Computer Science Master’s Project

# Converting SSFNet Simulation Definition to Genesis Format

By

**Jonathan Shea**

**Rensselaer Polytechnic Institute**

**Troy, NY 12180**

## Abstract

Effective large-scale network simulation is a difficult task to accomplish efficiently due to complex and dynamic routing tables, route stability, along with many other complexities. Due to the collective power of the Internet routers used in traffic routing, the Internet comprises the world's largest distributed computer system, which in turn makes network simulation increasingly difficult. Despite these complexities, network simulation is needed as a research tool to further understand large networks (such as the Internet), routing capabilities, and new technologies. Network simulation is not only necessary to further our understanding of large and small networks, but can potentially be applied to other areas of science, including biotechnology, electrical engineering, and virtually every other facet of science and high-technology fields. The aim of this paper is to outline current network simulation tools and techniques, and to describe a recently developed tool used to convert domain modeling language files between two simulators, SSFNet and Genesis.

## 1. Introduction

The Internet has been growing exponential rates of the past several years, which has led to many new protocols and algorithms to handle the increasing demands on networks. New protocols need to be tested and evaluated to find security, scalability, and quality-of-service issues among other critical components of network design. In order to test these protocols, simulation tools need to be in place with behavior mimicking the environment they will be used in, whether that is a small local area network, like a home network where only a few nodes contend for network bandwidth, or larger scale networks, such as the Internet, where millions of nodes are all interconnected. By effectively simulating protocols and the associated network behavior, developers and researchers are able to further enhance, tune, and resolve issues with not only new, but also existing, network protocols. Additionally, network protocols must be tested to ensure they are robust, secure and reliable. These tests must be completed under varying an unpredictable (sometimes) conditions. The goal of this survey is to outline and evaluate network simulation techniques, and then to focus on two prominent network simulators, SSFNet [10, Web5] and Genesis [21, 27], and introduce a converter which is used to convert configuration files between these network simulators. It should be noted that network simulation is just one application of discrete simulation technology that has been applied to other areas of science, including computer systems [1, 14], epidemiology [6, 7], medicine [12, 17], emergency services [9], origins of life [16] and virtually every other facet of science and high-technology fields.

Network simulators need to be able to perform several tasks efficiently to effectively simulate a network. First of all, the network simulator must accurately simulate a network of varying sizes. Effective network simulators must be able to simulate not only small and simple networks, such as a home LAN with only a few nodes, but also very large scale networks, such as corporate environments and even the Internet as a whole. To accurately simulate a network of varying size, a network simulator must be able to transparently simulate nodes, routers, data links, and other OSI model layer 2, 3, and 4 objects. Additionally, a network simulator must be able to efficiently route traffic between these objects [5, 25].

Another requirement of network simulators is operate within reasonable physical constraints. Memory, computational power and disk space are all finite and restrictive constraints. Despite the fact computers today are exceptionally powerful, an effective network simulator should seek to minimize the power necessary to simulate large scale networks. Different approaches have been taken to deal with this issue included distributed computing over several computers, stand-alone environments, and peer-to-peer simulations.

Unfortunately, designing effective and efficient network simulators is a difficult task to accomplish. As simulated networks grow in size, the necessary computational power and resources grows. Different simulation technique designs handle these issues in different ways, as will be explained later, however, an optimal design has not yet been achieved. As mentioned above, there are many techniques in use to simulate networks. While not one has been deemed the “best”, there are many advantages to using networked computers as well as a standalone design. These options will be examined later.

The remainder of this report is organized as follows. Section 2 outlines the history of present simulators. Section 3 presents and describes several current network simulation tools. Section 4 outlines the converter used to convert network configuration files. Section 5 is an evaluation. Section 6 is the conclusion and future work.

## 2. Brief History of Network Simulation

Mainstream network simulation began as the Internet (then ARPANET) began growing quickly and the need for an evaluative and descriptive tool to examine networks was needed. Tools were needed to solve issues that were arising from the rapidly increasing network size. This section will outline early network simulators which are now the basis for many current simulators.

