Integrating Feedback based Protocols to the Genesis system (Masters Project)

Adnan Saifee (saifea@cs.rpi.edu)

Abstract

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. The elementary objects of the traffic, network packets, are processed and delivered from one point to another in the network in a very short period of time of 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 network simulation and demonstrate that this solution can be applied to protocols that use traffic feedback to adjust the source traffic rate. The described method can be seen as a variant and modification of a general scheme for optimistic simulation referred to as Time-Space Mappings. This approach networks into parts called "domains".

It also partitions the 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 inter-domain flow delays and packet losses are exchanged. Domains iterate over the same time interval until the exchanged information converges to a constant value with the prescribed precision. After convergence, all domains start working on simulating the next time interval. This approach allows the parallelization with infrequent synchronization. It 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.

The biggest challenge for this method is to ensure iteration convergence for feedback protocols, such as TCP, that adjust source rate to the current network conditions. By a judicious design of the domains and information exchange, Genesis can efficiently parallelize network simulation with feedback flows.

The proposed method is useful in all applications in which speed of simulation is of essence, such as: on-line network simulation, network management, ad-hoc network design, emergency network planning, or the internet simulation.

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 [8]. The usual approach to providing required vast computational resources relies on parallelization of an application to take advantage of a large number of processors running concurrently. Such parallelization does not work efficiently for network simulations at the packet level because packets frequently cross the boundaries between partitions and have very short inter-event time (measured in milliseconds). Hence, the packets crossing the boundaries of the partitions impose tight synchronization between parallel processors, thereby ruining parallel efficiency of the execution [7]. The Internet, with its unique size, support for seamless interoperability, scalability and affinity for drastic change, is particularly difficult to simulate [6].

To avoid the pitfalls of packet level simulation parallelization, our approach combines simulation and modeling in a single execution [10]. In one view of this approach, 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 periodically 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 of this domain would be correct for other simulations to use. In this view, each domain simulation repeats its execution each time adjusting its model of the rest of the network and feeding the other simulations with ever more precise model of its own domain. This process repeats until all domain simulations collectively reach a convergence to the consistent state of all domains and all models.

This method can also be seen as a variant and modification of a general scheme for optimistic simulation referred to as Time-Space Mappings proposed by Chandy and Sherman in [2]. The simulation in their general scheme is viewed as proceeding in a multidimensional space in which one dimension represent the simulation time and the others spatial dimension of the application. This multidimensional space is then partitioned and a subset of these partitions is executed concurrently. The scheme is very general and do not specify how the partitions are defined, which partitions are processed concurrently or what to do in case an event (or to be precise an object and its associated events) passes between the partitions.

Our approach partitions the networks into parts that we call domains which consist of a subset of network sources, destinations, routers and links that connect them. Well defined domains should have as few links cut by the partitions (i.e. the links which the predecessor node is in one domain and the successor in another) as possible. The traditional network domains are convenient granules for such partitioning thanks to their well-defined points of entry and exit for inter-domain traffic. We also partition the simulation time into intervals. The resulting system enables integration of different simulators, hence we called it General Network Simulation Integration System, Genesis in short.

Each domain is simulated independently and concurrently with the others over the same simulation time interval. At the end of each time interval, domains exchange information about the flow delays and packet losses for traffic that crosses the domain boundaries. Based on this information, each domain creates a simple model of each flow entering it from others domains to faithfully simulate external and internal traffic in the domain. If the exchanged data differ significantly from the data received in the previous exchange regarding the same time interval, the domain simulator needs to re-execute the simulation over the same time interval, otherwise it can advance to the next time interval. In the initial iteration of the simulation process over the given time interval, each domain assumes either zero traffic flowing into it (when the whole 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 the packet delays and drops defined by the data received from the other domains in the previous iteration step. The whole process is shown in Figure 1

The selection of the proper time interval is crucial for the method efficiency and accuracy. Large intervals improve parallel efficiency by decreasing frequency of synchronization and information exchange between domains. Small intervals keep the network traffic characteristics little changed between iteration, speeding up the convergence and improving the accuracy of the results.

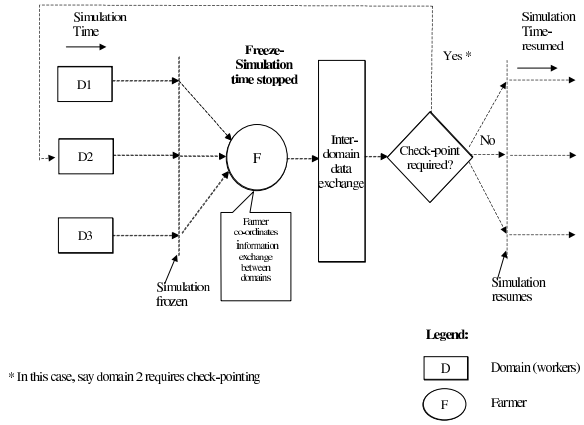

Figure 1: Progress of the Simulation Execution

To explain the scheme more precisely and to demonstrate why it converges to a solution, we provide the following, a bit more formal, description of the iteration process.

