

# Genesis: A System for Large-scale Parallel Network Simulation\*

Boleslaw K. Szymanski, Adnan Saifee, Anand Sastry, Yu Liu and Kiran Madnani  
Department of Computer Science, RPI, Troy, NY 12180, USA  
{szymansk,saifea,sastra,liuy6,madnak}@cs.rpi.edu

## Abstract

*We describe a novel approach to scalability and efficiency of parallel network simulation that partitions the networks into domains and simulation time into intervals. Each domain is simulated independently of and concurrently with the others over the same simulation time interval. At the end of each interval, packet delays and drop rates for each inter-domain flow are exchanged between domain simulators. The simulators iterate over the same time interval until the exchanged information converges to a constant value within the prescribed precision. After convergence, all the domain simulators progress to the next time interval. This approach allows the parallelization with infrequent synchronization.*

*The biggest challenge for this method is to ensure iteration convergence for protocols, such as TCP, that adjust source rate to the current network conditions. The main contribution of this paper is to demonstrate that by judicious design of the domain processing and information exchange, the proposed approach efficiently parallelizes network simulation with TCP flows.*

## 1. Introduction

The major difficulty in simulating large networks at the packet level is the enormous computational power needed to execute all events that packets undergo in the network [5]. Packets crossing the boundaries of the parallel partitions impose tight synchronization between parallel processors, thereby lowering parallel efficiency of the execution [4].

This paper reports the progress of our long-term research on developing an architecture that can efficiently simulate very large heterogeneous networks in near real time [10, 8]. Our approach combines simulation and modeling in a sin-

gle execution. The network parts, called domains, are simulated at the packet level concurrently with each other. Each domain maintains however the model of the network external to itself which is built by using periodical output from other domain simulations. If this model is faithfully representing the network, the domain simulation will exactly represent the behavior of its domain, so its model will be correct and can be used by other simulations. Each domain simulation repeats its execution each time adjusting its model of the rest of the network and feeding the other simulations with increasingly precise model of its own domain. As a result, all domain simulations collectively reach a convergence to the consistent state of all domains and all models.

Our method can also be seen as a variant of a general scheme for optimistic simulation referred to as Time-Space Mappings proposed by Chandy and Sherman in [2]. Although all optimistic simulations can be viewed as variants of this scheme, very few apply, as we do, iterations over the same time interval to find a solution. In fact, we are aware of only one other network simulator [3] that uses such an iterative scheme. It, however, simulates a single switch with many sources, whereas our design targets simulations of thousands of routers interconnected into a network. Consequently, the parallelization techniques used by these two simulators are different.

## 2. Genesis Approach

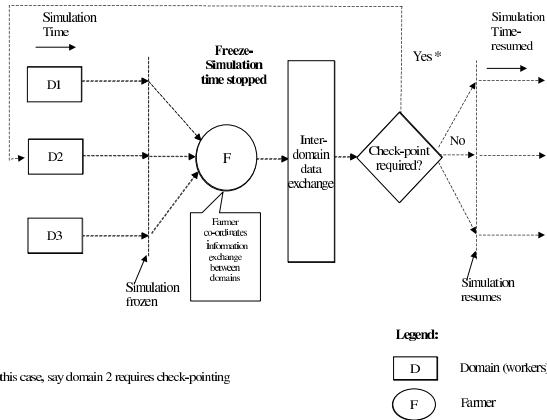
Although our approach has been described earlier [10, 8], we provide a brief summary here, to make the paper self-contained. The system is able to use different simulators in a single coherent network simulation, hence we called it General Network Simulation Integration System, or *Genesis* in short.