### 2.1 NEST – NEtwork Simulation Testbed

One of the first network simulators, in fact the basis for many of today's network simulators, was the NEST simulation testbed. NEST was developed in the late 1980s jointly between researchers at the David F. Bacon IBM T.J. Watson Research Center and Columbia University, both in New York State [11]. Designed to simulate networked environments, NEST was used to develop and analyze distributed systems and algorithms. At its time, NEST came with an advanced graphical user interface and dynamic configurability. NEST was designed to run under UNIX, in a single process. The simulator was designed with a focus on packet ordering and process timing. NEST gave researchers a tool to simulate real network applications, which could then be run on real networks with minor code changes.

The NEST user interface was designed to run as a separate entity as the simulator, giving researchers the ability to run the simulation on a compute server, whiles the user interface could be run on the user's workstation. One of the main features of the user interface was to dynamically change the network being simulated. Nodes could be added and removed, links changed, and the functions defining link and node behavior could be changed.

NEST's main objectives are to ensure simulation time is kept, which in turn ensures on-time and in order packet delivery, and context switching between nodes. Context switches are managed by allocating each node a block of time, which the node can either sleep, wait for massages, or run. Once the time slice is used, the simulators copies registers, stacks, and errno to a "save area", loads the next scheduled node, and restores the saved information.

As mentioned above, links between nodes can be defined to run using different functions. Transmission behavior can be defined however the user wishes. The default function is *reliable*, which operates the same way TCP would run. However, transmission links can be programmed to operate as congested links or long transfer delays.

NEST was used to test ARPANET, the predecessor of the Internet. ARPANET was experiencing a problem having to do with store-and-forward switches and dynamic packet routing and topology updates. Researchers used NEST to build a small version of ARPANET, and then sent data messages and topology updates. NESTs ability to crash and restart nodes was extremely useful in solving the issues in this project.

## 2.2 REAL – REalistic And Large

The next step in network simulation was with a simulator designed by researchers at Xerox PARC in California, University of California at Berkeley, and Cornell University in New York called REAL (REalistic And Large) [13, Web4]. REAL was built using NEST as the underlying basis for design. REAL was initially designed to compare the fair queuing algorithm and the FCFS (first-come-first-served) network scheduling algorithms. However, researchers soon realized the extent of REAL stretched much further and simulating scheduling algorithms, and with simple modifications, REAL's scope was greatly extended.

Much like NEST, REAL was based on a server, which handled the simulations, and a client, which handled the user interface of the server. These were run independently and even on separate machines using UNIX sockets to connect together. The user interface was also enabled to monitor several simulations on difference compute servers, allowing a single interface for several tests.

REAL simulates a packet-switched store-and-forward network, common to wide-area-networks at the time. The network layer used datagrams which can be lost or delivered out of sequence and are transmitted unreliably, similarly to the corresponding layer of the OSI model of today. The transport layer acted similarly to TCP, which provided reliable communication using flow control, timeouts, acknowledgements, and retransmissions. Simulated networks contained *sources* to generate traffic, *gateways* that routed traffic, and *sinks* that absorbed the traffic and sending acknowledgement for received packets. There are over 30 sources in REAL, ranging from FTP and telnet, to ill-behaved sources which send the largest packets allowed by the links.

In addition to multiple sources, REAL also simulates many gateway scheduling algorithms. The most common algorithms are first come, first served (FCFS), fair queuing (FQ) and congestion avoidance, developed by DEC. Each of these scheduling algorithms has its own operational differences, which will be outlined below.

- FCFS – packets are sent in the order they are received. A FIFO (first in, first out) queue is created as a buffer to manage the incoming packets, which will send the next available packet when the output line is available for sending. If the buffer is too full to handle more packets, any overflow packets are dropped.
- Congestion avoidance – Developers at DEC (Jacobson and Karels) have developed a way to avoid congestion in FCFS queuing algorithms by insert a bit, called a decbit, into the header of the packet. When the algorithm detects congestion (when the average queue length is greater than 2), the decbit is set. When the receiver detects a decbit set, it decreases its window size until the system has stabilized. This system implicitly introduces fairness.
- FQ – Fair queuing implements the fair queuing scheduling policy described as a bit-by-bit round-robin scheme. In this algorithm, the gateway selects outgoing packets from each source bit-by-bit, taking turns between each source with traffic to send. This tactic ensures the fairest scheduling, as each source gets exactly the same bandwidth as every other source. The main disadvantage to fair queuing is the significant overhead associated with switching between sources with outgoing traffic.

