System

In order to explain the reason emulation loses more packets than simulation and the increased difference with more background traffic in simulation, the update time interval lifecycle in ROSENET originally introduced in Section 2.3 needs to be used. As shown in Figure 44, after a number of packets are monitored by the client during time interval T1, a *TrafficSummaryModel* is sent to the server using time T2. It takes T3 for simulation time interval T1 to be simulated in the simulation, and T4 to send the generated *LowFidelityModel* back to the emulator. T1, T2 and T4 remain constant in all the experiments. But T3, the time to simulate for simulation time interval T1, varies since it takes more time to simulate the same simulation time interval when more background traffic flows are to be simulated.

When T3 becomes larger, the loss rate in a time interval with more background traffic flows will also be higher since the application shares the bottleneck link with other traffic flows. By the time this *LowFidelityModel* with higher loss rate is sent to the NIST Net emulator, the previous *LowFidelityModel* has been used for too long. Since the duration of the On/Off period for background traffic is 2.5 seconds and the time interval is much smaller than that, the time intervals with higher loss rate tend to come consecutively, which means the higher loss rate *LowFidelityModel* tends to be used by the NIST Net emulator for a longer time than the models with lower loss rate.

When the number of background traffic flows is small, the effects of increased simulation time T3 do not significantly affect the results. However, with more flows as background such as in the 8 background node-pairs case, the effects of increased simulation time are not negligible. This suggests that traffic load in simulation may affect the accuracy of emulation. The solution to this problem will be to use a more powerful machine that can tolerate the background traffic load change, or to use fluid models to simulate background traffic to reduce the effects of increased background traffic on emulation accuracy.

4.1.5 Sending Rate

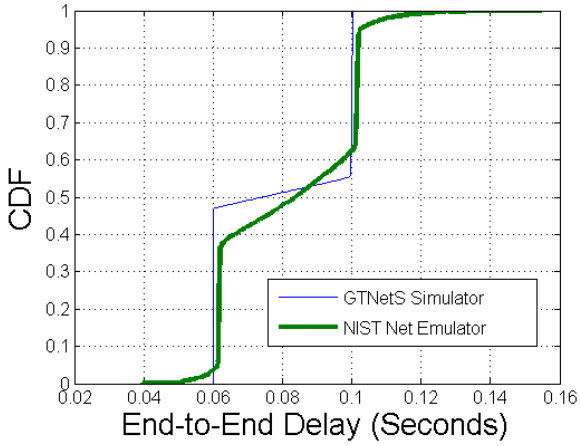

Figure 45: CDF of End-to-End Delay for 20 pkts/s Sending Rate

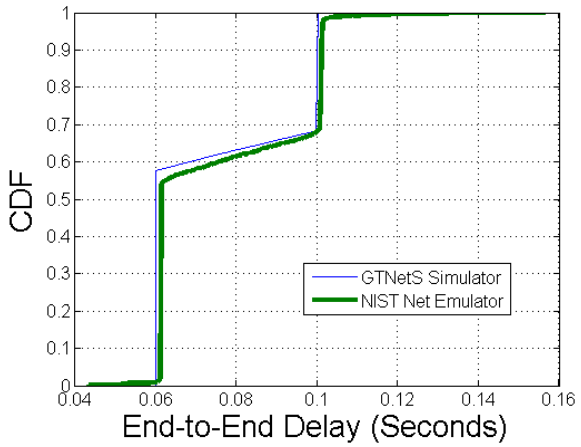

Figure 46: CDF of End-to-End Delay for 200 Pkts/s Sending Rate

In order to explore the effects of application traffic on the accuracy of emulation, the same experimental settings are used but the number of packets sent per second by the source application are varied. Figure 45 and Figure 46 show the cumulative distribution function (CDF) of end-to-end delay with the sending rate at 20 packets per second and 200 packets per second. In total 10,000 packets are sent in each experiment. It can be seen that with a higher sending rate, the prediction from NIST Net emulation is closer to

that of GTNetS simulation.

To explain this prediction accuracy difference caused by sending rate, the end-to-end delay for the first 1000 packets with 20pkts/s and 200pkts/s sending rate is plotted. Figure 47 shows that with 20 pkts/s sending rate, the end-to-end delay from simulation varies a lot from 0.06 second to 0.1 second, causing the delay estimation from emulation to vary dramatically, both positively and negatively as previously analyzed from Figure 38. In contrast, as shown in Figure 48, if the sending rate is 200 pkts/s, the end-to-end delay does not change very much and the emulation results do not have as many variances. Hence the reason the cumulative distribution function of end-to-end delay from Figure 45 and Figure 46 differs much more in the lower sending rate than in the higher sending rate. This indicates that prediction accuracy in ROSENET may be affected by traffic pattern in the application.

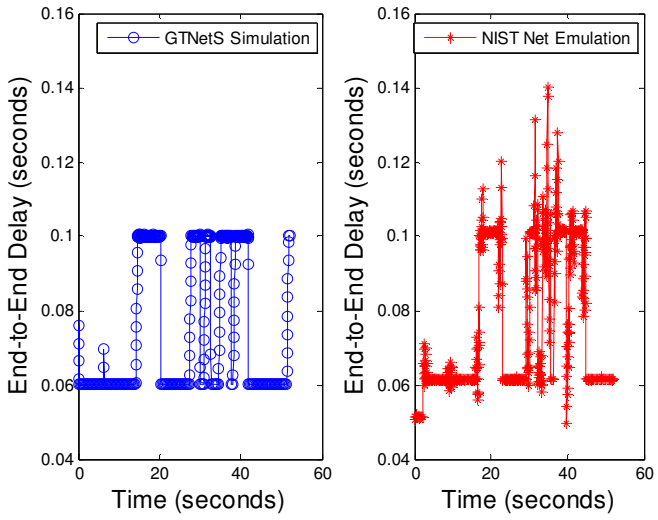

Figure 47: End-to-End Delay at 20 pkts/s Sending Rate

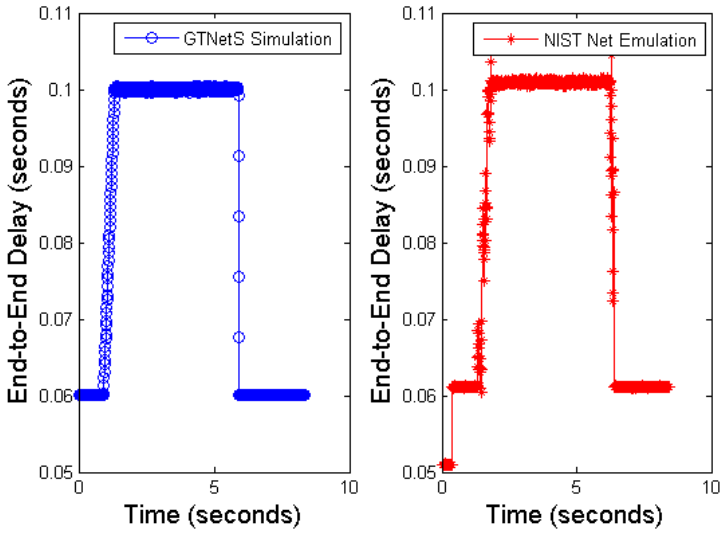

Figure 48: End-to-End Delay at 200 pkts/s Sending Rate

4.1.6 Conclusion

The experimental results examining the accuracy in regard to end-to-end delay, loss, and sending rate show that ROSENET can accurately predict QoS parameters while meeting real time constraints for network emulation, which confirms that ROSENET provides a promising approach to network emulation supporting accuracy and scale while meeting real-time constraints.

