The Genesis Interface to Distributed Wireless Simulation (Using GloMoSim)

Kiran Madnani
Rensselaer Polytechnic Institute, Troy, NY 12180, USA
email: {madnak}cs.rpi.edu

Abstract

The rapid advancement in portable computing platforms and wireless communication technology has led to significant interest in the design and development of protocols for instantly deployable wireless networks often referred to as "Ad-Hoc Networks". Ad-hoc networks are required in situations where a fixed communication infrastructure, wired or wireless does not exist or has been destroyed. The advantages span several different sectors of society. In the civilian environment, they can be used to interconnect work groups moving in a urban or rural area or a campus and engaged in collaborative operation such as distributed scientific experiments and search and rescue. In the law enforcement sector, applications such as crowd control and border patrol come to mind. In the military arena, the modern communications in a battlefield theater require a very sophisticated instant infrastructure with far more complex requirements and constraints than the civilian applications[3].

There are several challenges in the design and evaluation of these ad-hoc networks which make them unique. Some of these include the shared broadcast medium between thousands of nodes, the mobility of these nodes and the unpredictable nature of the wireless channel with problems such as fading, obstacles and interference.

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 parallel partitions impose tight synchronization between parallel processors, thereby lowering parallel efficiency of the execution[6]. In addition, in the case of ad-hoc networks, we deal with the possibility of nodes crossing the boundaries of the partition during the simulation.

The design of the GENESIS[8] technique used for simulating ad-hoc networks is described in the next sections. We implement the GENESIS technique over GloMoSim[9], a scalable simulation library for wireless network simulation developed at UCLA.

1 Introduction

1.1 Genesis Overview

Although our approach has been described earlier [10, 7], 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 either 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.

Each domain simulator creates all flows whose sources are within this domain by itself, but needs to approximate flows with the sources that are external to its 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 approxi-

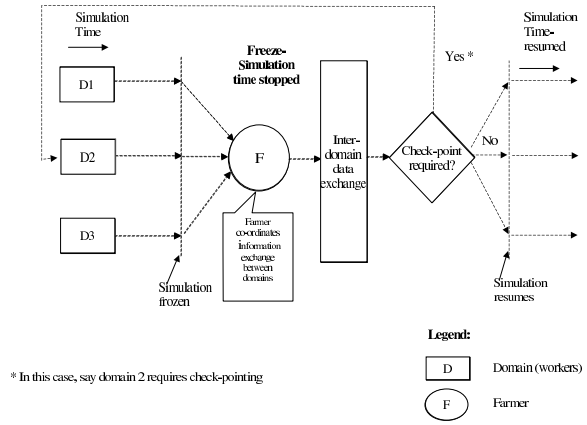

Figure 1: Progress of the Simulation Execution

mated 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 outlink proxies when packets reach the destination proxy.

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 can be approximated by randomly generated delay from the distribution with the same average value and variance as observed in the other domains 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 send 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. Some of our measurements [4] taken over the hierarchical networks indicate that the dominant term is of order $O(n^2)$ even for small networks. The impact of this observation on superlinear speedup of Genesis simulation is shown in Figure 2.

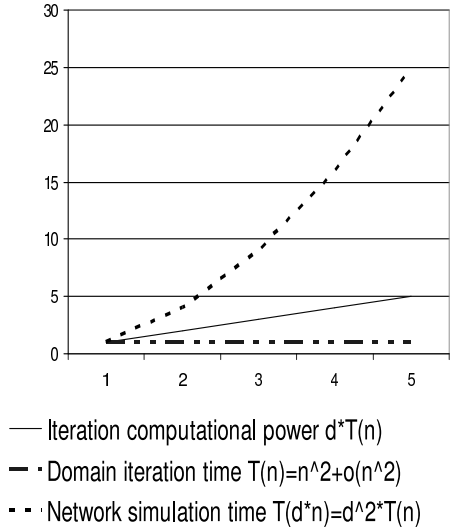

Figure 2: Analysis of Superlinear Speedup: with d domains of size n superlinear speedup will be achieved if the number of iterations $i < d$.

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. With modest number of iterations the total execution time can be decreased by the order of magnitude or more. Example of the superlinear speedup for 4 and 16 domain simulations of mixed TCP and UDP traffic is shown in Figure 3.

The extension of a wired network simulator needed for Genesis are quite standard and include the following components.

Domain Definition: which must be introduced by the user in the native simulation configuration file. It simply enumerates nodes belonging to each domain or labels each nodes with the domain identifier.