REAL also introduced a start to a distributed simulation called DisREAL [13]. DisREAL bound several instances of REAL simulators together and divided the simulation into slices which were handled by different computers. DisREAL used the monitoring system in NEST as the basis for its design. It operated in passes, where each pass is an interval of simulation time.

REAL was used to simulate networks with up to 100 nodes. The theory behind the distributed system led the researchers to believe networks of thousands of nodes would be easily simulated.

## 3. Current network simulators

Network simulators have evolved significantly over the past several years since the introduction and evolution of REAL and NEST. There are now many different types of network simulators, ranging from traditional WAN and LAN applications, to wireless LANs, mobile communications (cellular telephones), neural networks, along with many different types. This section intends to introduce four current simulators, ns2, Parsec, SSFNet and GENESIS.

### 3.1 ns and ns2 – The Network Simulator (version 2)

ns is powered by a discrete event simulator [5], as opposed to a process simulator as its predecessors. It is designed to simulate and support TCP using routing and multicast protocols. ns effectively simulates networks that are both wired and wireless (local and satellite) networks. ns is also used to simulate networks used for multimedia transport. ns is still an on-going research project with support from many places, including Defense Advanced Research Projects Agency (DARPA), the National Science Foundation (NSF), Xerox Palo Alto Research Center (PARC), and the Lawrence Berkeley National Lab (LBL), as well as researchers at many institutions, including University of California at Berkeley, Carnegie Mellon University, and the University of Southern California.

ns began in 1989 with the start of development on the REAL network simulator. ns used REAL as a base for its network simulation, much like REAL used NEST as a basis. In 1995, however, DARPA started supporting ns development through the VINT (Virtual InterNetwork Testbed). Since 1995, ns has evolved significantly and now remains a significant force in the research and development field. Many projects use ns as a base for network simulations.

ns currently supports a host of protocols including almost every variant of TCP, several forms of multicast, wired and wireless networking, several ad hoc routing protocols, satellite communications, as well as other common and obscure protocols. ns also includes the ability for others to contribute and maintain their own protocols and services. Currently there are over 50 contributions on ns's website, ranging from IS-IS routing algorithms to Bluetooth wireless simulators and graphing programs.

ns is built using a combination of Tcl and C++. Users interact with ns using Tcl scripts which define the network topology, protocols in use, traffic patterns, sources and sinks. Inside a configuration file, users define *nodes*, which may contain *agents*, which take care of sending and receiving packets. *Links* are used to connect nodes and have many user specified properties, including bandwidth, packet drop rates, and congestion.

Additionally, users can specify time-specific network changes, such as a link going down at 100 seconds of simulation. A set of experiments are developed using Tcl, which are then input into the ns event simulator.

The event simulator operates by maintaining a queue of events, which are scheduled by the nodes in the Tcl script. Events are packet IDs, unique to a specific packet, which have a scheduled time, and a pointer to the handle for the event. When the scheduled time arrives for a packet, ns delivers the packet to the event handler responsible for handling the event. Event simulation takes zero simulation time, so a delay would need to be implemented to simulate a true network. Delays are simulated by reissuing an event to the event scheduler, with the delay as the scheduled time. The event is retriggered when the delay is up.

Once a simulation is complete, ns outputs results into detailed txt files containing all the simulation data. Users can analyze this data using result analysis tools such as trace utilities and queue analyzers, or a graphical network simulation tool called Network Animator (NAM) developed as part of the VINT project [5]. The graphical simulator is meant only to be used as such; the results are not valid for an accurate simulation analysis.

Due to the sequential operating construction of ns, large-scale simulations are not very efficient. Researchers have developed the Telecommunication Description Language (TeD) [18], which automates network parallel network simulation in ns. TeD operates by using *entities* which communicate using *events* and *channels*. TeD maps each fundamental piece of ns (nodes, links, and agents) to specific TeD entities.

ns is currently used in many network simulation environments. The expandability and extensibility of ns has made the system a very popular network simulation system. ns also remains as the basis for many more current network simulators and other systems that need a robust and proven event scheduler.