4.2 Applying ROSENET to Defense Applications

Many of today's military services and applications run on geographically distributed sites. Before these services and applications can be deployed in an actual network, they need to be tested and evaluated under realistic scenarios with many unpredictable factors. This section discusses the challenges faced in applying ROSENET to defense applications. A case study applies synthetic traffic workloads over DARPA's NMS network topology for a large scale simulation and a metric called remote emulation delay

is defined to evaluate and quantify ROSENET's performance.

4.2.1 Military Network Emulation

Modern military operations are becoming increasingly more reliant on network communication and connectivity. Network centric warfare is expected to provide information superiority in modern warfare, which translates into war-fighting advantages over adversaries. However, information technology may fail to work as expected. In one Navy SEAL mission in 1983 [75], a soldier in Grenada had to call for air support using commercial landlines because of failures in military communication. More recently, in the Iraq War in April 2003 [76], ground forces suffered from lack of bandwidth and range as well as software lockup problems which rendered their computer system useless. Soldiers had to stop their vehicles to receive data, making them easy targets for enemy fire.

Therefore, before new military services and applications are deployed in an actual network environment, it is extremely important to test and evaluate them under a wide variety of network scenarios to determine possible unexpected system behaviors. A typical military communication scenario involves heterogeneously interconnected networks in a possible hostile setting that supports a large number of users and multimedia traffic with severe, critical, real-time requirements. The network design, configuration, and deployment problems in such a domain are extremely challenging.

ROSENET can be used to test and evaluate distributed services and applications, including modern military applications [77, 78], by integrating remote parallel simulation

servers with local network emulators. Sample military applications where ROSENET may be applied include:

- **Information Assurance in the Global Information Grid.** The Global Information Grid aims to integrate DoD's information systems, services, and applications into a seamless, secure, and reliable information environment to achieve information superiority over adversaries and form the basis for the network centric warfare doctrine. Tools are needed to ensure reliable delivery of critical messages, e.g., Call-for-fire messages, or medical evacuation orders.
- **QoS in wireless networks in urban environments**. Mobile applications operating in urban environments are becoming increasingly more important in military operations. People and vehicle movements, weather and terrain (e.g., high-rise buildings) are important and often unpredictable factors that have strong impacts on Quality of Service (QoS) for wireless communications. The ability to integrate different simulators into one framework and make them accessible to remote users are useful features, e.g., to provide real time analysis and control of networks.
- **Realistic communication in military training.** The need for collective and joint training is increasing as a result of the transformation to network centric warfare [79]. A testing framework capable of providing realistic communication scenarios over a secure wide area network at geographically distributed training sites will provide greater realism for joint tactical training.

4.2.2 Experiment Setup

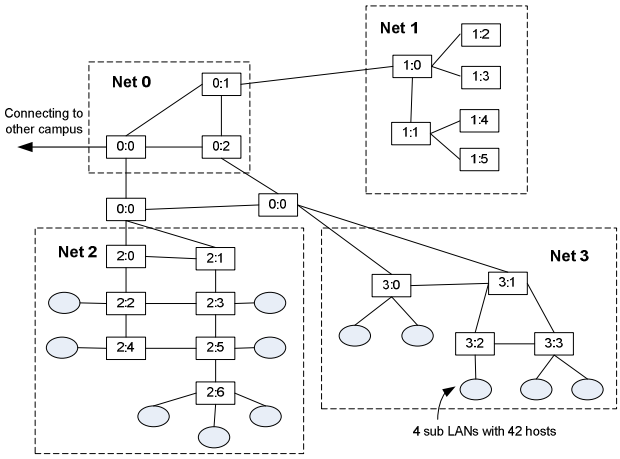

Figure 49: Basic DARPA NMS Campus Network

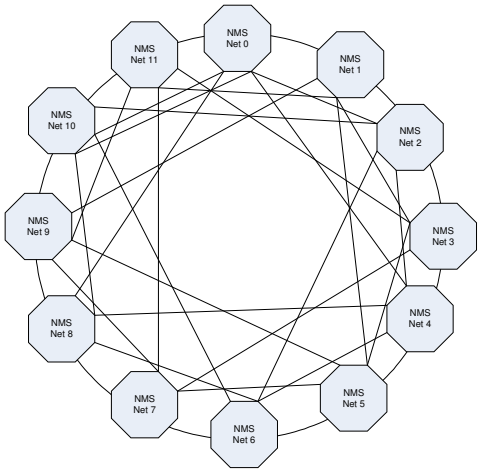

Figure 50: Ring Network Topology with Chords of NMS Campus Networks

In this case study synthetic traffic are applied over a network topology which is from DARPA's NMS program's baseline network model [28]. This network topology was originally designed to support large scale simulation, motivated by the Department of Defense's (DoD) Global Information Grid (GIG) efforts [80]. GIG is designed to be built upon a ubiquitous network supporting a wide range of defense applications. Before these applications are deployed, it is necessary to study their behavior in operational conditions

in GIG. As shown in Figure 49, each NMS campus network consists of four subnets, several routers, and a number of servers and clients generating background traffic. In total 574 nodes including 504 leaf nodes are modeled in each NMS network. All the NMS networks connect as a ring with chords connecting every other, 4^{th} and 10^{th} nodes in the ring. Figure 50 gives an example with 12 NMS networks.

The ROSENET system is implemented using distributed GTNetS simulator and NIST Net emulator. The hardware configuration to run the simulation experiment is a 40-node HP Integrity (2 x 900 MHz Itanium 2 IA-64) running Red Hat Enterprise Linux 4. Up to 32 nodes of the cluster are used in the experiment and each processor in the cluster simulates one NMS network. The source and destination applications tested by the emulator are mapped as leaf nodes on two campus networks and each is modeled on a different processor sending and receiving constant bit rate UDP packets. As background traffic, each leaf node in the NMS Campus network generates On/Off application traffic with data rate of 100kb/s to a random server in the neighboring NMS network simulated on another processor. In order to reduce the simulation load to catch up with the execution of the network emulator at real time, after sending a certain number of packets the connection terminates. Each application starts sending data at a random point in time to spread the background traffic over the simulation period. The emulator and simulator are placed within the Georgia Tech campus and the end-to-end delay between them is measured as within 2 milliseconds.

4.2.3 Measurement Metrics

Network simulators are usually designed to achieve ***accuracy*** and ***scale***. Accuracy is of major concern since users would use an experimental tool to test their applications if that experimental tool can accurately model the real world network environment. The baseline performance of ROSENET measuring the emulation accuracy in regards to end-to-end delay and loss has been evaluated and validated in Section 4.1 through a sequential simulation version of ROSENET. In addition, larger scale network simulation is preferred since it better represents the real world network environment. Parallel discrete event simulation has been used to improve the scale of network simulation and techniques are being developed to scale network simulation to even larger scale. However, scalability of a network simulator is not the focus in this study since scalability is more relevant to a particular simulator integrated in ROSENET. For example, the scalability of GTNetS simulator using the NMS Campus network topology has been studied in [19, 25]. On the other hand, for traditional network emulation, scalability has another meaning, which measures the throughput of the network interface of a network emulator. Similarly, for ROSENET this is more of interests to a particular network emulator such as NIST Net and therefore will not be discussed here.

Timeliness is required because network emulators must process events and deliver results to applications while meeting real time deadlines. Although efficient execution of simulation is required, no strict real-time constraints are imposed on the execution of network simulation. When network simulation is used for network emulation as in

ROSENET, timeliness becomes a major concern. Timeliness can be measured using a timeliness ratio or miss ratio, which is the percentage of packets that meet the real time deadlines, or in other words, their predicted end-to-end delay. In ROSENET, the network emulator client can use network models to quickly generate packet QoS predictions and is unlikely to miss the real time deadlines, given that only simple mathematical calculations are involved.