In Genesis, each network domain consists of a subset of network sources, destinations, routers and links that connect them. It is simulated concurrently with other domains and repeatedly iterates over the same simulation time interval, exchanging information with other domains after each iteration. In the initial iteration, each domain assumes ei-

---

\*This work was partially supported by the DARPA Contract F30602-00-2-0537 with the Air Force Research Laboratory (AFRL/IF) and by the URP of CISCO Systems Inc. The content of this paper does not necessarily reflect the position or policy of the U.S. Government or CISCO Systems—no official endorsement should be inferred or implied.

ther zero traffic flowing into it (when the entire simulation or a particular flow starts in this time interval) or the traffic characterization from the previous time interval. External traffic into the domain for all other iteration steps is defined by the activities of the external traffic sources and flow delays and packet drop rates defined by the data received from the other domains in the previous iteration step. The whole process is shown in Figure 1.



**Figure 1. Progress of the Simulation Execution**

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

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

Consider a flow from an external source  $S$  to the internal destination  $T$ , passing through a sequence of external routers  $r_1, \dots, r_n$  and internal routers  $r_{n+1}, \dots, r_k$ . The source of the flow is represented by the sequence of pairs  $(t_1, p_1), \dots, (t_m, p_m)$ , where  $t_i$  denotes the time of depar-

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

For sources that do not use feedback flow control, this approach faithfully recreates the dynamics of the flow [8]. However, to control congestion in network or internet, some protocols use congestion feedback. The most important among them is TCP protocol used in TCP/IP-based internet congestion control [7]. TCP uses sliding-window flow and error control mechanism for this purpose. The sliding-window flow control provides means for the destination to pace the source. The rate at which the source can send data is determined by the rate of incoming acknowledgment packets from them previous window. Any congestion on the path from the source to destination will slow down the data packets on their way to destination and the acknowledgment packets on their way back to source. As a result, the source will decrease it flow rate to decrease or eliminate this congestion. This mechanism is by no means reliable because TCP does not measure the congestion directly but deduces its presence from the round trip time for data and acknowledgment packets. In addition, there are no means in TCP to establish collaboration between different sources contributing to the congestion at the particular router. As a result, TCP flows demonstrate complex dynamics by adjusting their rate to the changing conditions on their paths to destination.

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

simulation of networks with TCP flows.

Our experience indicates that communication networks simulated by Genesis will converge thanks to monotonicity of the path delay and packet drop probabilities as a function of the traffic intensity (congestion). The speed of convergence depends on the underlying protocol. For protocols with no flow feedback control like UDP, simulations typically requires 2-3 iterations [8]. As presented in this paper, protocols with feedback based flow-control, like TCP, require number of iterations up to an order of magnitude larger than UDP-like protocols. However, our results were obtained with the fixed size of the time interval. We plan to investigate using the variance of the path delay of each flow to adaptively define the time interval to speed up convergence without severely affecting the parallel efficiency. When the delays change slowly, the time interval could be large before the convergence is affected. On the other hand, if the flow delays change rapidly, it should be small to decrease the absolute change of the traffic delay during a single iteration to speed up the convergence.

The efficiency of our approach is greatly helped by the non-linearity of the sequential network simulation. It is easy to notice that the sequential simulation time grows faster than linearly with the size of the network. Theoretical analysis supports this observation because for the network size of order  $O(n)$ , the sequential simulation time include terms which are of order:

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

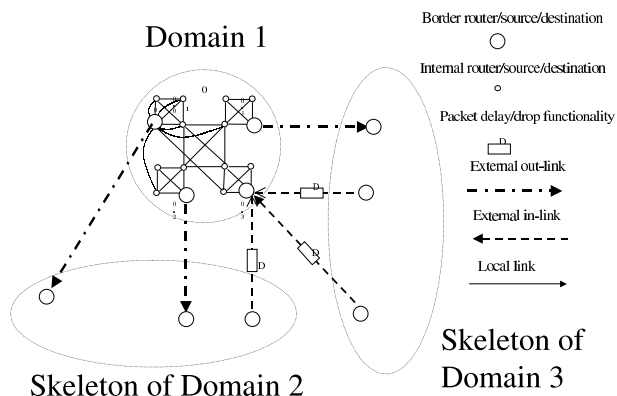
Some of our measurements [9] indicate that the dominant term is of order  $O(n^2)$  even for small networks.