## 3.2 Parsec – Parallel Simulation Environment for Complex Systems

Parsec was developed by researchers at UCLA as a solution for simulating large scale networks where sequential simulations would be infeasible [3]. Parsec maintains a parallel design structure which allows for many more calculations per second, which in turn allows simulations to be run in exponentially less time. Parsec was designed to create a feasible parallel network simulation environment to run on distributed- and shared-memory systems.

The Parsec environment consists of a programming language called Parsec (based on the Maisie simulation language [4]), a GUI called Pave (Parsec Visual Environment), and the simulation environment. The simulator operates similarly to ns in that they are both *event driven* simulators. Parsec, however, due to its distributed nature, maintains the event queue in a much different manner. In ns, the event queue is centrally managed, as only one processor needs access to the queue. Parsec distributes the events to each processor, and each processor maintains its own queue of events to simulate. Coordination between processors must take place, however, to ensure the events are executed in proper order. To overcome this problem, parallel processor maintain three variables, EOT, or the earliest output time the processor is allowed to timestamp on a process, EIT, or the earliest input time the processor is allow to accept from other processors, and *lookahead*, which is the time period for which the processor can accurately predict all the events it will generate. These variables are all dynamic in simulation time. Parsec uses a process interaction approach to event simulation. *Entities* are linear processors (LPs) which correspond to physical processes. Events are simulated as messages between the LPs.

Pave is the GUI for Parsec. It can be used to design simulation models for the Parsec framework. Developers can still program by hand if they prefer. Pave was designed specifically for Parsec to handle parallel simulations. Developers can program networks using notation similar to flowcharting to design networks.

The runtime environment for Parsec is very portable, allowing it to be run under several environments, including parallel and sequential systems, and many flavors of operating systems. Four main algorithms are used by the runtime environment to enable synchronization between processors: sequential, three different types of conservative algorithms, an optimistic algorithm, and the ideal simulation protocol (ISP). The sequential algorithm is designed to run on single processor machines. Conservative algorithms do not tolerate causality issues – every message must be received in the right order. Messages with timestamps earlier than the processor's EIT are the only ones that will be processed. The three algorithms are:

1. Null-based algorithm which broadcasts a null message to all processors when one processors EOT changes. Other processors then update their own EOT's when this occurs.
2. Conditional event algorithm which alternated between the EIT calculation phase and the event processing phase.

3. Accelerated null message, which is a combination of the first two methods.

Optimistic algorithms will process messages that are received with timestamps greater then the processor's EIT, however, the processor must detect causality issues. The optimistic algorithm the Parsec runtime environment uses is based on lookahead defined by Chandy and Sherman [8]. ISP is an algorithm which predicts the lower bound of simulation execution time using a critical path concept [2].

Parsec has been used in many simulations and has been used as to develop GloMoSim which is a simulator for wireless battlefield and search and rescue operations [28]. Parsec has extended GloMoSim to be able to handle tens of thousand of wireless nodes. Parsec has been tested in many other environments including ATM and VLSI circuit models, and parallel architecture models.

### 3.3 SSFNet – Scalable Simulation Framework + Network Models

SSFNet (Scalable Simulation Framework Network Models) is another popular network simulator. SSFNet provides components for modeling and simulation of Internet protocols and networks at and above the IP packet level of detail [Web5]. SSFNet runs on a single machine, which can limit the size of networks simulated, and the amount of memory required to simulate can be very large, however, Dartmouth College has developed a version of SSFNet called DaSSF [Web1]. DaSSF is designed using a process-oriented, synchronized parallel simulator. DaSSF supports parallel simulation by using shared memory between all processors. DaSSF runs on various platforms, and uses messages to communicate and synchronize between processors. When compared to the performance (speed and memory consumption) of ns and other single-machine simulators, DaSSFNet achieves much better results.

SSFNet is written in Java with scalability issues on the forefront of design aspects. Specifically, efforts have been made to enhance the modeling scalability, or the number of nodes, traffic patterns, bandwidth, etc. the simulator can handle, and performance scalability, or the number of processors the simulator can run on. Current distributions of SSFNet include two derivatives or SSF, SSF.OS, used to model operating systems, and SSF.Net, which is used to simulate networks. Additionally, libraries for common internet protocols are also included.

