Proc. Applied Telecommunication Symposium, San Diego, CA, April 2002, SCS Press, 2002

Time-Network Partitioning for Large-Scale Parallel Network Simulation under SSFNet*

Boleslaw K. Szymanski, Qiuju Gu, and Yu Liu
Department of Computer Science, RPI, Troy, NY 12180, USA

Keywords: network, interoperability, distributed processing, domain decomposition, fixed-point solution.

1 Introduction

The Internet is unique in its size, support for seamless interoperability, scalability and affinity for drastic change. The collective computational power of all Internet routers involved in network traffic routing makes the Internet the most powerful computer in the world. Network packets are processed and routed in a very short time in the order of a fraction of a second. These very characteristics make the Internet hard to simulate efficiently.

We describe a novel approach to scalability and efficiency of parallel network simulation and demonstrate that it is able to use different network simulators as components of the same simulation supporting interoperability and scalability. The described method can be seen as a variant and modification of a general scheme for optimistic simulation referred to as Time-Space Mappings [1] where it occupies sparsely populated place of methods based on iterative approximation of a solution in spatially separate parts at the same simulation time.

Our approach partitions the network into parts, called domains. The simulation time is partitioned into disjoint intervals. Each domain is simulated independently of and concurrently with the others over the same simulation time interval. At the end of each interval metrics of inter-domain flows are exchanged between domains. The domain simulators iterate over the same time interval until the exchanged metrics converge to constant values within the prescribed precision. After convergence, all domains simulators move simulation to the next simulation time interval. This approach is particularly efficient if the simulation cost grows faster than linearly as a function of the network size, which is the case for computer networks in general and the Internet in particular [7].

The main result of this paper is to demonstrate that by judicious design of the domains and information exchange, the proposed approach efficiently parallelizes network simulation run under SSFNet and can interoperate with implemented earlier [6] ns-based simulations. The method is useful in all applications in which the speed of the simulation is of essence, such as: on-line network simulation, network management, ad-hoc network design, emergency network planning, or Internet simulation.

2 Genesis

Although our approach has been described earlier in the context of a simulator for verification of the approach [8] and as applied to ns-simulator [6], we provide a brief summary here, to make the paper self-contained. The system enables integration of different simulators into a 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 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 either zero traffic flowing into it from outside (when the entire simulation or a particular flow starts in this time interval) or the external 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

*This work was partially supported by the DARPA Contract F30602-00-2-0537 with the Air Force Research Laboratory (AFRL/IF) and by the grant from the University Research Program 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.

traffic sources and flow delays and packet drop rates defined by the data received from the other domains in the previous iteration step.

Each domain is simulated by a separate simulator which uses native traffic generator (either SSFNet or ns based [5], depending on the actual simulator used to for the domain) to create flows whose sources are within this domain. The simulator approximates flows with the sources that are external to the domain but are routed to or through the domain. This approximation is achieved as follows. In addition to the nodes that belong to the domain by the user designation, Genesis creates also *domain closure* that includes all the sources of flows that reach or pass through this domain. Since those are copies of nodes active in other domains, we call them *source proxy.* Each source proxy uses the flow definition from the simulation configuration file and the native traffic generator.

The flow delay and the packet drop rate experienced by the flows outside the domain are approximated by the random delay and probabilistic loss applied to each packet traversing in-link proxies. These values are generated according to the average packet delay and its variance as well as the observer 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 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 departure 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 simulation time. Hence, to replicate the flow with the source proxy P sending packets to router r_{n+1}, we just need to produce a sequence of packets $(t_1 + D_1, p_1), \dots (t_k + D_k, p_k)$, where $D_j = \sum_{i=1}^{n} d_{i,j}$, which in fact is creating the same sequence of packets as S does and delaying packet j by the time D_j. Each delay at the 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, the actual delay D_j can be approximated by randomly generated delay from the distribution with the same average value and variance as D_j and packet loss can be applied randomly with the probability defined by the observed frequency of the actual packet loss on the external path. These three values (average packet delay and its variance and the frequency of packet drop) are sent to the source proxy to be used in generating the flow. Thanks to the aggregated effect of many flows on queue sizes, this delay changes more slowly than the traffic itself, making such model precise enough for our applications.

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 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 simulation time include terms which are of order $O(n * \log(n))$, that correspond to sorting event queue, and $O(n(log(n))^2)$ 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 generate by each source and even the number of packets in each flow may grow at the rate $O(log(n))$ 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 [7] taken over the hierarchical networks 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 [7] to which the presented method fits very well, especially when combined with the on-line network monitoring.