Source and Link Proxies: which are introduced by the Genesis based on the domain definition. Proxies belong to the domain closure. Source proxies represent sources that produce

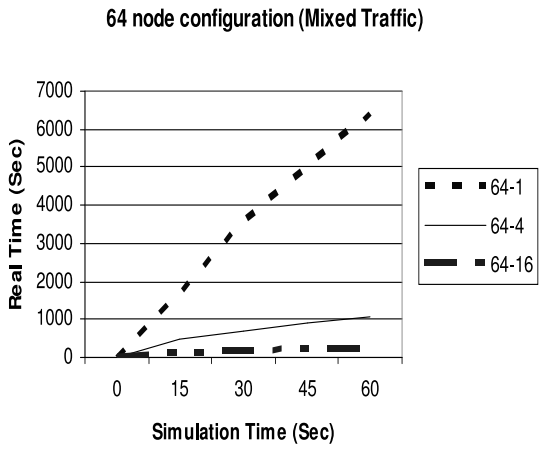

Figure 3: Simulation Times for 2,000 UDP and 1,000 TCP Flows in the 64-router Network Split into 4 and 16 Domains.

flows crossing or directed to the domain. Link proxies approximate packet delays and packet dropping on the path from the source to the domain boundary.

Data Collector: that is added to each flow leaving a domain. It collects the data about the packet delay and dropping from the flow source to the domain boundary.

Checkpointing and Freeze Event: that enable domain simulation synchronization. Freeze event causes all the simulators to stop at the same simulation time and it is added to the future event list by the Genesis system (this is the only new event introduced by Genesis). Checkpointing uses fast, diskless and application independent fork-based memory copy [2] to create a copy of each simulator at the beginning of each simulated interval. At the end of simulated interval, all domains either reactivate a copy (re-simulating the same interval but withe new data about external flow) or delete the copy and continue simulation into a new time interval.

1.2 Genesis Distributed Wireless Simulation Overview

The rapid advancement in portable computing platforms and wireless communication technology has led to significant interest in the design and development of mobile networks [3]. However, the management and evaluation of such networks presents unique challenges. Some of these include the shared broadcast medium between thousands of nodes, the mobility of these nodes and the unpredictable nature of the wireless channel with problems such as fading, obstacles and interference. The design overview of the Genesis interface to mobile network simulator, GloMoSim [9] is described in this section. The next few sections describe the design and implementation of this interface in detail.

As in the previous interfaces, we decompose the network into domains that contain network nodes, which in this case can be mobile. Thus, a domain is defined by the geographical area that it covers. The user's domain definition is a Genesis specific part of the GloMoSim configuration file. It contains also the radio-range of a node and using these parameters, Genesis computes the *closure* of the domain defined as the conjunction of the domain proper and its boundary regions of the radio-range width. The closure enables the system to account for nodes which lie outside the domain but still can interfere with the nodes inside it. Based on the domain closure, Genesis identifies the nodes active for each domain.

As in the Genesis interface to wired networks, domains are simulated concurrently with each other over the same time interval. The domains freeze [7] at user-specified intervals. At the time of freeze the inter-domain data exchange takes place. In GloMoSim, a node can schedule events (transmit and receive packets) while it is mobile. The current Genesis extension to GloMoSim accounts for the "mobility-trace" defined mobility in which the user specifies the speed, start and destination locations of the nodes in a configuration file. Knowing the above parameters, Genesis computes before the simulation the time and location at which the node crosses the domain boundaries. Using this information, each domain simulator knows when and where the mobile node will be active in its domain.

The introduction of domain closures creates regions in the network topology which overlap at least two domains. Thus, a node in such a region is active in both domains at the same time. The Genesis domain simulators which simulate activities of such a node must include the same events for the node. To achieve this, the inter-domain messages include information about communication (packets received and sent) by nodes lying in the domain-closure. Each domain receiving this information checks if the same communications

were executed for its copy of the nodes in question. If not, the time interval is re-simulated with the modified list of events for the offending node.

Each domain has at most eight domains as neighbors. Thus, each domain needs to communicate information about the activity of domains lying in its closure to its corresponding neighbor only. We achieve this by establishing a peer-to-peer connection between domains. In other words, each domain receives data from at most eight of domains during the freeze event. On exchange of this information, each domain checks whether it needs to go-back and re-simulate the freeze interval (based on the information collected and its own information).