As mentioned above, DaSSFNet is a distributed version of SSFNet. DaSSFNet is fully compatible with SSFNet, which means all packages developed for SSFNet can be used with DaSSFNet. DaSSFNet was originally designed as a C++ port of SSFNet, however, it evolved into the distributed system it is today through a reorganization of the SSF simulator which allowed for both shared memory processors and distributed memory clusters.

SSFNet (hence, DaSSFNet) networks are described using a DML (Domain Modeling Language). DML allows users to configure very large multi-protocol networks using a hierarchical method. A DML consists of white space delimited key-value pairs which describe the network using defined keys. Keys are defined by SSFNet and classes imported in to the existing framework. Classes can be developed by end users, however a multitude of classes exist ranging from simulators of network worms to internet protocols and servers.

SSFNet does allow for multiple processor machines to simulate a network. In fact, using multiple processors increases performance significantly. SSFNet organizes the processors by using multiple queues to order events. SSF uses multiple threads running in parallel to ensure the queues are executed in order. SSFNet hides all these details from the modeler. Modelers tell SSFNet how to partition the network between processors by using the *alignment* attribute in the DML file.

## 3.4 Genesis – GEneral Network Simulation Integration System

Genesis uses a distributed memory and simulator technique to simulate networks [22]. Genesis can connect ns2 [21], SSFNet [20] and GloMoSim [16] into one large simulation that requires infrequent synchronization but approximate traffic flows [23] and novel traffic generators [26]. It uses a general time-space simulation technique [19] proposed by Chandy-Sherman in [8]. Its main application is to network management [29].

Genesis physically divides the DML into partitions, called simulation domains (domains for short), with each DML defining the slice of the network the individual machine will simulate independently and simultaneously as the others. Many computers each simulate their own portion the network, which eliminates the bounding on a single computer.

Such a design, however, needs a way to communicate between network slices simulated on different machines. Genesis provides a novel approach to this problem by introducing a proxy domain [20]. Each network segment contains and *external link* to the proxy domain.  The proxy domain handles switching all inter-domain packets.  This takes zero simulation time.

Such an approach allows individual machines to simulate much smaller network, however, altogether, the entire system is simulated through the collective efforts of each machine.  Each machine uses a simulator based on ns to simulate network to simulate its assigned domain.  Genesis adds several features to ns to allow for such a simulation to work.  First, Genesis enables ns to suspend internal simulation of networks to allow packets leaving over proxy interfaces to leave the system.  This allows the system to seamlessly integrate the networks together.  Secondly, Genesis records information about delays and drop rates.  This allows inter-domain links to delay inter-domain messages.  Finally, Genesis allows for *fake sources,* or proxy nodes which enable inter-domain communication.  These nodes are enabled to send traffic to nodes external to the current network.

## 4.  DML Converter

Domain Modeling Language (DML) is used to model networks in SSFNet and Genesis.  Both network simulators use the same prototypes for DML, which allows the code to be used almost interchangeably.  DML design specifications are almost identical for the two systems, however, there are differences in the layout of the DMLs.  For instance, GENESIS needs to have each network slice in a separate file, and a proxy domain must exist in each network configuration file to allow inter-processor exchanges.  Links are created between the proxy domain and the network slices.

## 4.1 Domain Modeling Language (DML)

Due to the distributed nature of Genesis, a traditional DML file will not work. Each computer which will be simulating a system needs its own version of the DML (Domain Modeling Language) file, with only the portions of the network that computer will be simulating. Additionally, every DML must also have a proxy domain, which is used to create links between every network in the system. The proxy domain acts as an intermediary whenever traffic travels between networks.

## 4.2 Converter

Researchers will have to convert DML files from the format SSFNet handles to the divided format Genesis expects. The converter also needs to dynamically create the proxy domain and the links connecting the domains through the proxy domain. The remainder of this paper will outline the theory behind the converter, including any assumptions the converter makes, as well as the current status of the converter.

The purpose of the converter is to extract a single network definition, along with the supporting information from every other network from a larger DML file. A Genesis DML needs the following information:

- The entire network definition of the specified network
- Interface definitions of links in other networks connected to the specified network

- Hosts in other networks who communicate with the specified network
- BGP graph neighbors for every neighboring AS (autonomous system)
- Traffic patterns for every inter-domain link in the network
- Proxy domain with links to every inter-domain interface

All of this information is available in the aggregate DML. The converter extracts these pieces of information, and builds a Genesis DML for every network in the system.