Consider a network $\Gamma = (N, L)$, where N is a set of nodes and L (a subset of Cartesian product $N \times N$), is a set of unidirectional links connecting them (bidirectional links are simply represented as a pair of unidirectional links).

Finally, consider a set of flows in the network, $\Phi \subset N \times N$, where each flow is defined by a pair of nodes ($n_{s,d}$) referred to as the source and destination of the flow. It is a function of the network to define a path from the source to destination nodes in the network. Typically, definition of the network as described above is provided as a configuration file for the network simulation. Such a configuration file often includes the start and end time of each flow and its type, protocol used and parameters needed to generate the flow during simulation. Our method requires additional definition which splits the network into the domains, which are just disjoint partitions of the network nodes into some q parts: $(N_1, ..., N_q)$. For each domain N_i, nodes in $N - N_i$ are called external. A closure of the domain N_i consists of the following nodes:

- all nodes in N_i, which do not have links to external nodes and are referred to as internal nodes of the domain,
- all nodes in N_i, which do have links to external nodes and are called the border routers of the domain,
- all sources of flows whose destinations are in N_i are called external source proxies of the domain,
- additional node called the external destination proxy, to which all flows whose sources are in N_i are directed.

The closure of the domain N_i includes also the following links:

- set of internal links defined as $L_i = L\&N_i \times N_i$,
- set of out-link proxies that, for each border router that routes a flow to the external node, connect it to the external node proxy (we will denote them by O_i),
- set of in-link proxies that connect external source proxy to the border router that receives this flow from an external node (we will denote them I_i).

Note that once the partition of nodes of the network is defined in the simulation configuration file, each domain closure can be computed either before the simulation (i.e., which nodes belong to each closure and its internal links) or during the simulation, when the routing of packets is known (all the proxy links).

Each domain closure is simulated by a separate simulator S_i. Hence, this simulator has full and exact description of the flows whose sources are within domain but needs to approximate flows that have the sources that are external nodes are routed to or through the domain. This approximation is achieved as follows. Each external source proxy uses the flow definition from the simulation configuration file. For sources that do not use feedback flow control, this approach will faithfully recreates the dynamics of the flow generation [10]. Otherwise, for sources of the TCP traffic, the quality of the approximation will

depend on the quality of the replication of the round trip traffic by the packets and their acknowledgments. The delay and the packet drops experienced by the flows outside the domain is approximated by the aggregate delay and packet drop applied to packet traversing the in-link proxies. These values are communicated to the simulator by its peers during the exchange of information performed at the end of simulation of each time interval. This requires that each simulator collects this data for all of its own out-link proxies, at the moment in which packets reach the external destination proxy.

This approach allows characterization of traffic outside each domain in terms of a few parameters changing slowly compared to the simulation time interval. In the implementation presented in this paper, we characterize inter-domain traffic as an aggregation of the flows, and each flow is represented by the activity of its source and the packet delays and losses on the path from its source to the boundary of the domain. Thanks to queuing at the routers and the aggregated effect of many flows on the size of the queues, the path delays and packet drop rates change more slowly than the traffic itself, making such model precise enough for our applications.

It should be noted that we are also experimenting with the direct method of representing the traffic on the in-link and out-link proxies as a self-similar aggregated traffic defined by a few parameters that can be used to generate the equivalent traffic using on-line traffic generator described in [14]. The difficulty of this approach is to reliably segregate the result into individual flows inside the domain.

No matter which characterization is chosen, the simulator can find the overall characterization of the traffic through the nodes of its domain. Let $\xi_k(M)$ be a vector of traffic characterization of the links in set M in k-th iteration. Each simulator can be thought of as defining a pair o f functions:

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

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

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

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

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

We believe that communication networks simulated that way will converge thanks to monotonicity of the path delay and packet drop probabilities as a function of the traffic intensity (congestion). For example, if in an iteration k

a domain N_i of the network receives for its destinations more packets than the fixed point solution would deliver, then this domain will be more congested by the flows arriving from outside. As a result, all packets, including the packets with destinations outside the domain, will experience inside the domain more delays and packet drops than under the fixed point solution. Hence, this domain would produce fewer packets from its sources than the fixed point solution would. These packets will create inflows in the iteration $k + 1$. Clearly then, the fixed point solution will deliver the number of packets that is bounded from above and below by the numbers of packets generated in two subsequent iterations I_k and I_{k+1}. Hence, in general, iterations will produce alternately too few and too many packets in the inflows providing the bounds for the number of packets in the fixed point solution. By selecting the middle of each bound, the number of steps needed to convergence can be limited to the order of logarithm of the needed accuracy, so convergence is expected to be fast.

The convergence is dependent on the underlying protocol. For protocols with no feedback control like UDP, on small network simulations requires 2-3 iterations [10]. As presented in this paper, For protocols with feedback based flow-control, like TCP, the number of iterations required is up to the order of magnitude larger. 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 is more important to decrease the time interval to decrease the absolute change of the traffic delay during a single iteration to speed up the convergence than to worry about the parallel efficiency..