Although ROSENET can meet the timeliness requirement by using network models, the validity of this remote emulation method depends on the validity of the network models. By validity it means that if the network model describes the current network traffic status, the system is valid. A ROSENET system based on network models far in the past may not generate meaningful results; in the worst case results will be comparable to a trace-based emulation system or some existing emulation tools which use static network parameters for the entire emulation. The following experiments examine more realistic and dynamic network scenarios and also bound the tradeoff of accuracy in time in a controllable and predictable manner.

Therefore the remote emulation delay metric introduced in Section 2.2.3 is used as the measurement metric. The analysis in Section 2.2.3 shows that remote emulation delay is composed of data collection (or update interval) $T_{collection}$, transmission delay $T_{transmission\_1}$ and $T_{transmission\_2}$, and simulation cost $T_{simulation}$. The lower bound of remote emulation delay should be one data collection or model update interval $T_{collection}$, which means the QoS predictions for a packet will come on time but will be based on network traffic status

at least that late in the past.

In the following sections, the contributions to remote emulation delay by each component in the ROSENET system are analyzed and the upper bound of remote emulation delay in each scenario is identified. Section 4.2.4.1 studies how data collection affects remote emulation delay. Section 4.2.4.2 discusses the relationship between remote emulation delay and various model update interval values. Section 4.2.4.3 analyzes how transmission delay between emulation client and simulation server affects remote emulation delay. Section 4.2.4.4 evaluates the effects of the scale of distributed simulation on remote emulation delay.

4.2.4 Experimental Results

Three groups of experiments were performed measuring remote emulation delay under different scenarios. As shown in Figure 25, remote emulation delay is the sum of time spent for the emulation client to collect data from applications in one time interval to generate a *TrafficSummaryModel*, send it over a LAN or WAN to the simulator to simulate for one time interval in simulation time, and for the simulator to generate the *LowFidelityModel* to send it back to the emulation client to perform QoS predictions. Since this process is repeated continuously over the simulation period and is thus executed in parallel by the distributed simulators and emulator, the actual remote emulation delay value may be less than the sum of all the time consumed in this process, due to time overlap at different components of the system.

4.2.4.1 Baseline Remote Emulation Delay

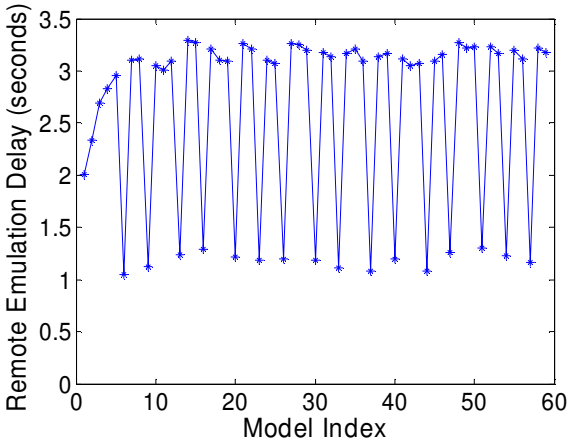

Figure 51: Remote Emulation Delay in Normal Execution

First the remote emulation delay distribution in one execution in the ROSENET system is examined. The GTNetS simulation executes on 8 processors, each of which simulates one NMS Campus network containing 574 nodes and each background traffic connection terminates after sending 50,000 bytes. Figure 51 shows the remote emulation delay value under normal execution conditions, which means all the steps in Figure 25 are performed with the data collected by the emulation client for 2 seconds, sent to the simulator to simulate, and the results sent back to the emulator. The figure illustrates how remote emulation delay changes as model updates (with 2 seconds update interval) over a period of 120 seconds. The average remote emulation delay per model update is 2.5745 seconds. As shown in this figure, when the execution starts, the remote emulation delay values are somewhat different from the values that come later, due to the simulation warm-up period which creates and initializes the large network topology totaling 4592 nodes. After the simulation becomes stable, the remote emulation delay value varies between 1.2 and 3.2 seconds alternatively. This is because the network emulator intercepts packets and stores

the data in the operating system kernel. To avoid overhead in context switches and the cost in copying data from kernel to user space, kernel read is not performed very frequently. Therefore data may be accumulated if one kernel read fails to get any data and the next read may get more than one update interval data accumulated, thus the remote emulation delay value of 1.2 and 3.2 seconds, instead of an ideal condition of 2 seconds.

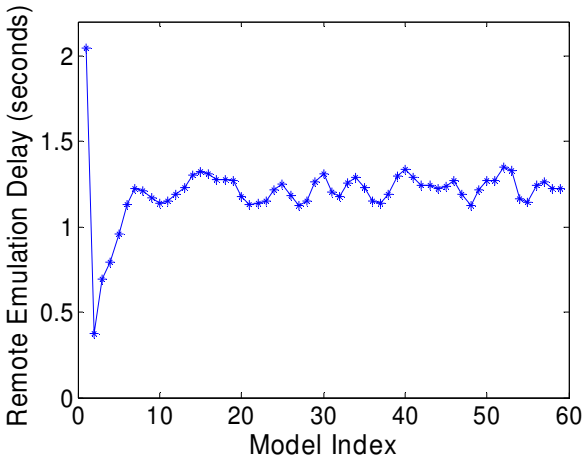

Figure 52: Remote emulation delay with Pre-Stored Data

Since data collection takes at least 2 seconds to generate a model for 2 seconds' traffic and is much larger than the cost in other stages of the model update process, another group of experiments are performed, skipping the data collection step by using previously collected and stored data for model generation to avoid the data collection cost with time $T_{collection}$. The results using pre-stored data in Figure 52 show that after the initial warm up period, the remote emulation delay value remains almost constant at around 1.25 second, unlike the two sided values of 1.2 and 3.2 seconds in the normal condition. This is because when all the data has been previously collected and ready to be read from kernel, remote emulation delay is only composed of transmission delay $T_{transmission\_1}$ and $T_{transmission\_2}$, and simulation cost $T_{simulation}$. The average remote emulation delay with pre-

stored data from Figure 52 is 1.2014 second, which is larger than the 0.5745 seconds as one might expect by subtracting the 2 seconds data collection time from the 2.5745 seconds average remote emulation delay measured in Figure 51. The reason for this is that the average remote emulation delay is obtained by averaging among all the remote emulation delay values including the simulation warm-up period. The fact that the smaller remote emulation delay in Figure 51 is around the same (around 1.2 second) as the stable remote emulation delay value in Figure 52 confirms the previous analysis that remote emulation delay is the sum of the cost in different components of the system and it demonstrates that data collection (2 seconds) is the main cost in remote emulation delay, taking off which will dramatically reduce the average remote emulation delay.

From this group of experiments, it can be seen that data collection time $T_{collection}$, which is usually on the order of one update interval time or seconds to minutes, is a major contributor to the remote emulation delay in ROSENET. Since data collection time is fixed and not avoidable in ROSENET, it is a predictable factor and actually dominates the remote emulation delay value as will be demonstrated in the following experiments. The experimental results also indicate that data collection could affect remote emulation delay and the following section studies how data collection or model update intervals affect remote emulation delay.