Note: The converter does have a few assumptions about the ordering of the traditional DML. These assumptions are outlined and diagrammed in Appendix A.

The first step the converter does is to copy header information and the first network to the Genesis DML. Header information consists of the primary network definition and network specifications, such as frequency and random stream generators. Once the header is copied, the converter extracts the entire specified network definition. As shown in appendix A, the converter expects everything above the first network definition to be the header.

Once the header and specified network are copied, the converter parses the entire file to extract information about every host defined by the network. The converter searches for the traffic flow definition in the original DML, and for every flow pattern, notes the clients and servers that have flows to or from the specified network. These values are used to determine which hosts from each network need to be referenced in the Genesis DML, and which traffic patterns need to be defined.

The next step the converter makes is to determine which links need to be copied. The original DML file contains link information for inter-network links. An external awk script is used to extract this information.

Once the link information is extracted, the converter locates each of the networks with inter-network interfaces and copies the appropriate interface and router definitions. Once the interface is copied, the converter locates neighbor AS information, and if the network is a neighbor to the specified system, copies the information. The next step the converter takes is to copy hosts and servers from the network, if any exist.

Once network definitions for every network relevant to the specified network have been extracted from the original DML, the converter creates the proxy AS, and the links to every inter-network interface.

The final step the converter takes in creating the Genesis DML is creating the traffic patterns. The link information extracted previously is used in this case again to create traffic patterns between each client and server in the network.

## 5. Evaluation

The converter currently works for files designed in the same manner as Appendix A dictates. For instance, if traffic patterns were defined above the first network definition, the converter would treat this as header information, and it would appear twice in the file, once as a header, and once as converted traffic pattern information.

At this time, the converter relies on manual input to determine the proxy links, as this is entirely based on network design, and does not appear to be extractable from the original DML.  For this same reason, the converter cannot calculate gfids for these links. This problem is currently being researched, however; as of now, the converter relies on this information to be manually input.

The converter also will not handle traffic patterns that summarize servers and clients.  For instance, the nhi_range function will not be converted to individual servers, nor will clients be distributed to these servers.

## 6. Conclusions

As networks grow increasingly complex, the network simulators used to develop and test both new and existing protocols will need to become more robust.  This survey has tried to illustrate the importance of distributed systems with regard to simulating large scale networks efficiently.  Each of the simulators introduced has a slightly different approach to parallel simulation, from complex message passing algorithms in Parsec, alignment definition in SSFNet, and network partition simulations in Genesis.  There are benefits associated with each of these design decisions, however, each also has limitations.  For example, Genesis minimizes inter-processor traffic by independently simulating a network partition on its own processor.  Only inter-partition traffic will be transferred between processors.  However, the main drawback is the increase in the network configuration files.  Each partition requires proxy nodes and domains.

The converter developed to convert DMLs between SSFNet and Genesis is an indispensable tool for researchers who wish to switch between SSFNet and Genesis.  Separating DML files between the two systems is a very laborious task, and near impossible for networks of extreme size.

## 6.1 Future Work

As mentioned in Section 5, there still is work that needs to be completed on the converter to make it a fully functional tool. At this time, the converter can handle structured DMLs with straightforward configurations. The converter needs to be enhanced to handle more generic DML structures as well as a few common, but complex functions.

## Appendix A:

```
Net [
        <header>
        net [id #
                router [
                        id #
                        interface [
                                id #
                                …  ( interface definition )
                        ]
                        …  ( more interfaces )
                ]
                … ( more router definitions )
                graph [
                        ProtocolSession [
                                name bgp
                                …  ( protocol definition )
                                neighbor [
                                        as # …  ( AS definition )
                                        …
                                ]
                                …  ( more neighbor definitions )
                        ]
                        …  ( more protocol definitions )
                ]
                host [
                        id #
                        … ( host definition )
                        interface [
                                id #
                                … (interface definition )
                        ]
                ]
                ... ( more host definitions )
                link [ attach # attach # delay # ]
                …       ( more link definitions )
        ]
        … ( more network definitions )

        link [ attach # attach # delay # ]
        …  ( more inter-domain links )

        traffic [
                pattern [
                        client #
                        servers [
                                nhi # port #
                        ]
                ]
                … ( more patterns )
        ]
]
```

## Appendix B:

Following is an example of a DML file conversion. The network involved in this example is a network with 4 areas, each with two hosts, a server and a host. Each host accesses a server in a different network. The network uses routers configured to use BGP to handle traffic between the areas.