We conclude that it is possible to speed up the sequential network simulation more than linearly by splitting it into smaller networks and parallelizing the execution of the smaller networks. As we demonstrate later, with modest number of iterations the total execution time can be decreased by the order of magnitude or more. Our primary application is the use of the on-line simulation for network management [9] to which the presented method fits very well, especially when combined with on-line network monitoring.

### 3. Design Overview

Genesis, though independent of the underlying simulator used, nevertheless requires extensions in the description of the simulation provided by the user as well as in the simulation system. The extensions presented in this paper were made specifically for the ns [6] system, probably the most widely used network simulator, thanks to its large number of implemented protocols. They also specifically support both UDP-like and TCP-like protocols.

The user is responsible only for annotating domains in the simulation configuration file. This is achieved simply by labeling each node in the configuration by the corresponding domain number. Based on these annotations, the extensions to the ns system process domain definition and its closure, collect the data for information exchange and implement the information exchange, as well as monitor convergence. A sample domain and its closure is presented in Figure 2 and discussed below.



**Figure 2. Active Domain with Connections to Other Domains**

Support for domain definition in Genesis, i.e., identifying which nodes belong to a particular domain, is the first step towards creating the domain closure. By definition, in the domain closure each external source proxy is directly connected to the destination domain of its flow. We refer to such replicated source as a *source proxy* and we call the link that connects it to the domain border router an *in-link proxy*.

The design supports the selective activation and deactivation of domains. The purpose is to process entire simulation configuration on each participating processor, but then, during simulation, to keep active only one domain closure while maintaining the routing information for the entire simulation. This information is needed in identifying the

destination domains for all packets that leave the domain.

Consider the sample network in Figure 2. The network is split into three individual domains, numbered 1, 2 and 3. Packets that flow into the domain from outside (with sources in skeletons of domains 2 and 3 in Figure 2) are produced by their source proxies in the domain closure and delayed or dropped during transition through in-link proxies (marked by boxes in Figure 2).

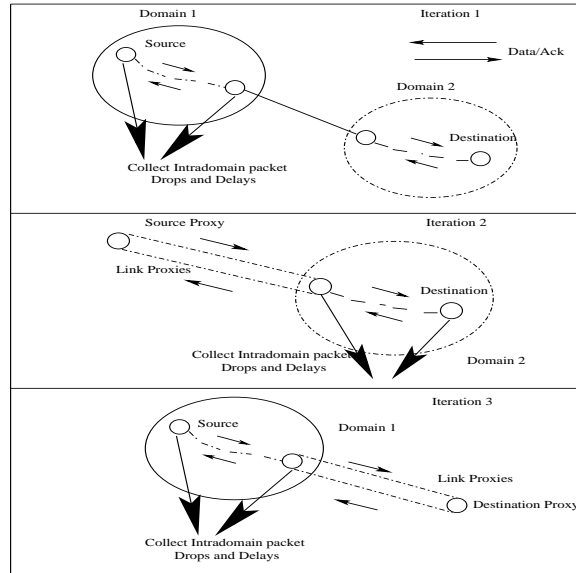
Exchange of data uses a Farmer-Worker collaboration model in which one processor collects the data from all the others and redirects them to all the simulators. This simplifies design of the data exchange but is not the most efficient solution, especially because the simulator rarely finishes each time interval simulation at the same time. Scalability of the solution, and the large communication latency when the distributed rather than parallel environment is used, favors the tree-like data exchange design in which constant size group of processors report to a next level representative. This is the organization that we are currently implementing.

Recording the information needed for data exchange involves calculating, for each packet leaving the domain, the time expired from the instance a packet leaves its source to the time it reaches the destination proxy. Also recorded is information about each packet source and its intended external node destination as well as whether the packet was dropped by a router inside the domain.

This approach is intuitive and works well for protocols that generate packets without feedback flow control, such as Constant Bit Rate (CBR), UDP and others. However, modeling the inter-domain traffic which uses feedback based flow control, such as one of many variants of TCP, requires more processing capabilities. The process of modeling such traffic is shown in Figure 3 and involves the following steps.