The efficiency of our approach is greatly helped by the non-linearity of the network simulation. It is easy to notice that the simulation time of a network 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,
- of order $O(n^2)$, that result from packet routing, and
- even of order $O(n^3)$, that are incurred while building routing tables.

Some of our measurements [12] indicate that the dominant term is of order $O(n^2)$ even for small networks.

It follows from the above that the execution time of a network simulation may hold a quadratic relationship with the network size. Therefore, it is possible to speed up the network simulation more than linearly by splitting a simulation of large networks 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.

One of the desirable features of our design is its independence from the underlying simulators used to simulate the individual domains. Hence, our

architecture can be integrated with a number of existing technologies and has also a potential for fostering interoperability between them.

Our primary application is the use of the on-line simulation for network management [12] to which the presented method fits very well and can be combined with the on-line network monitoring. The simulations in this application predicts changes in the network performance caused by tuning network parameters. Hence, the fixed point solution found by our method is with high probability the point into which the real network will evolve. However, this is a still an open issue under what conditions we can guarantee that the fixed point solution is unique, and if it is not, when the solution found by the method is the same as the point that the real network reaches.

2 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 execution system. The extensions presented in this paper were made specifically for the ns [9] system, probably most widely used network simulator, thanks to its large number of implemented protocols. They also specifically support both UDP-like and TCP-like protocols. At the same time, the care was taken to make the extensions as generic as possible and work is underway to implement similar extensions to such widely different simulators as ssfnet [3] and glomosim [11].

The user is responsible only for annotating domains in the simulation configuration file. This is achieved simply by labelling 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.

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 is directly connected to the destination domain of its flow. We refer to such replicated source as an *external 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 keep active during the simulation only one domain closure and at the same time 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 example network in Figure 2 where the network is split into three individual domains, named 1, 2 and 3. Each of the domain simulations runs concurrently with the others and they exchange information about the path

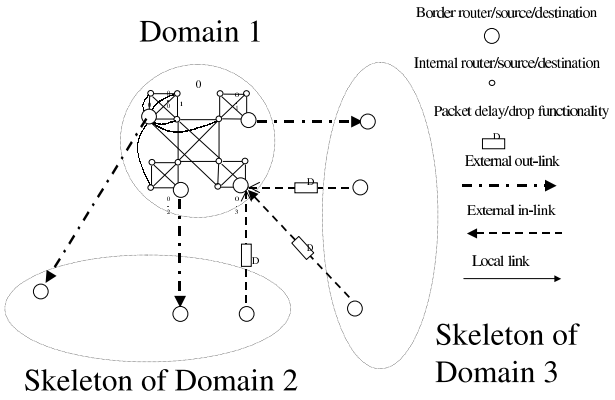

Figure 2: Active Domain with Connections to Other Domains

delays incurred by packets leaving the domain. The interval for exchange of this information is user configurable. In the figure, we assumed one second interval time, each domain may run its individual simulation from the simulation time n seconds to the simulation time $n+1$ seconds. After completion of this step, information about delays of packets leaving the domain during this time interval is passed onto the target domain to which these packets are directed. If these delays differ significantly from what was assumed in the target domain, the simulation of the time interval $(n, n+1)$ is repeated. Otherwise, the simulation progresses to the time interval $(n+1, n+2)$. The threshold value of the difference between the current delays and the previous ones under which the simulation is allowed to progress in time it is set by the user. This threshold impacts the speed of the simulation progress and defines the precision of the simulation results.

Exchange of data uses a Farmer-Worker collaboration model in which one processors 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 external 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.

Packets that flow into the domain from outside (with sources in skeletons of domains 2 and 3 in Figure 2) are produced by their external source proxies in the domain closure and delayed or dropped during transition through in-link proxies (marked by D boxes in Figure 2).

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 one of many variants of TCP, require more processing capabilities. The process of modeling such traffic is shown in Figure 7 and involves the following steps.

1. In the first iteration, the packets from a source, denoted as DATA packets, which are destined for a destination in another domain flow along the internal links of the domain as defined in the network configuration. These internal links also serve as the path for the 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 border router.

2. In the second iteration, the timing and routing information collected at the source domain is used to create an external source proxy and in-link proxy in the destination domain. This 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 this external domain. In addition, the timing and routing information within the destination domain for packets intended for external nodes is collected at the border router and in the destination domain. This information will be used in the source domain to recreate in-link proxy that will carry feedback information to the nodes in the source domain.

3. In the third iteration, in-link proxy and external 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 simulate 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 modelled on information from $n-1$ iteration about the ACK packets produced by the DATA packets that were modelled on 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 external source proxies in both the source and the destination domains.