The files involved are listed below:

ssf.dml.txt – SSFNet version of the DML.

net1.dml.txt – Network 1 extracted from SSFNet DML

net2.dml.txt – Network 2 extracted from SSFNet DML

net3.dml.txt – Network 3 extracted from SSFNet DML

net4.dml.txt – Network 4 extracted from SSFNet DML

## References:

[1] S. Azzaro and B.K. Szymanski, ``Simulating Dedicated UNIX PC-Based Application Systems," *Proc. 1990 Winter Simulation Conference*, O. Balci, R.P. Sadowski, R.E. Nance (edts), New Orleans, LA, December 1990, pp. 831-838.

[2] R. Bagrodia, V. Jha, M. Takai, *Performance Evaluation of Conservative Algorithms in Parallel Simulation Languages*, Technical Report CSD 980026, Computer Services Department, UCLA, 1998.

[3] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, H. Y. Song, "Parsec: A Parallel Simulation Environment for Complex Systems", IEEE Press 1998

[4] R. Bagrodia, W. Liao, "Maisie: A Language for the Design of Efficient Discrete-event Simulations," *IEEE Transactions on Software Engineering*," Volume 20, Number 4, pp. 225-238, 1994.

[5] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McAnne, K. Varadhan, Y. Xu, H. Yu, "Advances in Network Simulation", IEEE Press May 2000

[6] T. Caraco, M.C. Duryea, G. Gardner, W. Maniatty, and B.K. Szymanski, ``Host Spatial Heterogeneity and Extinction of an SIS Epidemics," *Journal of Theoretical Biology,* **192**, pp. 351-361, 1998.

[7] T. Caraco, W. Maniatty, and B.K. Szymanski, ``Population Dispersion and Equilibrium Infection Frequency in a Spatial Epidemic," *PhysicaD*, **132**, pp. 511-519, 1999.

[8] K. M. Chandy and R. Sherman, "Space-Time and Simulation", Proc. Distributed Simulation Conference, Society for Computer Simulation, San Diego, CA, 1989, pp 35-57

[9] G. Chen, B.K. Szymanski, and L. Wilson, ``Component-Based Simulation and Agent-Based Brokering: Towards Ad Hoc Simulations in Crisis and Emergency Management," *Proc. Computer Networks and Distributed Systems Modeling and Simulation, CNDS'03*, Orlando, FL, January 2003, pp. 37-44.

[10] James Cowie, David M. Nicol and Andy T. Ogielski, "Modeling the Global Internet," *Computing in Science & Engineering*, Vol. 1, No. 1, January/February 1999, pp. 42-50.

[11] A. Dupuy, J. Schwartz, Y. Yemini, "NEST: A Network Simulation and Prototyping Tool", Comm. ACM, October 1990, pp. 64-74.

[12] S. Glavankov, D. White, T. Caraco, A. Lapenis, G. Robinson, W. Maniatty, and B.K. Szymanski, ``Lyme Disease in New York State: Spatial Pattern at a Regional Scale," *American Journal of Tropical Medicine and Hygiene,* **65,** No. 5, May 2001, pp. 538-555.

[13] S. Keshav, "REAL: A Network Simulator", technical report 88/472, Univ. California Berkeley 1988.

[14] H. Lamehamedi, Z. Shentu, B.K. Szymanski, and E. Deelman, ``Simulation of Dynamic Data Replication Strategies in Data Grids," *Proc. 12th Heterogeneous Computing Workshop (HCW2003)*, Nice, France, April 2003, IEEE Computer Science Press, Los Alamitos, CA, 2003.

[15] N. Lehman, T. Caraco, W. Maniatty, and B.K. Szymanski, ``Spatial Models of Persistence in RNA Worlds: Exploring the Origins of Life," *Parallel Processing and Applied Mathematics,* LNCS Vol. 2328, Springer Verlag, Berlin, June 2002, pp. 896-903.