4.2.4.2 Remote Emulation Delay vs. Update Interval

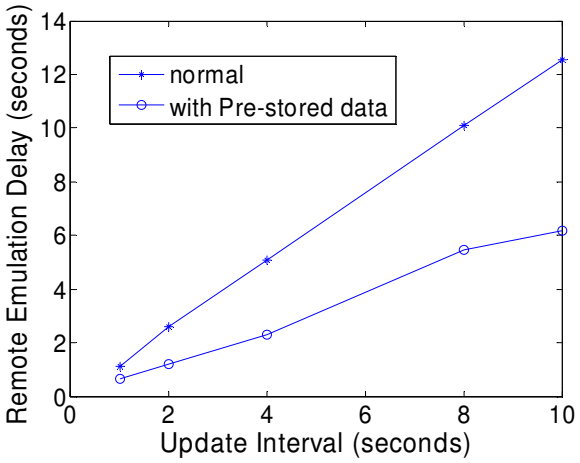

Figure 53: Remote Emulation Delay vs. Update Interval

Update interval (seconds)	1	2	4	8	10
Remote emulation delay normal (seconds)	1.1321	2.5745	5.0932	10.0873	12.5258
Remote emulation delay with pre collected data (seconds)	0.6526	1.2014	2.2863	5.4813	6.1936

Figure 54: Data for Remote Emulation Delay vs. Update Interval

Figure 53 shows the average remote emulation delay with different model update intervals ranging from 1, 2, 4, 6, 8 and 10 seconds for both normal execution and with pre-stored data. In both cases remote emulation delay increases almost linearly as models are updated at longer intervals, or less frequently. The results are also listed in Figure 54 as a table, from which it can be seen that remote emulation delay in normal condition with data collection stage is mainly determined by update interval, which agrees with the observation from Section 4.2.4.1.

On the other hand, based on the results in Section 4.2.4.1 it was expected that remote emulation delay with pre-stored data should remain almost constant with different update interval values since there is no data collection. This is against the observation made from Figure 53. This linear increase in remote emulation delay without data collection is caused by the increase in simulation cost $T_{simulation}$ in Figure 25. With longer update interval, simulation interval also becomes longer and more events need to be processed, thus larger $T_{simulation}$. If the network topology is small, the traffic flow is small, or the processor is sufficiently fast, this cost may not increase significantly even with a larger simulation interval. However, in the case with 574 nodes on each processor for 8 processors and each leaf node sending data to a remote node on another processor, the increase in simulation time caused by longer simulation time is not negligible.

By comparing the two remote emulation delay curves in Figure 53, it can be seen that remote emulation delay is dominated by the update interval so that the impact of the increase in simulation time $T_{simulation}$ is not observable. In that sense, it can be concluded that ROSENET can tolerate some lag of the simulation due to the fact that the simulation and data collection occur simultaneously at different parts of ROSENET. This also means that the high fidelity simulation may not be required to execute as fast as real time and the real time requirement on the simulation can be relaxed according to different update interval values. This relaxed simulation speed requirement may be addressed in future work.

4.2.4.3 Remote Emulation Delay vs. Remote Access Delay

In order to evaluate how well ROSENET can support remote network emulation, an artificial end-to-end delay is introduced between the emulation client and simulation server to mimic the scenario when a network emulation client or user access the simulation cluster over a local or wide area network. According to [81], most (85%) of the round-trip delay in the Internet varies from 15 to 500 milliseconds, which translates to an end-to-end delay ranging from 7.5 to 250 milliseconds assuming symmetrical delay each way. Even if the delay is not symmetrical, an end-to-end delay with 7.5 to 500 milliseconds is a safe estimation. Therefore end-to-end delay in this range is introduced between emulation client and simulation server and collect the remote emulation delay value changes versus the end-to-end delays. This end-to-end delay value corresponds to the transmission delay of $T_{transmission\_1}$ and $T_{transmission\_2}$ in Figure 25.

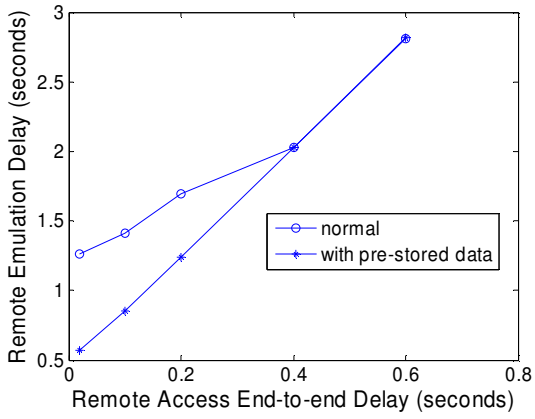

Figure 55: Remote Emulation Delay with Varying End-to-End Delay between Emulation Client and Simulation Server (2machines, 1 second update interval)

First the experiments are performed under challenging scenarios to measure remote emulation delay. The update interval is set as 1 second and it is expected that this is the

lower bound of update intervals in a remote access scenario. The simulation is performed on two machines with very limited background traffic to measure the baseline overhead. Figure 55 shows that the average remote emulation delay changes with the end-to-end delay between the simulation cluster and emulation client. In comparison, the remote emulation delay is collected with pre-stored data as in previous experiments, meaning data collection step is skipped by using previously collected and stored data for model generation. As the figure shows, remote emulation delay increases with end-to-end delay in both scenarios and the increase is linear with pre-stored data.

The figure also shows that when the remote access end-to-end delay is small, the remote emulation delay value in normal conditions is larger than in a pre-stored scenario since the update interval is the major cause of remote emulation delay. Starting from 0.4 seconds end-to-end delay, remote emulation delay becomes the same in both cases whether data collection is included or not. This is because when the end-to-end delay ($T_{transmission\_1}$ and $T_{transmission\_2}$) is 0.4 second, the round-trip delay (0.4 second *2) plus the simulation cost $T_{simulation}$ which is measured as 0.2 second, is approximately 1 second. The 1 second data collection time is thus overlapped by the model transmission time and simulation cost as illustrated in Figure 25.

When the model transmission delay (or end-to-end delay) is larger than 0.4 second, the model transmission delay over the wide area network, not the update interval, becomes the major contributor in remote emulation delay in ROSENET. Since the update interval is likely to be much larger than 1 second and the end-to-end delay over Internet is usually

smaller than 0.4 second, this is the worst case scenario performance for remote emulation delay with varying end-to-end delays.

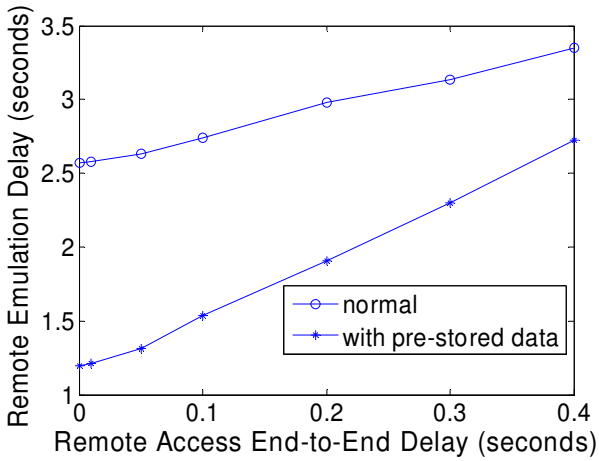

Figure 56: Remote Emulation Delay with Varying End-to-End Delay between Emulation Client and Simulation Server (8 machines, 2 second update interval)

Having tested the extreme scenarios where end-to-end delay can dominate the remote emulation delay value, another group of experiments are performed to examine how remote emulation delay will change over end-to-end delay under not-so-extreme conditions. Figure 56 shows the remote emulation delay value changing with varying end-to-end delays between the simulation cluster and emulation client with 8 machines and 2 seconds update intervals. The background traffic has the rate 100kb/s and the connection terminates with a limit of 50,000 bytes stream starting at a random time during the simulation. As the figure shows, remote emulation delay increases with the end-to-end delay in both scenarios and the increase is linear. With larger end-to-end delay the difference in remote emulation delay between the normal scenario and pre-stored data becomes smaller since larger end-to-end delay has a greater impact on the remote

emulation delay with pre-stored data scenario. This agrees with the observations from Figure 55 but the remote emulation delay under normal and pre-stored scenarios in this case won't be the same under this end-to-end delay range, due to larger overhead in the simulation cost $T_{simulation}$.