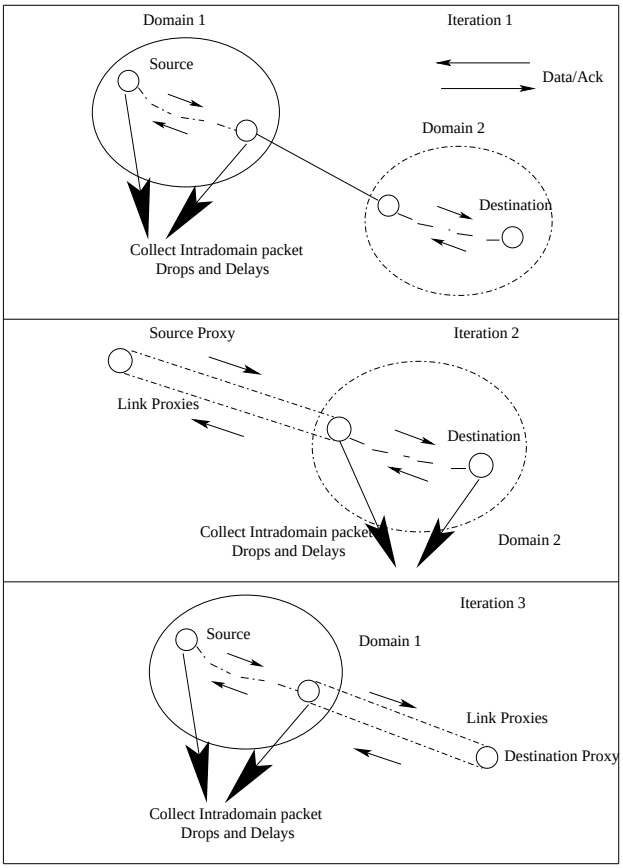

Figure 3: Increased number of iterations to support feedback-based protocols

For network configurations with transit domains, the proxy link creation process is more complicated. Each Transit domain has 4 proxy links attached to it as compared to 2 for the source and destination domains. Also the transit domains have to send the packet delay and drop rate information to both its neighboring domains.

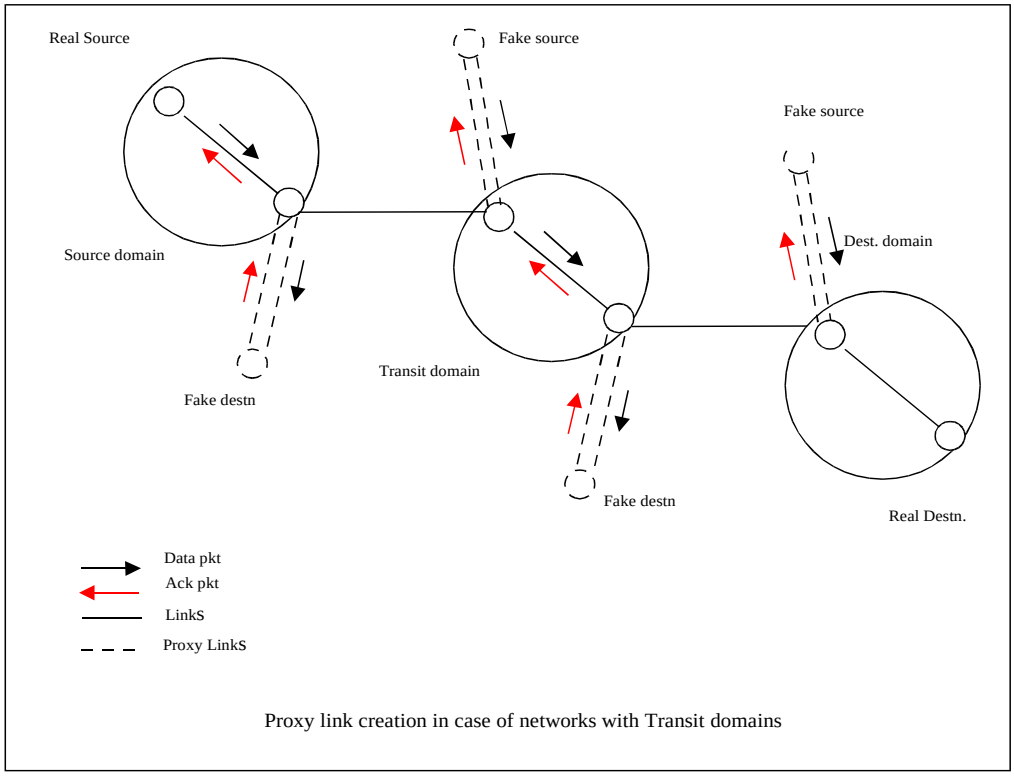

Figure 4: Transit domain: Proxy links and external source and destination proxies

In the following section, we described the implemented Genesis extensions in the context of ns simulator [9] which enables us to provide more details.

2.1 NS Simulator Specific Support for Genesis

2.1.1 Domain definition

Domain in ns is a Tcl-level scripting command that is used to define the nodes which are part of the domain for the current simulation. In the first iteration of the simulation the traffic sources outside the domain are inactive. The traffic generated within the domain is recorded and the statistics calculated. In the following iterations, the sources active within other domains with a link to the

domain in question are activated.

When a domain declaration is made in the Tcl script, the nodes defined as a parameter to this command are stored in the form of a list. Each time a new domain is defined, the new node list is added to a domain list (a list of lists). The user selected domain is made active. connected to a node in this domain and the other end connected to a node in another domain is defined as a cross-link. All packets sent on these links are collected for their delay and drop rate computation.