[16] K. Mandani and B.K. Szymanski, ``Integrating Distributed Wireless Simulation Into Genesis Framework," *Summer Computer Simulation Conference*, Montreal, Canada, July 2003, pp. 203-209.

[17] W. Maniatty, B.K. Szymanski, and T. Caraco, ``High-Performance Simulation of Evolutionary Aspects of Epidemics," *Applied Parallel Computing,* B. Kagstrom et al (eds), Papers presented at 4th Int. Workshop, PARA'98, June 16, 1998, Umea, Sweden, Lecture Notes in Computer Science, Vol. 1541, Springer-Verlag, Berlin, 1998, pp. 322-331.

[18] B. Premore, D. Nicol, "Parallel Simulation of TCP/IP using TeD", 1997 Winter Simulation Conference, Dartmouth College.

[19] B.K. Szymanski, Q. Gu, and Y. Liu, ``Time-Network Partitioning for Large-Scale Parallel Network Simulation under SSFNet," *Proc. Applied Telecommunication Symposium,* San Diego, CA, April 14-17, 2002, SCS Press, pp. 90-95.

[20] B.K. Szymanski, Y. Liu, and R. Gupta ``Parallel Network Simulation under Distributed Genesis," *Proc. 17th Workshop on Parallel and Distributed Simulation,* San Diego, CA, June 2003, to appear.

[21] B.K. Szymanski, Y.Liu, A. Sastry, and K. Madnani, ``Real-Time On-Line Network Simulation," *Proc. 5th IEEE Int. Workshop on Distributed Simulation and Real-Time Applications DS-RT 2001*, IEEE Computer Society Press, Los Alamitos, CA, 2001, Cincinnati, OH, August 13-15, 2001, pp. 22-29.

[22] B.K. Szymanski, A. Saifee, A. Sastry, Y. Liu and K. Madnani, ``Genesis: A System for Large-scale Parallel Network Simulation," *Proc. 16th Workshop on Parallel and Distributed Simulation*, Washington, DC, May 12-15, 2002, IEEE CS Press, pp. 89-96.

[23] B. Szymanski and Y. Liu, ``Loosely-Coordinated, Distributed, Packet-Level Simulation of Large-Scale Networks," *Proc. Winter Simulation Conference, WSC03*, New Orleans, LA, December 2003, to appear.

[25] G. R. Yaun, H. L. Bhutada, C. D. Carothers, M. Yuksel, S. Kalyanaraman, "Large Scale Network Simulation Techniques – Examples of OSPF and TCP Models"

[26] M. Yuksel, B. Sikdar, B.K. Szymanski, and K.S. Vastola, ``Workload generation for ns simulations of wide area networks and the Internet," *Proc. Communication Networks and Distributed Systems Modeling and Simulation,* SCS, San Diego, CA, 2000, pp. 93-98.

[27] J.-F. Zhang, J. Jiang, and B.K. Szymanski, ``A Distributed Simulator for Large-Scale Networks with On-Line Collaborative Simulators," *Proc. European Multisimulation Conference*, vol. II, pp. 146-150, Warsaw, Poland, June 1999, Society for Computer Simulation Press, Brussels, Belgium, 1999.

[28] X. Zeng, R. Bagrodia, M. Gerla, "GloMoSim: A Library for Parallel Simulation of Large-scale Wireless Networks," *Workshop on Parallel and Distributed Simulation*, pp. 154-161, 1998.

[29] T. Ye, S. Kalyanaraman, B. Mo, B.K. Szymanski, D. Harrison, B. Sikdar, H. Kaur, and K. Vastola, ``Network Management and Control Using Collaborative On-line Simulation," *Proc. IEEE Int. Conference on Communications ICC2001*, IEEE Computer Science Press, Los Alamitos, CA, 2001, Helsinki, Finland, June 2001.

[Web1] DaSSF – http://www.cs.dartmouth.edu/~jasonliu/projects/ssf/

[Web2] http://www.isi.edu/nsnam/ns/

[Web3] http://nile.wpi.edu/NS/

[Web4] REAL – http://www.cs.cornell.edu/skeshav/real/overview.html

[Web5] SSFNet – http://www.ssfnet.org