1. In the first iteration, the packets with a source within the domain and destination outside that domain flow along the path defined by the network routing through internal links to the destination proxy. We refer to such packets as DATA packets. The same internal links also serve as the path for the flow feedback, that is acknowledgment, abbreviated as ACK, packets. The timing and routing information of both kinds of packets (DATA and ACK) within the domain are collected at the flow source and the destination proxy.
2. In the second iteration, the timing and routing information collected at the source domain is used to create a source proxy and in-link proxy in the destination domain. The source proxy in the destination domain is activated and the traffic external to the domain and entering the domain is simulated using information collected in the first iteration in the source domain. In addition, the timing and routing information within the destination domain for packets flowing to external

nodes is collected at the destination proxy. This information will be used in the source domain to define the source domain in-link proxy that will reproduce ACK packets and send them to the original flow source.



**Figure 3. Increased number of iterations to support feedback-based protocols**

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

Note, that unlike the traffic without feedback control that uses one iteration delayed data to model traffic in the destination domain, delay here is two iterations. That is, the ACK packet traffic in the source domain in iteration  $n$  is modeled based on information from  $n - 1$  iteration about the ACK packets produced by the DATA packet flow that was modeled using information from  $n - 2$  iteration about the DATA packets in the source domain. Hence, there is delayed feedback involved in the convergence in this case, since an extra iteration is required to recreate the in-link proxy and source proxies in both the source and the destination domains.

In the following section, we described in some detail the implemented Genesis extensions in the context of ns simulator [6].

### 3.1. NS Simulator Specific Support in Genesis

Several changes were introduced to NS simulator to enable Genesis integration of domains. The initial implementation of these changes for UDP traffic has been described in [8], here we list them with the remarks what changes were necessary for TCP traffic.

1. Domain definition was introduced as a Tcl-level scripting command that is used to define the nodes which are part of the domain for the current simulation. The domain closure for TCP traffic is extended by adding proxies for all external TCP destinations that will produce ACK packets sent to the sources within the domain. Border is the most important new class added to the ns. A border object represents the active domain in the current simulation.
2. Connector is responsible for receiving, processing and then either delivering the packets to the neighboring node or dropping them. To support protocols with flow-control, modifications have been made to support sending feedback information in the form of ACK packets and collecting information about those packets sent from the external TCP destination. Connectors with target proxies abbreviate the path from the boundary of the domain to the destination by providing direct connector from there to the target proxy.
3. Traffic generator is used to generate traffic flows according to a timer. ACK traffic generators are modified to activate/deactivate sending packets, depending on the position of the source and domain and the iteration number (see Figure 3).
4. Link proxies are used to connect the source proxies to a particular cross-link on the border of the destination domain both for DATA and ACK packets.
5. Synchronization of individual domain simulations is based on messages sent to the server and preceded with *Message Identification Parameters* which identify the state of the simulation. A decision whether to checkpoint the current state or to restore the saved state is made by each domain simulation based on the comparison of flow delays and packet drop rates in two subsequent iterations. Diskless checkpointing enables the simulation to easily iterate over the same simulation time interval.

The steps of collaboration of simulators and the server are shown in Figure 1.