Source generators connected to sources outside the active domain are deactivated. This is done by a new Tcl script statement that attaches an inactive status to nodes outside the active domain (cf. Section 2.1.3 Traffic Generator description below).

2.1.2 Connector

The connector performs the function of receiving, processing and then delivering the packets to the neighboring node or dropping the packets. A modification has been made to this connector class which now has the added functionality of filtering out packets destined for the nodes outside the domain and storing them for statistical data calculation.

A connector object is generally associated with a link. When a link is set up, the simulator checks if this link connects nodes in different domains. If this is the case, this link is classified as a cross-link and the connector associated with this link is modified to record packets flowing across it. Each such packet is either forwarded to the neighboring node or is marked as leaving the domain based on its destination.

Additionally to support protocols with flow-control, some modification have been made to support feedback information in the form of ACK packets.

2.1.3 Traffic Generator

TrafficGenerator Class is used to generate traffic flows according to a timer. This class is modified, so that for the domain simulation, the traffic sources can be activated or deactivated. Initially, at the start of the simulation, the traffic generator suppresses nodes outside the domain from generating any traffic.

2.1.4 Link Proxies

Link proxies are used to connect the source proxies to a particular cross-link on the border of the destination domain. When a source proxy is connected to a domain by an in-link proxy, the packets generated by this source are sent into the domain via the in-link proxy and not the regular links which are set up by the user network configuration file. The link proxy adds a delay and, with certain probability, drops the packet to simulate packet's behavior during passage through the regular route. With the external source proxies and link proxies, the statistical data from the simulation of another domain are collected, and the traffic to the destination domain is regenerated.

When a link proxy is built, the source connector and the destination connector must be specified. A link proxy shortens the route between the two connector objects. Each connector is identified by the nodes on both ends of it. Link connectors are managed in the border object as a link list. The flow id to build up a link proxy is specified, one link proxy is used for one flow.

Link proxy is used to simulate a particular flow, so when the features (packet delay and drop rate) of this flow change, the link proxy object needs to be updated. After updating the parameters of the link proxy object, the performance of the corresponding link proxy changes immediately. Link proxies themselves are managed in the border object as a link list.

2.1.5 Connectors with External Target Proxies

In the original version of ns, connectors are defined as *an NsObject with only a single neighbor*. But our new ns simulation required this definition to be changed to build link proxies to shortcut the routes for different packet flows. These link proxies are set up according to the network traffic flows. Each flow from the external source proxies will need a separate link proxy. The flows that go through one source connector may reach different cross-link connectors at the destination border. Hence, there will be link proxies connecting this connector to some different connectors. Different flows going into one connector are sent to different link proxies, which are defined as external target proxies here. Thus, the connector could now be defined as *an NsObject with one neighbor and a list of external target proxies*. When the proxy connection is enabled in a connector, this connector has a list of link proxies (external target proxies). Thanks to that, the collector classifies the incoming packets by flow id and sends them to their destinations.

The connector class will maintain a list of external target proxies. Once a new link proxy is set up from this connector, it will be added to this connector's external target proxy list (this is done by the shortcut method defined in the Border class).

2.1.6 Border

Border is a new class added to the ns. It is the most important class in the domain simulation. A border object represents the active domain in the current simulation. The main **f**unctionality of the border class includes:

- Initializing the current domain: setting up the current domain id, assigning nodes to different domains, setting up the date exchange etc.
- Collecting and maintaining information about the simulation objects, such as a list of traffic source objects, a list of the connector objects and a list of the link proxy objects maintained by the border object.
- Implementing and controlling the traffic source proxies: setting up and updating link proxies, etc.

The border object is set up first, and its reference its made available to all objects in the simulation. A lot of other ns classes need to refer to the variables and methods in the border object. The border class has an array which for each simulation object stores the domain name to which this object belongs. This information is collected from domain description files that are created by the domain object implementation. The names are created for the files assigned to each domain to store some persistent data needed for inter-domain data exchange and restoration of the state from the checkpoint.

All traffic source objects created in the simulation are stored. These traffic sources can be deactivated or activated using the flow id. All the connector objects created in the simulation are stored. These connectors are identified by the two nodes to which they are connected. The connector information is used to create link proxies.

The traffic sources outside the current active domain are deactivated while setting up the network and domains. When a link proxy is set up for a flow, the traffic source of this flow will be reactivated. The border class searches the traffic source list to find the object, and calls the reactivate() method of the matching source object to reactivate this flow.

When the border receives flow information from other domains, it will set up a link proxy for this flow, and initialize the parameter of the link proxy using the received statistical data. When setting up a link proxy, it goes through the connector list to find the source and the destination of the connector objects, and then shortcuts the route between them by adding an external target proxy into the source connector. All the created link proxy objects are stored in the border as a linked list ready for further update.