This group of remote emulation delay versus end-to-end delay experiments for remote access show that end-to-end delay could affect the remote emulation delay to a certain extend, but the impact is small and can be tolerated due to the fact that the upper bound for the wide area end-to-end delay is usually smaller than the lower bound of update interval in ROSENET. Even under challenging scenarios where end-to-end delay becomes the main contributor of remote emulation delay, the remote emulation delay is still bounded and within a reasonable range due to bounded end-to-end delay. If the wide area network latency falls out of the range of the 85% of the round-trip delay in the Internet (15 to 500 milliseconds), such as larger than 500 milliseconds, the network emulator will continue to use the previous network model until the new network model comes, which should be no later than the end-to-end delay of the Internet. Thus this experiment tells us that remote access only has very limited impact on remote emulation delay and the validity of the ROSENET approach supporting remote emulation is thus justified in this sense.

4.2.4.4 Remote Emulation Delay vs. Distributed Simulation Scale

Since ROSENET targets using high performance computing facilities to perform large scale network simulation to provide more realistic network scenarios for network emulation, a group of experiments were performed using larger scale network simulation

to find out what impact the simulation scale could have on remote emulation delay. When more federates or processors are involved in the distributed simulation, a larger network topology and more traffic flows can be simulated. At the same time the execution time will become larger as shown by the scalability test results in [19]. In this experiment, the update interval is 1 second and maximum data stream for each connection is 500,000 bytes. Each leaf node sends On/Off application data at 100kb/s.

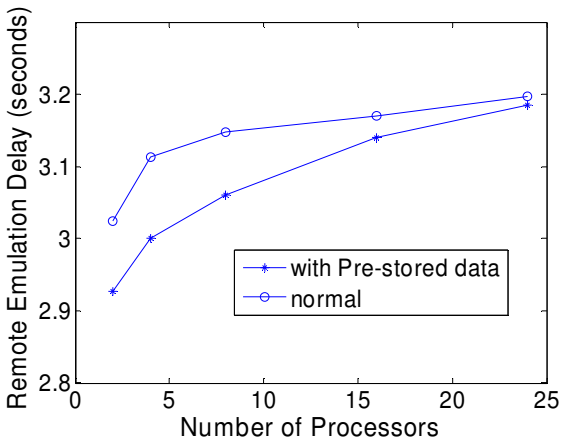

Figure 57 Remote Emulation Delay with Different Number of Processors

Figure 57 measures the remote emulation delay value versus the number of processors in the high fidelity simulation. With each processor simulating a DARPA NMS campus network with 574 nodes, the system is tested on 2, 4, 8, 16, and 24 processors. When more processors are involved and larger network topologies are simulated, remote emulation delay becomes larger. This increased cost is incurred due to the fact that the cost to advance time with more federates becomes larger. It also can be seen that remote emulation delay for pre-stored data is smaller than remote emulation delay under normal conditions. The difference in the remote emulation delay values between pre-stored and normal condition is between 0.05 and 0.1 seconds, which is very small. Due to limited

resources the experiments could not be extended to larger scales. The experimental results show that simulation scale has a very small impact on remote emulation delay, due to the fact that other costs overshadow the effects of simulation scale. This demonstrates ROSENET's capability to support large scale simulations, and future work could be performed measuring ROSENET's tolerance limits for simulation scale and simulation load.

4.3 Case Study Using Skype

When QoS of real time applications are studied, they should be tested with communication network characteristics in which the type of network and background traffic can be controlled. The goal of this case study is to illustrate the use of ROSENET to examine the impact of changing network behaviors on real time applications. For this case study, the Skype VoIP application is chosen, which is commercial overlay peer-to-peer network software whose implementation is proprietary. Experiments [82] have been performed on the Internet to determine how Skype works and only a few experiments have been completed to test Skype over controlled network environment. The ROSENET system provides a tool to test Skype under different network environments and scenarios.

4.3.1 Experimental Setup

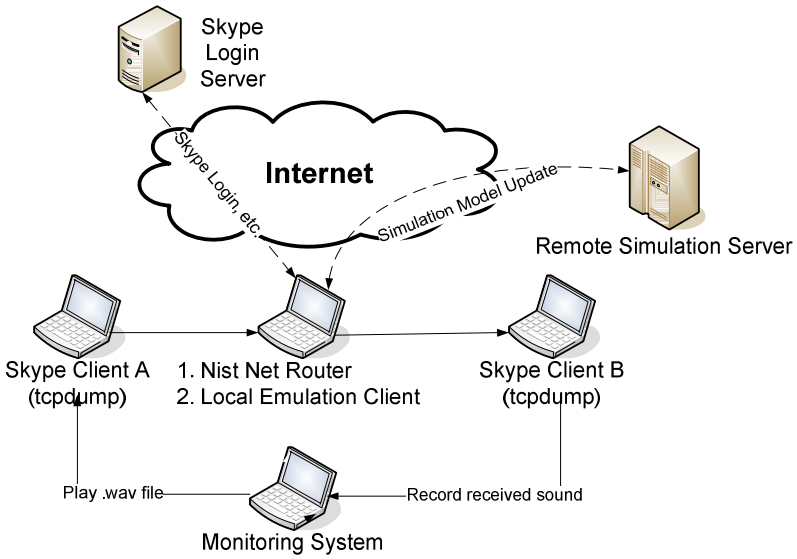

Figure 58: Skype Experiment setup

Figure 58 shows the Skype experimental setup. The NIST Net router is equipped with a second network card to connect to the Internet so that Skype clients on test machines A and B can login to the Skype server initially. After the initial login traffic is routed directly between machine A and B through the NIST Net router without going through a relay node on the Internet. The emulation client runs on the same machine as the NIST Net router and is connected with the remote simulation server which simulates the network environment used by Skype. A fourth computer is used as a monitoring system that generates voice to feed into the Skype Client A and records received voice on Skype Client B. Ethereal [74] is used to log the UDP traffic during the conversation on the test machines.

The remote simulation server simulates a dumbbell network topology in which the two Skype clients are modeled as a pair of source/destination nodes sharing a bottleneck link

with a number of background traffic nodes which generate On/Off application traffic. The On/Off applications have an exponential distribution with On/Off time duration at 2.5 seconds average interval. During the On period, the On/Off application sends constant bit rate streams at10Mb/s. The simulation results such as end-to-end delay and loss are collected on the run at the end of each simulation time interval and generalized as model updates to the NIST Net emulator.

4.3.2 Skype Traffic Modeling

Because live Skype traffic will be fed into the simulation server periodically, it is necessary to figure out how to model Skype traffic in GTNetS simulation. [82] has found that Skype can automatically adjust its packet rate and packet size (QoS) to adapt to changing network bandwidth and loss probabilities by switching to a different codec. Skype uses UDP unless TCP is required. In this experiment both machines use UDP during the conversation session and a very small amount of TCP packets are sent out to communicate with Skype super nodes on the Internet as control messages. The size of Skype voice packet is decided by the codec used. The typical bandwidth with Skype is 3-16 kbps.

No silence suppression is used in Skype, which means when neither caller nor callee is speaking, voice packets still flow between them. Therefore at any time during the simulation two flows are going through the NIST Net emulator at the same time. Since the VoIP application provided by GTNetS supports silence suppression, CBR UDP applications are used instead to simulate Skype traffic since Skype changes packet rate and size not very frequently in the experiments. If Skype changes codec, the packet size

and rate will change, which will be modeled by the *TrafficSummaryModel* using different CBR rate and packet size. The corresponding end-to-end delay changes will be modeled by *LowFidelityModel* and applied to the NIST Net emulation.