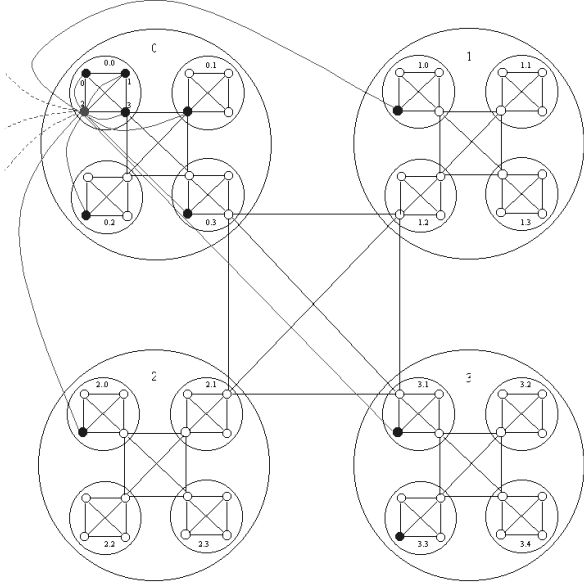
## 4. Performance

Our tests for the Genesis involved three sample network configurations, one with 64 nodes and 576 TCP flows, the other with 27 nodes and rich interconnection structure with 162 TCP flows and the third one with 256 nodes and 3072 TCP flows. These networks are symmetrical in their internal topology. We simulated them on multiple processors by partitioning them into different numbers of domains with varying number of nodes in each domain. The rate at which packets are generated and the convergence criterion are varied to give a wide-range of values for the speedup and accuracy of the simulation for both non-feedback and feedback-based protocols. To test the Genesis performance at the borders of the domain, temporal congestion is introduced by varying the packet generation rate along links leaving the domain. Other performance measurement parameter introduced is the ability to average out delay information over the previous iterations. This helps to achieve faster convergence to the solution and decrease the number of the iterations over the same time interval.

For 64 and the 256-node networks, the smallest domain size is four nodes; there is full connectivity between these nodes. Four such domains together constitute a larger domain in which there is full connectivity between the four sub-domains. Finally, four large domains are fully connected and form the entire network configuration for the 64-node network. On the other hand, 16 of such large domains are grouped together to form the entire network configuration for the 256-node network (cf. Figure 4).

The 27-node network is an example of Private Network-Network Interface (PNNI) network [1] with a hierarchical structure. The Private Network-Network Interface protocol suite is an international draft standard proposed by the ATM Forum. The protocol defines a single interface for use between the switches in a private network of ATM (asynchronous transfer mode) switches, as well as between groups of private ATM networks. The main feature of PNNI protocols is *scalability*, because the complexity of routing does not increase drastically as the size of the network increases. Thus, this is an interesting test case for Genesis approach. PNNI network smallest domain is composed of three nodes. Three such domains form a large domain and three large domains form the entire network (cf. Figure 5).

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



**Figure 4. A fragment of 256-node configuration showing flows from a sample node to all other nodes in a network**

#### 4.1 256-node network

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

Each node has twelve flows originating from it. Symmetrically, each node also acts as a sink to twelve flows. The flows from a node  $x.y.z$  go to nodes:

$$\begin{array}{lll}
 a.b.c.(d+1)\%4 & a.b.c.(d+2)\%4 & a.b.c.(d+3)\%4 \\
 a.b.(c+1)\%4.d & a.b.(c+2)\%4.d & a.b.(c+3)\%4.d \\
 a.(b+1)\%4.c.d & a.(b+2)\%4.c.d & a.(b+3)\%4.c.d \\
 (a+1)\%4.b.c.d & (a+2)\%4.b.c.d & (a+3)\%4.b.c.d
 \end{array}$$

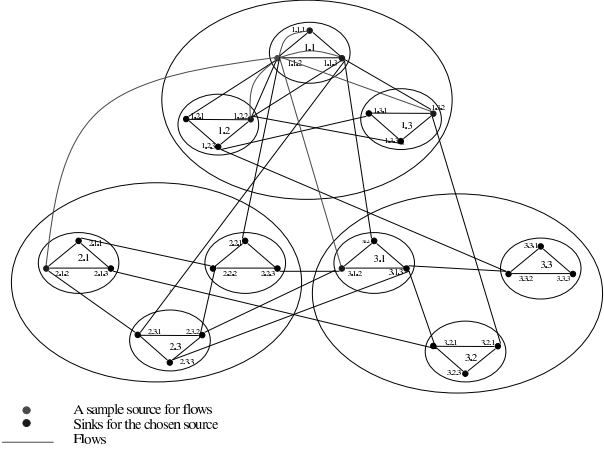
Thus, this configuration forms a hierarchical and symmetrical structure on which the simulation is tested for scalability and speedup.