2.1.7 Checkpointing

This feature has been included in Genesis to enable the simulation to easily iterate over the same simulation time interval. We use diskless checkpointing, in which each simulation process creates a child when it leaves a freeze point. The child is suspended, but preserves a state of the parent at the freeze time. The parent proceed to the next freeze point. Once there, the parent decides whether to return to the previous state, in which case it unfreezes the child and then kills itself, or to continue the simulation to the next time interval, in which case the suspended child is killed. This method is efficient because the process memory is not duplicated initially; later only pages that become different for the parent and child are duplicated during execution of the parent. The only significant cost is the execution of fork statement creating a child, which however is several orders of magnitude smaller than saving state to disk.

2.1.8 Synchronizing Individual Domain Simulations

Individual domain simulations are distributed across multiple processors using a client-server architecture. Multiple clients connect to a single server that handles the message passing between them. The server is defined as a single

process to avoid the overhead of dealing with multiple threads and/or processes. The server uses two maps (data structures). One map keeps track of the number of clients that have already supplied the delay data for the destination domain. The other map is toggled by clients that need to perform checkpointing. All messages to the server are preceded by *Message Identification Parameters* which identify the state of the client. A decision whether to checkpoint the current state or to restore the saved state is made by the client based on the comparison of packet delays and drop rates in two subsequent iterations.

A client indicates to the server whether it requires checkpointing in the contents of the message itself. A client which has to checkpoint causes all other clients to block until it has resent the data to the server and the server has delivered it to the destination domain (in other words a domain on another machine). This is achieved by exchanging the maps at the end of each iteration during the simulation freeze.

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

3 Genesis and the High Level Architecture

The High Level Architecture (HLA) provides the specification of a common technical architecture for use across all classes of simulations based on Parallel and Distributed Discrete Event Simulation [5]. The HLA generalizes and builds upon the results of the Distributed Interactive Simulations world and related efforts such as the Aggregate Level Simulation Protocol [4]. The central goal of the high-level architecture is to support interoperability among simulations utilizing different internal time management mechanisms. Real-world entities are modelled in the HLA by objects. Each object contains an identity that distinguishes it from other objects, state for the object, and a behavioral description that specifies how an object reacts to state changes. Each object identity and attribute pair has an owner that is responsible for generating updates to the value of that attribute (e.g., position information). Component simulations in the HLA are called federates and they have to adhere to the common standards to be able to interoperate with each other. The basic form of exchanging information between federates is *publish* and *subscribe* paradigm. Any federate wishing to provide information publishes it for others and this information is provided to all federates that subscribed to this service. For example, although at any instant, there can be at most one owner of an attribute, ownership of the attribute may pass from one federate to another during an execution. Any number of federates may *subscribe* to receive updates to attributes as they are produced by the owner who publishes them.

One of the objective of the GENESIS system is to provide interoperability between various network simulators. Essentially, it will be possible for Genesis to use the HLA or an idea of publish and subscribe paradigm for information exchange between domain simulators. However, Genesis provides its own infrastructure to support seamless exchanges of data between domain simulations for efficiency reasons. The individual domain simulations could be different in-

stances of the same simulator, as it was the case for the results presented in this paper, or completely different simulators. No special time management infrastructure is required. All the simulators run independently of one another and are controlled by their own local simulation clock. We provided a simple Common Message Interface (CMI) that standardizes the format of the exchanged data. We are planning also to replace the current centralized Farmer-Workers structure with distributed tree-like data exchange structure.

4 Performance

Our tests for Genesis involved 3 sample network configurations, one with 64 nodes, the other with 27 nodes and rich interconnection structure and the third one with 256 nodes. These networks are symmetrical in their internal topology. We simulated them on multiple processors by decomposing 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 Genesis performance at the borders of the domain, temporal congestion is introduced by varying the packet rate along links leaving the domain. Other performance measurement parameters 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.

The 64 and the 256-node networks are designed with a great deal of symmetry. 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 5).

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. There is an inherent trade-off between scalability versus quality of routing. PNNI networks can be used to study large scale simulations where two factors are critical to the simulation *performance* and *reusability*. The PNNI protocol is the most sophisticated signaling protocols devised to date, aimed at supporting QoS-based routing along with unprecedented levels of scalability. The PNNI protocols are challenging, both for modeling and efficient parallel execution. Thus, this is an ideal test case for our GENESIS

method. 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 6).

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. 3

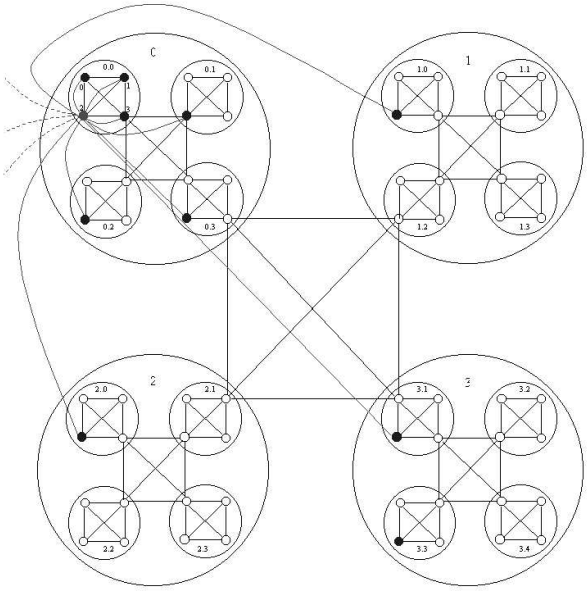

Figure 5: 256-node configuration showing flows from a sample node to all other nodes in a network