4.3.3 Multiple Flow Synchronization

Since no silence suppression is supported in Skype, the high fidelity simulation and low fidelity emulation are required to support two flows at the same time. The existence of multiple flows in the system causes a problem that is not met before when there is only one flow as in the experiments in Section 4.1. If the two traffic flows are injected in the simulation and emulation at different time, the simulation and emulation results will be invalid since caller and caller should be talking to each other at the same time. Hence the traffic flows should happen at the same time in both simulation and emulation.

For this reason, the two flows in both simulation and emulation are synchronized by applying the *LowFidelityModel* and *TrafficSummaryModel* describing traffic in the same time frame. Since the network models describe traffic for a time interval, the definition of synchronization is loosened to the same time interval instead of the same point in time. In high fidelity simulation, a minimum end time is calculated among all flows' *LowFidelityModel* in order to guarantee that all traffic applies to the simulation in the same time frame. Accordingly the *LowFidelityModel* generated after that round describes QoS status of the network for all the flows at the same time frame and can be used by the NIST Net emulator as long as they are applied to the emulator at the same time. NIST Net can apply network properties such as end-to-end delay and loss on different flows at the same time, but it does not need to address the problem of synchronization of all flows

since the network parameters in NIST Net are static.

On the other hand, when traffic is monitored and collected at the NIST Net router, each flow may have a different time frame for their model unless the traffic is collected at the interval defined in time instead of in the number of messages. It is also safe to assume that the length of time interval of *TrafficSummaryModel*s from different flows is different in practice. When the high fidelity simulator uses traffic models in the simulation, it must make sure the models describing the same time frame are used.

4.3.4 Experimental Results

The results collected in the experiments are end-to-end delay and loss. The traces collected show that Skype sending rate remains approximately 4kbps and the packet size is around 49 bytes. Since the Skype traffic rate is very low, the loss rate is very small. On the other hand, the end-to-end delay is largely affected by the On/Off background traffic sharing the bottleneck link with the Skype traffic so the focus is on the end-to-end delay in this section.

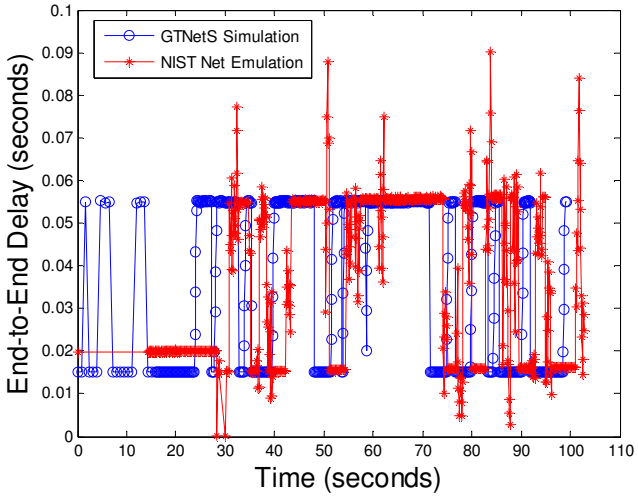

Figure 59: Caller to Callee End-to-End Delay

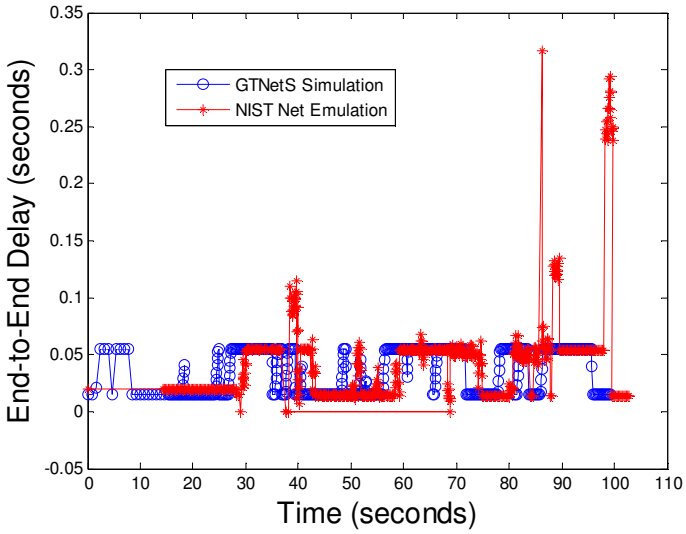

Figure 60: Callee to Caller End-to-End Delay

Figure 59 and Figure 60 show the end-to-end delay from caller to callee and from callee to caller. Initially the NIST Net end-to-end delay is set at 20 milliseconds when no *LowFidelityModel* is available to be used by the emulator. That is why the initial end-to-end delay for emulation is approximately 20 milliseconds in both figures. As the high

fidelity simulation takes some time to initialize, a number of *TrafficSummaryModel*'s are also generated by the client at the same time. After the high fidelity simulator starts to simulate, it needs to consume the *TrafficSummaryModel* that has been accumulated during the time it initializes the simulation. This is the reason that a large lag is shown in both figures between emulation and simulation and the lag is much large than one time interval.

In Figure 60 during the time period from 80 seconds to 100 seconds, a few very large values of end-to-end delay in emulation results are shown. These very large end-to-end delay values can be explained by the fact that with two flows going through the NIST Net kernel module, it requires more CPU power to monitor the traffic, generate the predictions, and copy the data from kernel to user space while the client in the user space communicates with the simulation server to update network models for both flows. Hence certain packets are delayed in their delivery due to competition for CPU.

4.4 Conclusion

This chapter evaluates ROSENET's performance and addresses the challenges in applying a remote network emulation approach to today's new services and applications through case studies of military application and commercial VoIP application Skype. Experiments examining the emulation accuracy in regard to end-to-end delay, loss, and sending rate show that ROSENET can accurately predict QoS parameters while meeting real time constraints for network emulation. The case study using a large scale simulation with DARPA's NMS network topology in ROSENET illustrates the use of the remote emulation delay metric to predict the ROSENET system's performance. Both the

analytical model and experimental results show that remote emulation delay has a predictable and also bounded value, which can be used to inform users of potential inaccuracy in the emulation while users take advantage of the benefits of remote network emulation. In the second case study Skype is used as a real world application to illustrate how ROSENET can be used to test real time applications. The experimental results show that this large scale network emulation framework capable of integrating a remote high fidelity simulation facility with local network emulation can meet requirements for scale, accuracy, timeliness and accessibility..

5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

Motivated by the requirements of testing new services and applications and the problems in existing emulation tools, the ROSENET approach utilizes a remote high fidelity simulation to update and calibrate a local low fidelity emulator, in order to achieve scale, accuracy, and timeliness in network testing. The ROSENET achieves timeliness through the low fidelity emulator which quickly provides QoS predictions to real time distributed applications, and scale and accuracy through remote high fidelity simulation running on remote high performance computing facilities.

Although research has been done using high performance computing facilities for real time network simulation or emulation, no previous work has addressed the accessibility of remote network simulation to users for emulation purposes. To reduce the communication cost and potential latency over a wide area network, periodically updated network models are used in the remote network emulation framework to synchronize the distributed simulators and emulators. The validity of using network models to describe traffic over a time interval is based on the observations that network traffic is constant (mathematically, operationally, or predictively) within a time interval. Due to the Internet's distributed topology and its support for dramatically heterogeneous mixtures of protocols, services and applications, it is very difficult to use a single static model to accurately represent a wide spectrum of network dynamics. A library-based network modeling techniques use different modeling approaches to describe various traffic

patterns by selecting models according to certain criteria such as user requirements for accuracy, communication and computational cost and dynamically update the models or model parameters during the emulation process.

Emulation results for end-to-end delay and loss from the ROSENET emulation system are compared with those from the high fidelity simulation, assuming that the results from high fidelity simulation represent the "true" QoS predictions. The two case studies using a large scale network topology with synthetic traffic workload and a real world VoIP application Skype test ROSENET in a real world environment. The results show that ROSENET is a promising approach that can provide scale, accuracy, and timeliness for testing new generations of services and applications.