#### 4.2. 27-node configuration

The network configuration shown in Figure 5, the PNNI network adopted from [1], consists of 27 nodes arranged into 3 different levels of domains containing three, nine and 27 nodes, respectively. Each node has six flows to other nodes in the configuration and is receiving six flows from other nodes. The flows from a node  $a.b.c$  can be expressed as:

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

$$\begin{array}{ll}
 a.(b+1)\%3.c & a.(b+2)\%3.c \\
 (a+1)\%3.b.c & (a+2)\%3.b.c
 \end{array}$$



**Figure 5. 27-node configuration and the flows from the sample node**

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

Domain	27-nodes	64-nodes
Network	3885.5	1714.5
Large	729.5	414.7
Small	341.9	95.1
Speedup	11.4	18.0

**Table 1. Measurements results on IBM Netfinity (times are in seconds) for non-feedback based protocols. Large domains contain 9 or 16 nodes and small domains contain 3 or 4 nodes**

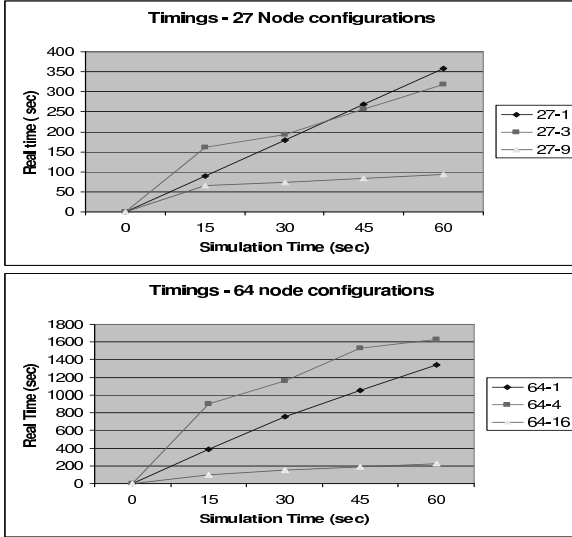


Figure 6. Speedup achieved for 27 and 64-node network for TCP traffic

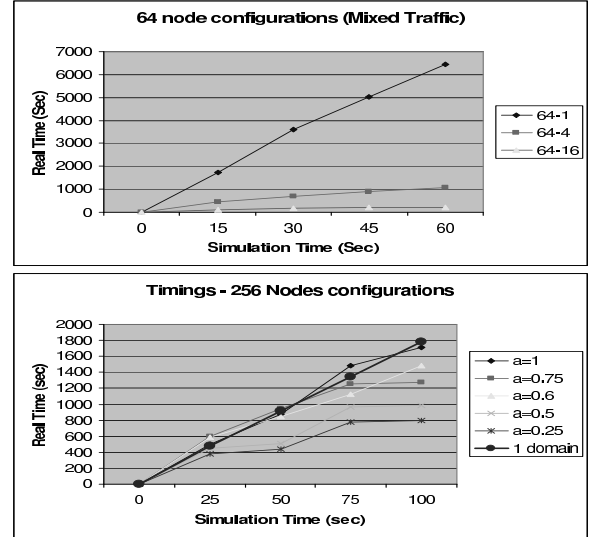


Figure 7. Speedup achieved for 64 and 256-node network for mixed traffic TCP-1/3 UDP-2/3

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

Table 1 presents a small subset of the timing results obtained from the simulation runs. It shows that partitioning the large network into smaller individual domains and simulating each an independent processor can yield a significant speedup.

Domain	27-nodes	64-nodes	256-node
Network	357.383	1344.198	1780.092
Large	319.179	1630.029	(A)
Small	93.691	223.368	799.267
Speedup	3.81	6.02	2.69

Table 2. Measurements results on IBM Netfinity (times are in seconds) for feedback based protocols (A)-indicates non-convergence due to domain size