3 SSFNet Framework

SSFNet is an open source, java based suite of models of protocols (IP, UDP, TCP, BGP4, OSPF and others), network elements (hosts, router, links) and assorted support classes for realistic multi-protocol, multi-domain Internet modeling and simulation [3]. SSFNet uses a java-based discrete-event simulation engine SSF [4] with well-defined Application Programmer Interface (API) [2], so all classes in SSFNet can interact with the underlying simulation engine transparently. In SSFNet, the network model is synthesized with DML (domain modeling language) and the simulation is instantiated with the configuration databases.

The principle classes of SSFNet are organized into two derivative frameworks: SSF.OS (for modeling of hosts and protocols), and SSF.Net (for modeling of connection, creating nodes and links). These classes are extended from five base classes of SSF. Metamorphism achieved by class inheritance and method overloading is the principle mechanisms for SSFNet to communicate with the underlying engine.

Entity serves as a container of components, which are aligned, to a common local simulation time. InChannel and outChannel are communication endpoints for event exchange between entities. Event is the base class for information exchange. Process is the base class for describing dynamic behavior of an entity. Each process is owned by a specific entity and may wait for event arrival on channels owned by its entity or wait for the elapse of simulation time, or both.

Package SSF.NET includes four principle classes: Net, Host/Router, NIC and link to model the network topology. Class Net is extended directly from SSF.Entity. It is the utmost container of all the network components. It loads DML configuration databases and controls the instantiation of the entire model, which includes hosts/routers and their installed protocols, links which connects hosts and/or routers, as well as traffic scenarios and so on.

Net is the container of all the network components. It keeps track of the references to all the hosts/routers configured. Similarly, every host/router is a container of its installed protocols; it keeps track of its attached interfaces. Link in SSFNet is implemented as direct channel mapping between interfaces attached to its endpoints. The receiver process of NIC is responsible for receiving incoming packets and pushing to its attached protocol. The queue manager of NIC enqueues or drops outgoing packets in an implementation-dependent manner. At IP level, a route or, more precisely, the interface (NIC) is chosen for every packet leaving the local host. The packets to the local host are assigned to proper sessions.

4 Genesis Extensions to SSFNet

In order to integrate SSFNet simulator into Genesis, SSFNet has been enhanced by the following additions: Domain Definition, Source and Link Proxy, Mapping Bridge, Data Collector, Checkpointing and Freeze that further discussed below.

4.1 Implementation of Decomposable SSFNet

In SSFNet, a network is modeled as a hierarchy of Net that is a collection of hosts, routers, links and component sub-Nets. Sub-Net inclusion is a powerful construct that facilitates building very large models from pre-configured sub-networks. Net is also a convenient tool for network partitioning required by Genesis. Hence, a domain definition is simply implemented by specifying the NHI of active domain at DML configuration databases. The modified Net class will retrieve its domain identifier from DML configuration database and store it at its global data area making it is easily accessible by other components.

Dynamic flow replication is implemented with source proxy and link proxy and controlled by the freeze component described later.

In SSFNet, when traffic starts, the client will first connect to the known port of the server. Then, the client sends control data (including the size of the file requested) and waits for data from the server. Once the server is initialized, it listens to incoming connections from clients. After accepting a new connection, the server builds a data socket and spawns a slave server that transmits the data between the client and the server. The interval over which packet are sent is controlled by a Timer that is an abstract class extended from Entity. It wraps itself in a Timer Event, and sends itself to its copy in the local channel with the appropriate timeout delay. At delivery, the Timer Event checks if it has been cancelled and if it has not, it executes its own call-back method. An application can overload the call-back method with appropriate application-specific event handler, and can control the behavior of Timer by invoking set or cancel methods explicitly.

If a traffic source is not in the current active domain, it will be suspended after its initialization. The slave server for this suspended traffic source is called a source proxy. The reference to a source proxy is registered at global area with corresponding flow identifier, so that it can be resumed by freeze. When a source proxy is resumed during the simulation, it starts to send packets via its link proxies. As shown in Figure 1, a source proxy will be connected directly to

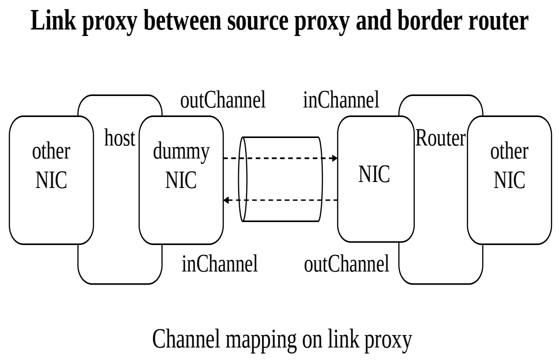
Link proxy between source proxy and border router
outChannel
inChannel
other NIC
host
dummy NIC
NIC
Router
other NIC
inChannel
outChannel
Channel mapping on link proxy

Figure 1: Proxy link implementation

border router by link proxies. Link proxies implement the link delay configured by the queue manager defined at the freeze time.