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:

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

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 6, 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:

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

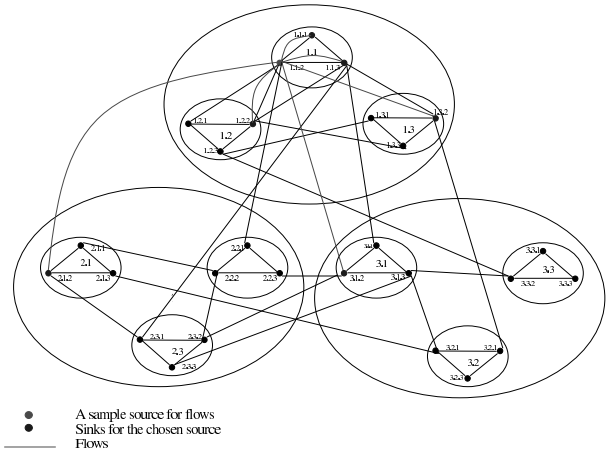

Figure 6: 27-node configuration and the flows from the sample node

In a set of measurements, the sources at the borders of domains produce packets at the rate of 20000 packets/sec 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/sec for the entire simulation. For these measurements we define the traffic source to be Telnet which generates packets with constant size of 500 bytes.

Speedup was measured in test cases involving only feedback-based protocols, only non-feedback based protocols and the mixture of both. The mixed protocol had traffic which consisted primarily of UDP (up to 66 percent), while the remaining traffic was TCP. Here TCP and UDP are used to simulate feedback and non-feedback based protocols respectively. An observation made was that for mixed traffic that involved a significant amount of non-feedback based traffic, there are fewer iterations over each time interval and hence greater speedup is achieved.

The table describes a small subset of the timing results obtained from the

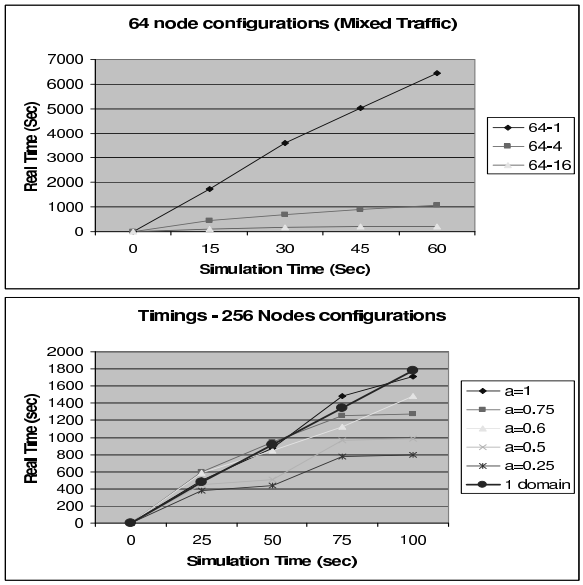

Figure 7: Speedup achieved for 64 and 256-node network for mixed traffic

simulation runs. This is to show that there indeed is a speedup by decomposing the large network simulation into smaller individual domain simulations each running on an independent processor.

Domain	27-nodes	64-nodes	256-node
Large	3885.5	1714.5	—
Medium	729.5	414.7	—
Small	341.9	95.1	—
Speedup	11.4	18.0	—

Table 1: Measurements results on IBM Netfinities (times are in seconds) for non-feedback based protocols

For the non-feedback based protocols originally delays from the previous iteration were directly used in the next iteration. But in the final design this method has been slightly modified according to the theoretical analysis presented in the introduction. A delay for each flow used in the next iteration is an average of the delays from the current and previous iterations. As discussed earlier, the value of the fixed point solution delay is expected to be in between of the delays measured in the two subsequent iterations. As expected, using the average delay improves the performance of our method.

Figure **??** shows Genesis speedup for 64-node domains with mixed traffic. Also it can be seen that varying values of 'a' the factor which propagates the

Domain	27-nodes	64-nodes	256-node
Large	357.383	1344.198	1018.533
Medium	319.179	1630.029	—
Small	93.691	223.368	378.518
Speedup	3.81	6.02	2.69

Table 2: Measurements results on IBM Netfinities (times are in seconds) for feedback based protocols

delay values through all iterations impacts the speed of the simulation increasing/decreasing the time required for convergence based on the value of 'a'. It can be observed in the ideal case delays obtained by an average of delays from the previous iteration and the current iteration gives ideal speedup.

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.

5 Future Work

In this paper we present a novel approach to large scale network simulation, which combines simulations 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. This approach challenges the very basis of simulating a network and suggests that because we cannot verify the correctness of the individual packets anyway, the correct simulation will produce traffic with the same statistical properties as 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 fixed point solution. Thanks to this totally new approach, we are able to observe superlinear speed up of network simulation on distributed computer architectures. In the paper we demonstrate that this approach works for simple as well as complex network protocols such as TCP that involve feedback based flow control.