For the non-feedback based protocols originally delays from the previous iteration were directly used in the next iteration, leading to modest speedup (cf. Table 2 and Figure 6). According to the analysis presented in [8], the fixed

point solution delay lays in between the delays measured in the two subsequent iterations. Hence, a delay for each flow used in the next iteration is a function of the delays from the current and previous iterations. As expected, using the this method of computing delay improves the Genesis performance. This is shown in Figure 7 for 64-node domains with mixed traffic. If  $d_{old}$  is the previous estimate of the delay, and  $d_m$  is currently observed values of the delay, then the new estimate of the delay is computed as  $d_{new} = a * d_m + (1 - a) * d_{old}$ , where  $0 < a < 1$ . Varying values of the parameter  $a$ , impacts the responsiveness of the delay estimate to new and old values of observed delays. As a result,  $a$  impacts the speed of the simulation by increasing/decreasing the time required for convergence with an optimum value at  $a = 0.5$ .

To measure the accuracy of the simulation runs, queue-monitors were placed along internal links along which congestion is most prevalent. These queue-monitors indicated that the number of packets dropped and the queue-sizes differed from the corresponding values measured over the sequential simulation of the entire network much less than 1% for both the feedback and non-feedback based protocols. Another measure of accuracy was based on the long range dependent behavior of aggregation of TCP flows that was expected to be self-similar. We calculated the Hurst parameter on selected links with heavy TCP traffic using rescaled adjusted range statistic and arrived at the same value of 0.699 for both a sequential simulation of the entire network and the Genesis domain-based simulation.

## 5. Future Work

In the paper, we demonstrated that Genesis approach works for complex network protocols such as TCP. Genesis delivered superlinear speedup for large network simulation on distributed computer architectures.

Several directions for improving efficiency of the current implementation include:

- adaptive selection of time interval based on a variance of the delay and packet drop rate of the inter-domain traffic,
- graph-theoretic based network partitioning algorithm that will optimize domain definitions by minimizing inter-domain traffic,
- non-constant model of the flow delay, for example using the linear model of the flow delay based on empirical data or using empirically collected delay time distribution should speed up convergence to the fixed point solution,
- aggregation of inter-domain flows passing through the same border router may improve efficiency by enabling replacement of many individual source proxies by a single aggregate proxy.

## References

- [1] Bhatt, S., R. Fujimoto, A. Ogielski, and K. Perumalla, "Parallel Simulation Techniques for Large-Scale Networks" *IEEE Communications Magazine*, 1998.
- [2] Chandy, K. M., and R. Sherman, "Space-time and simulation," *Proceedings of Distributed Simulation, 1989*, Society for Computer Simulation, pp. 53–57.
- [3] McGough, A. S., and I. Mitrani, "Efficient Distributed Simulation of a Communication Switch with Bursty Sources and Losses," In *Proceedings of the 14th Workshop on Parallel and Distributed Simulation*, pp. 85–92, May 2000.
- [4] Fujimoto, R.M., "Parallel Discrete Event Simulation," *Communications of the ACM*, vol. 33, pp. 31-53, Oct. 1990.
- [5] Law, L. A., and M. G. McComas, "Simulation Software for Communication Networks: the State of the Art," *IEEE Communication Magazine*, vol. 32, pp. 44-50, 1994.
- [6] ns(network simulator). See web site at [http : //www - mash.cs.berkeley.edu/ns](http://www-mash.cs.berkeley.edu/ns).
- [7] Stallings, W., *High Speed Networks: TCP/IP and ATM Design Principles*, Prentice Hall, Upper Saddle River, NJ, 1998.
- [8] Szymanski, B., Y. Liu, A. Sastry, and K. Madhani, "Real-Time On-Line Network Simulation," *Proc. 5th IEEE International Workshop on Distributed Simulation and Real-Time Applications DS-RT 2001*, August 13-15, 2001, IEEE Computer Society Press, Los Alamitos, CA, 2001, pp. 22-29.
- [9] Ye, T., D. Harrison, B. Mo, S. Kalyanaraman, B. Szymanski, K. Vastola, B. Sikdar, and H. Kaur, "Traffic Management and Network Control Using Collaborative On-line Simulation," *Proc. International Conference on Communication, ICC2001*, 2001.
- [10] Zhang, J. -F., 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, Society for Computer Simulation Press, 1999.