4.2 Collecting Data About Flows

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

Routing is done at IP-Level in SSFNet. By querying routing tables, IP class is able to pick up an appropriate NIC interface attached to the local host and put the outgoing packets into its output queue. Since link in SSFNet is implemented by channel mapping between the two attached interfaces, "push" invoked by one interface will put a packet into its peer's inChannel with appropriate delay. This mechanism is used for building link proxies which shortcut the path from source proxy to the corresponding border router of current domain. In addition, the IP class has also been enhanced to (i) sent outgoing data through link proxy instead of the normal route, (ii) dump information about outgoing data into Collector, and (iii) preserve the regular routing for control data. Some additional new features were added to IP Header and Host to facilitate the classifying of outgoing packets.

4.3 Simulation Freezing and Synchronization

In Genesis, the simulation time is partitioned into separate intervals and domain simulators iterate over each interval until all simulations converge. To enable reruns the whole simulator is cloned at the beginning of each interval. The cloned copy is activated when the rerun is necessary. We use JNI technology to do the memory checkpointing and the interaction between Java and C copy routines is shown in Figure 2.

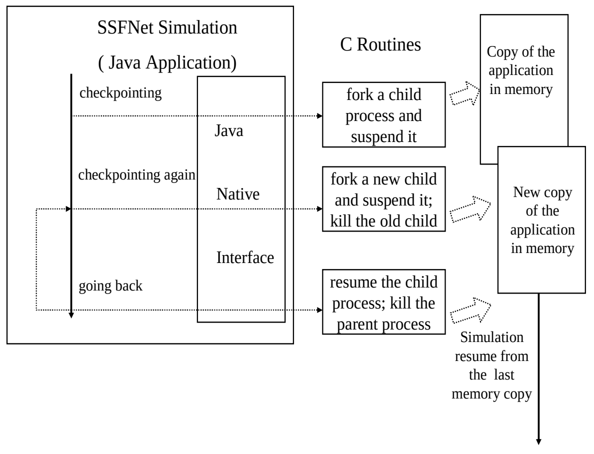
SSFNet Simulation
(Java Application)
C Routines
checkpointing
checkpointing again
going back
Java
Native
Interface
fork a child process and suspend it
fork a new child and suspend it; kill the old child
resume the child process; kill the parent process
Copy of the application in memory
New copy of the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

Figure 2: Checkpointing SSFNet simulation

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

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

4.4 Interoperability

Java-based SSFNet and C++/TCL-based ns [5] use different network models and different simulation frameworks. To facilitate interoperability between these two, we defined generic network model and flow-based message exchange format.

We added a new class Mapping to SSFNet that is instantiated by Net Object at initialization and uses two configuration files: nodes.conf and flows.conf. These files include information about generic network model of the network to be simulated. While exchanging information with the outside world, SSFNet simulators will query mapping tables to compose and parse messages into generic format. We also modified class ptpLinkLayer to facilitate the buildup of link proxies with generic flow message. While link is initialized, ptpLinkLayer can register the NIC references (call edge interface) of its two endpoints.

5 Experiments and Test Results

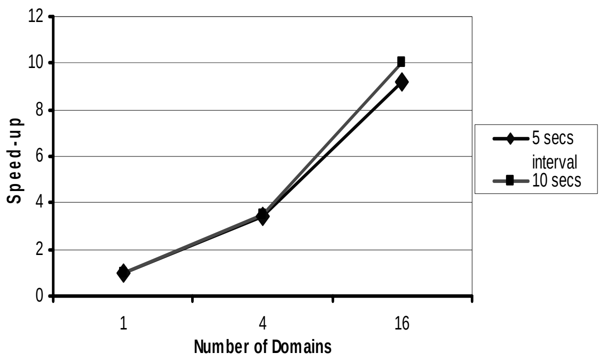

Figure 3: Speedup for 64-node simulation under different time intervals

We have measured the performance of the decomposable SSFNet in two experiments whose results are presented below. One compared flows in a single domain simulation with those in the multiple domains simulation. Another one involved a simulation in which ns and SSFNet interoperated. These experiment were performed on 27-node network and 64-node network described in detail in [6] and included hundred to thousand flows.

The speedup of the single iteration step for the 27-node PNNI network is shown in Figure 4. Note the speedup is slightly superlinear for 10 seconds of the simulation interval time. In actual simulation, the speedup is more modest because each iteration step is repeated several times before the simulation moves to the next time interval. The actual speedup for the pure SSFNet simulation for up to 16 processors is shown in Figure 3. Interestingly, using different simulators on different domains does not decrease speedup

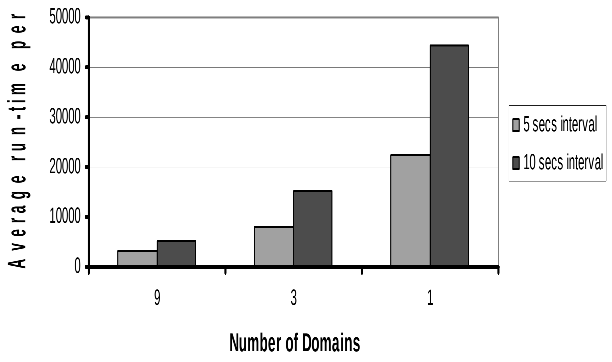

Figure 4: Single iteration step timings for 27-node PNNI network

significantly. Running half of domains under SSFNet and half under ns resulted in high parallel efficiency as shown in Figure 5.