5.2 Future Work

Future work is recommended in the following areas:

- **Remote Wireless Network Emulation**. Large scale mobile testbeds are being developed such as ORBITS. The parallel discrete event simulator GTNetS also provides wireless simulation capabilities. Many new services and applications utilize wireless network environment and they require wireless network emulation tools for testing and evaluation. The ROSENET approach can be extended to wireless network by integrating the client/server architecture with wireless simulators or testbeds to provide remote network emulation capabilities. The remote accessibility of ROSENET emulation is especially important in this sense as users are not limited in a laboratory environment and a lot of research questions are open to be explored.

- **Network Traffic Pattern Analysis.** Although network traffic varies a much in the Internet environment, classify traffic into patterns and match modeling techniques to specific traffic patterns will help improve network modeling and update efficiency.
- **Large Scale Testing.** An emerging trend in parallel and distributed simulation is to execute simulations over very large number of processors involving hundreds of thousands of processors by using super computers such as Blue Gene. Extending ROSENET's support for simulation on these supercomputers and test with machines involving thousands of processors is of great importance under the context of new generations of services and applications such as network centric warfare and petascale simulation for science and engineering.

REFERENCES

1. Larry, P., et al., *A blueprint for introducing disruptive technology into the Internet.* SIGCOMM Comput. Commun. Rev., 2003. **33**(1): p. 59-64.

2. Fujimoto, R.M., *Parallel and Distributed Simulation Systems*. 2000: Wiley Interscience.

3. Emulation. *Wikipedia*. [cited; Available from: http://en.wikipedia.org/wiki/Emulation.

4. Carson, M. and D. Santay, *NIST Net: a Linux-based network emulation tool.* SIGCOMM Comput. Commun. Rev., 2003. **33**(3): p. 111-126.

5. Allman, M., A. Caldwell, and S. Ostermann, *One: The Ohio Network Emulator*. 1997, Computer Science Department, Ohio University Athens, Ohio.

6. Rizzo, L., *Dummynet: a simple approach to the evaluation of network protocols.* ACM Computer Communication Review, 1997. **27**(1): p. 31-41.

7. Ingham, D.B. and G.D. Parrington, *Delayline: A Wide-Area Network Emulation Tool.* Computing Systems, 1994. **7**(3): p. 313-332.

8. Baumann, R. and U. Fiedler, *A tool for emulating real network dynamics on a PC*. 2005, ETH Zurich.

9. Yeom, I. and A.L.N. Reddy, *ENDE: An End-to-end Network Delay Emulator Tool for Multimedia Protocol Development.* Multimedia Tools Appl., 2001. **14**(3): p. 269-296.

10. Vahdat, A., et al., *Scalability and accuracy in a large-scale network emulator.* SIGOPS Oper. Syst. Rev., 2002. **36**(SI): p. 271-284.

11. White, B., et al., *An integrated experimental environment for distributed systems and networks.* SIGOPS Oper. Syst. Rev., 2002. **36**(SI): p. 255-270.

12. Fall, K., *Network Emulation in the Vint/NS Simulator*, in *Proceedings of the Fourth IEEE Symposium on Computers and Communications*. 1999, IEEE Computer Society.

13. Zheng, P. and L.M. Ni, *EMPOWER: A Cluster Architecture Supporting Network Emulation.* IEEE Trans. Parallel Distrib. Syst., 2004. **15**(7): p. 617-629.

14. Raychaudhuri, D., et al. *Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols*. in *Wireless Communications and Networking Conference*. 2005.

15. Herrscher, D., S. Maier, and K. Rothermel. *Distributed Emulation of Shared Media Networks*. in *International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS)*. 2003. Montréal, Quebec, Canada.

16. Maier, S., D. Herrscher, and K. Rothermel. *On Node Virtualization for Scalable Network Emulation.* in *2005 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS '05)*. 2005. Philadelphia, PA, USA.

17. Fujimoto, R.M., *Parallel Discrete Event Simulation.* Communications of the ACM, 1990. **33**(10): p. 30-53.

18. Riley, G., R.M. Fujimoto, and M. Ammar. *A Generic Framework for Parallelization of Network Simulations*. in *Proceedings of the Seventh International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 1999.

19. Riley, G.F. *The Georgia Tech Network Simulator*. in *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*. 2003. Karlsruhe, Germany: ACM Press.

20. Liu, J. and D.M. Nicol. *DaSSF 3.0 User's Manual*. 2001 [cited July 18, 2007]; Available from: http://www.cs.dartmouth.edu/research/DaSSF/Papers/dassf-manual.ps.

21. Cowie, J.H., D.M. Nicol, and A.T. Ogielski, *Modeling the Global Internet.* Computing in Science and Engineering, 1999. **1**(1): p. 42-50.

22. Zeng, X., R. Bagrodia, and M. Gerla, *GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks*, in *Proceedings of the 1998 Workshop on Parallel and Distributed Simulation*. 1998. p. 154-161.

23. QualNet. *QualNet User Manual* [cited July 18, 2007]; Available from: http://www.qualnet.com.

24. Liu, Y., B.K. Szymanski, and A. Saifee, *Genesis: a scalable distributed system for large-scale parallel network simulation.* Comput. Networks, 2006. **50**(12): p. 2028-2053.

25. Fujimoto, R.M., et al. *Large-Scale Network Simulation: How Big? How Fast?* in *Proceedings of the11th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'03)*. 2003. Orlando, FL: IEEE Computer Society.

26. Simmonds, R. and B. Unger, *Towards scalable network emulation.* Computer Communications, 2003. **26**(3): p. 264-277.

27. Cameron, K., S. Rob, and U. Brian, *Improving Scalability of Network Emulation through Parallelism and Abstraction*, in *Proceedings of the 38th annual Symposium on Simulation*. 2005, IEEE Computer Society.

28. Nicol, D.M. *DARPA Network Modeling and Simulation (NMS) baseline network topology.* 2003 [cited July 18, 2007]; Available from: http://www.ssfnet.org/Exchange/gallery/index.html.

29. Zhou, J., et al., *MAYA: Integrating hybrid network modeling to the physical world.* ACM Trans. Model. Comput. Simul., 2004. **14**(2): p. 149-169.

30. Liljenstam, M., et al., *RINSE: The Real-Time Immersive Network Simulation Environment for Network Security Exercises (Extended Version).* Simulation, 2006. **82**(1): p. 43-59.

31. Xia, H., et al., *The MicroGrid: Using Online Simulation to Predict Application Performance in Diverse Grid Network Environments*, in *Proceedings of the Second International Workshop on Challenges of Large Applications in Distributed Environments - Volume 00*. 2004, IEEE Computer Society.

32. Cowie, J., et al., *Towards Realistic Million-Node Internet Simulations*, in *International Conference on Parallel and Distributed Processing Techniques and Applications*. 1999.

33. Bagrodia, R., et al. *An accurate, scalable communication effects server for the FCS system of systems simulation environment*. in *Proceedings of the 38th conference on Winter simulation*. 2006. Monterey, California.

34. Xu, D., et al., *Split Protocol Stack Network Simulations Using the Dynamic Simulation Backplane*, in *Proceedings of the Ninth International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. 2001.

35. Riley, G., et al. *Distibuted Network Simulations using the Dynamic Simulation Backplane*. in *International Conference on Distributed Computer Systems*. 2001.

36. *Georgia Tech Dynamic Network Emulation Backplane*. 2002 [cited; Available from: www-static.cc.gatech.edu/computing/pads/netemulation/

37. Fujimoto, R.M., et al. *Design of High-performance RTI software*. in *Distributed Simulations and Real-time Applications*. 2000.