One of the most immediate follow ups to this work is its integration with the Artificial Intelligence tool design for efficient search of network parameters that can improve the network performance [13]. The framework for such integration has been described in [12], yet a lot of work is needed to evaluate in what cases such automated network management can compete with the human network managers and operators.

Another interesting direction on which work is being done is interoperability of simulators running different domains. Currently the Genesis method is

being extended to ssfnet simulator, which is vastly more scalable and efficient than ns, and to glomosim [11] which was designed specifically for mobile and wireless networks. Since it is feasible and useful to simulate interactions of several different network domains using different technologies, it is important to be able to simulate each of these domains using a simulator best fitting the corresponding technology. Genesis provides simple and efficient interface for such collaboration. To support this goal we have design and implemented standard Message Identification Parameter format that can exchange data across all major network simulators.

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 as to minimize 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 external source proxies by a single aggregate proxy.

6 User manual

The additions and modifications to ns for GENESIS have been done on two platforms - Solaris and FreeBSD. This section gives the paths for the source code and the documentation.

Paths on Solaris:
Source code: /projects/tcpproj/adnan/ns-2.1b7a/
Scripts

- TCP based networks :
  /projects/tcpproj/adnan/ns-2.1b7a/tcptest

  Below every subdirectory for the configuration there are subdirectories for each domain specific script. eg. 27_3proc/ns1/ , 27_3proc/ns2 and 27_3proc/ns3 The server code is present in 27_3proc/server/ directory.
- UDP based networks:
  /projects/tcpproj/adnan/ns-2.1b7a/test The same directory structure is present for the above network configurations here.
- Documentation:
  /projects/tcpproj/adnan/ns-2.1b7a/newdocs

Configuration	Subdirectory
27 nodes, 1 domain	27_1proc
27 nodes, 3 domains	27_3proc
27 nodes, 9 domains	27_9proc
64 nodes, 1 domain	64_1proc
64 nodes, 4 domains	64_4proc
64 nodes, 16 domains	64_16proc
256 nodes, 1 domain	256_1proc
256 nodes, 4 domain	256_4proc
256 nodes, 16 domain	256_16proc

Table 3: Scripts and their directories

Paths on Netfinity:
Source code: /usr/home/saifea/ns-allinone-2.1b7a/ns-2.1b7a/
Scripts

- TCP based networks:
  /usr/home/saifea/ns-allinone-2.1b7a/ns-2.1b7a/tcptest
  Directory structure is same for the above mentioned network configurations.
- UDP based networks:
  /usr/home/saifea/ns-allinone-2.1b7a/ns-2.1b7a/test Directory structure is same for the above mentioned network configurations.
- Mixed Traffic based networks:
  /usr/home/saifea/ns-allinone-2.1b7a/ns-2.1b7a/mixtest
  Following configuration script exist:

Configuration	Subdirectory
16 nodes, 4 domains	16_4proc
64 nodes, 1 domain	64_1proc
64 nodes, 4 domains	64_4proc
64 nodes, 16 domains	64_16proc

Table 4: Scripts and their directories

7 References

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," *Proc. Distributed Simulation, 1989*, Society for Computer Simulation, pp. 53–57.

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

[4] Dahmann, J. S., R. M. Fujimoto, and R. M. Weatherly, "The DoD high level architecture: An update," *Proceedings of the 1998 Winter Simulation Conference*, 1998.

[5] Defense Modeling and Simulation Office, "High level architecture," http://hla.dmso.mil.

[6] Floyd, S., and V. Paxson, "Difficulties in Simulating the Internet," *IEEE/ACM Transactions on Networking*, Vol.9, No.4, pp. 392-403, August 2001.

[7] Fujimoto, R.M., "Parallel Discrete Event Simulation," *Communications of the ACM*, vol. 33, pp. 31-53, Oct. 1990.

[8] 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.

[9] ns(*network simulator*). http://www-mash.cs.berkeley.edu/ns.

[10] Szymanski, B., Y. Liu, A. Sastry, and K. Madnani, "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.

[11] UCLA Parallel Computing Laboratory and Wireless Adaptive Mobility Laboratory, "GloMoSim: A Scalable Simulation Environment for Wireless and Wired Network Systems," http://pcl.cs.ucla.edu/projects/domains/glomosim.html.

[12] 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.

[13] Ye, T., S. Kalayanaraman, "An Adaptive Random Search Algorithm for Optimizing Network Protocol Parameters," *International Conference on Network Protocols (ICNP), 2001*, submitted, see also http://www.cs.rpi.edu/ szymansk/sonms/icnp.ps.

[14] Yuksel, M., B. Sikdar, K. S. Vastola and B. Szymanski, "Workload generation for ns Simulations of Wide Area Networks and the Internet," *Proc. Communication Networks and Distributed Systems Modeling and Simulation Conference*, pp 93-98, San Diego, CA, USA, 2000.