The interesting question is how valid are results of the simulation, compared with the sequential execution. Hence, we compare corresponding flows in the sequential and four-domain simulations for 64-node network. The results are shown in Table 1 and demonstrate that the differences are small.

source	destination	number of domains	flow delay	drop rate
1.2.121	1.4.114	1	0.0389	0.571
		4	0.0366	0.553
1.2.121	1.4.114	1	0.0389	0.571
		4	0.0366	0.553
1.2.121	1.4.114	1	0.0389	0.571
		4	0.0366	0.553
1.2.121	1.4.114	1	0.0389	0.571
		4	0.0366	0.553
2.2.321	1.2.317	1	0.0867	0.714
		4	0.0878	0.905
3.2.121	1.2.118	1	0.0909	0.571
		4	N/A	1.0
4.2.121	1.1.319	1	N/A	1.0
		4	0.0924	0.975

Table 1: Flows in single and four domain simulations of 64-node network

6 Conclusion

In this paper we present a novel approach to large scale network simulation, that combines simulations

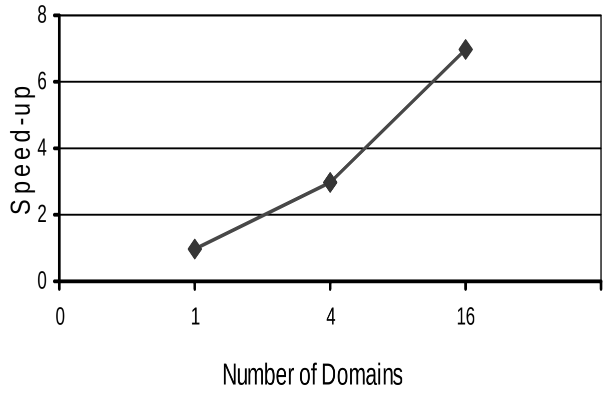

Figure 5: Speedup for 64-node network with SSFNet and ns domains

and models in a single system. Each model is fed by the data produced by the simulation and as the component simulations converge to the fixed point solutions so do the models based on them. We introduced this approach after questioning what it means to simulate a network. Since the correctness of a simulation cannot be verified at the individual packet level anyway, we adopted that view that the simulation needs only to produce network traffic with the same statistical properties as possessed by the traffic in the simulated network. Accordingly, we measure quality of the convergence of our iterative scheme by the divergence of the statistical properties of the traffic from the same properties of the fixed point solution. Thanks to this new approach, we are able to deliver superlinear speedup for network simulations running on distributed computer architectures. In the paper, we demonstrate that this approach works under SSFNet, a network simulator with different design that our initial implementation of Genesis that targeted ns. Moreover, we were also able to run the heterogeneous simulations in which ns and SSFNet domains interoperated efficiently.

Several possible 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] Chandy, K. M., and R. Sherman, 1989. "Space-time and simulation." *Proceedings of the Workshop on Distributed Simulation.* SCS Press, 53–57.

[2] J. H. Cowie, 1999. *Scalable Simulation Framework API Reference Manual*, Version 1.0. http://www.SSFNet.org/SSFdocs/ssfapiManual.pdf

[3] Cowie, J. H., D. M. Nicol, and A. T. Ogielski, 1999. "Modeling 100,000 Nodes and Beyond: Self-Validating Design." *Computing in Science and Engineering.*

[4] Nicol D., M. Goldsby, and M. Johnson, 1999. "Fluid-based Simulation of Communication Networks using SSF." *Proceedings of the European Simulation Symposium* (Erlangen-Nuremberg, Germany).

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

[6] Szymanski, B., Y. Liu, A. Sastry, and K. Madnani, 2001. "Real-Time On-Line Network Simulation." *Proceedings of the 5th IEEE International Workshop on Distributed Simulation and Real-Time Applications DS-RT 2001.* IEEE CS Press, Los Alamitos, CA, 22-29.

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

[8] Zhang, J. -F., J. Jiang and B. K. Szymanski, 1999. "A Distributed Simulator for Large-Scale Networks with On-Line Collaborative Simulators." *Proceedings of the European Multisimulation Conference*, vol. II. SCS Press, 146-150.