38. Xu, K., et al. *Looking ahead of real time in Hybrid component networks*. in *Proceedings of the fifteenth workshop on Parallel and distributed simulation*. 2001. Lake Arrowhead, California, United States: IEEE Computer Society.

39. Liu, J., et al. *An Open and Scalable Emulation Infrastructure for Large-Scale Real-Time Network Simulations*. in *Proceedings of the 26th IEEE International Conference on Computer Communications*. 2007.

40. Gu, Y. and R. Fujimoto. *A Flexible Architecture for Remote Server-Based Emulation*. in *Proceedings of the International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems*. 2004: IEEE Computer Society.

41. Perumalla, K.S. *Parallel and distributed simulation: traditional techniques and recent advances*. in *Winter Simulation Conference*. 2006.

42. Zhang, Y. and N. Duffield. *On the constancy of internet path properties*. in *the First ACM SIGCOMM Workshop on Internet Measurement*. 2001. San Francisco, California, USA: ACM Press New York, NY, USA.

43. Kuhl, F., R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Introduction to the High Level Architecture for Simulation*. 1999: Prentice Hall.

44. Black, J.W. and D.L. Harris. *RTI Recommended Practices*. in *Proceedings of the Fall Simulation Interoperability Workshop*. 1999. Orlando, FL.

45. Anirban, M., L.E. Derek, and K.V. Mary, *Improving multirate congestion control using a TCP Vegas throughput model.* Comput. Netw. ISDN Syst. **48**(2): p. 113-136.

46. Staniford, S., V. Paxson, and N. Weaver. *How to own the Internet in Your Spare Time*. in *the Proceedings of the 11th USENIX Security Symposium (Security '02).* 2002.

47. *The Network Simulator - ns2 homepage, http://www.isi.edu/nsnam/ns/.*

48. Frost, V.S. and B. Melamed, *Traffic modeling for telecommunications networks*, in *IEEE Communications Magazine*. 1994. p. 70-81.

49. David, L.J., M. Benjamin, and W. Walter, *Stochastic modeling of traffic processes*, in *Frontiers in queueing: models and applications in science and engineering*. 1997, CRC Press, Inc. p. 271-320.

50. Jain, R. and S. Routhier, *Packet Trains--Measurements and a New Model for Computer Network Traffic.* IEEE Journal of Selected Areas in Communications, 1986. **SAC-4**(6): p. 986-995.

51. Anick, S., D. Mitra, and M.M. Sondhi, *Stochastic Theory of a Data-Handling System with Multiple Sources.* Bell System Tech. J., 1982. **61**(8): p. 1871-1894.

52. Paxson, V. and S. Floyd, *Wide-Area Traffic: The Failure of Poisson Modeling.*

IEEE/ACM Transactions on Networking, 1995. **3**(3): p. 226-244.

53. Crovella, M.E. and A. Bestavros, *Self-similarity in World Wide Web traffic: evidence and possible causes.* IEEE/ACM Trans. Netw., 1997. **5**(6): p. 835-846.

54. Leland, W.E., et al., *On the self-similar nature of Ethernet traffic (extended version).* IEEE/ACM Trans. Netw., 1994. **2**(1): p. 1-15.

55. Willinger, W., et al., *Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level.* IEEE/ACM Transactions on Networking, 1996. **5**(1): p. 71-86.

56. Erramilli, A., et al., *Self-Similar Traffic and Network Dynamics.* Proceedings of the IEEE, 2002. **90**(5): p. 800-819.

57. Abry, P. and D. Veitch, *Wavelet analysis of long-range-dependent traffic.* EEE Transactions on Information Theory, 1998. **44**(1): p. 2-15.

58. Vishwanath, K.V. and A. Vahdat, *Realistic and responsive network traffic generation.* SIGCOMM Comput. Commun. Rev., 2006. **36**(4): p. 111-122.

59. Lan, K.-c. and J. Heidemann, *Rapid Model Parameterization from Traffic Measurements.* ACM Transactions on Modeling and Computer Simulation, 2002. **12**(3): p. 201-229.

60. Wong, J.W., *Queueing Network Modeling of Computer Communication Networks.* ACM Comput. Surv., 1978. **10**(3): p. 343-351.

61. Zhang, W. and J. He. *Statistical Modeling and Correlation Analysis of End-to-End Delay in Wide Area Networks*. in *the Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD).* . 2007.

62. Yang, M., et al., *Predicting Internet end-to-end delay: A statistical study.* Annual Review of Communications, 2005. **58**: p. 665-677.

63. Doddi, S., *Empirical modeling of end-to-end delay dynamics in best-effort networks*. 2005, Texas A&M University.

64. Riedi, R.H., et al., *A Multifractal Wavelet Model with Application to Network Traffic.* IEEE Transactions on Information Theory, 1999. **45**(4): p. 992-1018.

65. Huang, L. and K. Sezaki, *End-to-End Internet Delay Dynamics.* Proceedings of the 6th Asia-Pacific Conference on Communications (APCC), 2000.

66. Ohsakia, H., M. Muratab, and H. Miyaharaa, *Modeling end-to-end packet delay dynamics of the Internet using system identification,*. Proceedings of Seventeenth International Teletraffic Congress, 2001: p. 1027-1038.

67. Paxson, V., *End-to-End Internet Packet Dynamics.* IEEE/ACM Transactions on Networking, 1999.

68. Bolot, J.-C. *End-to-End Packet Delay and Loss Behavior in the Internet*. in *SIGCOMM*. 1993.

69. Tuan, T. and K. Park, *Multiple time scale congestion control for self-similar network traffic.* Performance Evaluation, 1999. **36**: p. 359-386.

70. Ljung, L., *System Identification: Theory for the User*. 1999: Prentice Hall.

71. Perumalla, K.S. and S. Sundaragopalan, *High-Fidelity Modeling of Computer Network Worms*. 2004, Georgia Institute of Technology.

72. Gu, Y. and R. Fujimoto. *Performance Evaluation of the ROSENET Network Emulation System*. in *Proceedings of The 11-th IEEE International Symposium on Distributed Simulation and Real Time Applications*. 2007: IEEE Computer Society.

73. Gu, Y. and R. Fujimoto, *Performance Evaluation of the ROSENET Network Emulation System*. 2007, submitted.

74. Ethereal. *Ethereal: a Network Protocol Analyzer*. [cited 2007 August 28]; Available from: http://www.ethereal.com/.

75. Grenada. *The United States at War*. [cited August 28, 2007]; Available from: http://www.geocities.com/Pentagon/Camp/7624/Grenada.htm.

76. Talbot, D., *How Technology Failed in Iraq.* Technology Review, 2004.

77. Gu, Y. and R. Fujimoto, *Remote Network Emulation for Defense Applications*. 2007, submitted.

78. Gu, Y. and R. Fujimoto. *Applying Parallel and Distributed Simulation to Remote Network Emulation*. in *Proceedings of the 2007 Winter Simulation Conference*. 2007.

79. Mevassvik, O.M., K. Bråthen, and R.M. Gustavsen. *JADE – An Experiment in Distributed Simulation Based Joint Tactical Training*. in *Transforming Training and Experimentation through Modelling and Simulation*. 2006.

80. Cole, R.G., et al. *On a Global Information Grid Simulation Platform for Investigations of End-to-End Performance*. in *IEEE Military Communications Conference (MILCOM)*. 2005.

81. Floyd, S. *Building Models for Aggregate Traffic on Congested Links*. 2002 [cited July 18, 2007]; Available from: http://www.icir.org/models/linkmodel.html.

82. Baset, S.A. and H. Schulzrinne, *An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol*, in *IEEE Infocom*. 2006.