All the domains must simulate the same time interval, that is, all domains must simulate the next iteration at the same time. In order to achieve this synchronizaion, a simple farmer-worker architecture is overlayed over all domains. Once each worker (from now on, we use the terms worker and domain interchangably unless otherwise specified) has made a decision to go-back/go-ahead, it informs the farmer about the same. The farmer then broadcasts a re-check signal to all the domains along with the domain id's of all the domains that need to go-back. This is to intimate all domains to re-check their logs if any of their neighbors have resimulated the freeze interval (this is to prevent cascading conflicts - a chain of conflicts in one domain which may lead to conflicts in another domain - explained in a section below). Thus, only when all the domains are ready to go-ahead, the farmer sends a sync-signal to all the workers. In other words, only when all workers send the farmer a go-ahead signal, the farmer in-turn sends them all a signal to go-ahead. Each worker waits needs to receive this sync from the farmer before it can go-ahead. Thus in this way, synchronization is based on messages between the farmer and worker, and the former is used to identify the state of the simulation.

The following algorithms briefly describes the Genesis model for distributed wireless simulations in a nutshell:
Each domain does the following till the end of simulation:

1. Send up to 8 messages to all its neighboring domains.
2. Receive logs from all of its neighbors and compare the received data with the corresponding logged data.
3. Detects conflicts based on the cases described in the section below.
4. In the event of a conflict, the domain which needs to take appropriate corrective action, will go-back and re-simulate the freeze interval and inform the farmer about the same(send a 1 signal to the farmer).
5. Else if it does not need to go-back, the domain will send a go-ahead signal to the farmer represented by 0.
6. Wait for a signal from the farmer. If the farmer sends a recheck signal, then the domain checks to see if its neighbor has gone back. If yes, it rechecks the logs of the neighbors with that of its own for any (cascading) conflict. If the domain then needs to go-back,it informs the farmer of the same. Else the domain waits for a sync signal from the farmer to go-ahead and simulate the next iteration.

The farmer does the following till the end of simulation:

1. Wait for a go-ahead/go-back signal from all workers(domains)
2. If any domain needs to go-back , the farmer sends a recheck broadcast to all the domains along with the domain id's of the domains that need to go-back.
3. Else the farmer busy waits until all workers can go-ahead.
4. Only when all workers can go-ahead, the farmer broadcasts a sync message to all workers to simulate the next interval.

2 Design details

2.1 Freeze scheduler

As in the Genesis approach to wired networks, a domain is simulated concurrently with other domains, that is, the domains iterate concurrently over the same time interval. The domains freeze[7] at user-specified intervals during which the inter-domain data exchange takes place.

Basically, a freeze is scheduled to pause the simulation at points of time that are assigned in the configuration file by users. When the simulation is frozen, the method GlomoFreeze() is called. All other functionality for interdomain data exchange and decision making (to go-ahead or to resimulate the interval) is called in this method. On return from GlomoFreeze(), the simulation resumes.

2.2 Domain Definition

In the Genesis extension to GloMoSim, we decompose the network into domains which contain nodes. However, unlike in the Genesis extension to wired networks [7][8] the nodes are mobile in this case. Thus, the domain definition is a function of the geographical area covered by the domains. The user is responsible for specifying the number and area of the domains in a configuration file. The radio-range of a node is also calculated based on the radio-model and parameters supplied by the user in the configuration file. Using these parameters, the closure of the domain is calculated. We define the closure of the domain as the sum of the domain boundary and the radio-range on all the four boundaries of the domain. We consider the closure of the domain in order to account for nodes which lie in this area (and are not part of the actual physical boundary of the domain) but are still within the radio-range to nodes lying in that respective domain. Our extension to GloMoSim uses this information to precompute the member nodes of the domain (from now on, we will use the term domain and domain closure interchangeably unless specified).

The present version of GloMoSim allows for only a single partition in the network. In the Genesis interface for distributed wireless simulations (as done in our previous implementations in ns2 and SSFNet), the domains are defined within this partition. These are defined in the config.in file (the place where user defines parameters) and are parsed into the driver.pc file. Thus, although the simulation would have only one partition, it will be allowed to have multiple domains. For each domain we specify its start and end x and y co-ordinates, in the domain.input file. Only one of the domains is kept active on each processor and only the nodes within this domain can send and receive the messages in the simulation on this processor.

As in the Genesis approach to wired networks, domains are selectively activated and deactivated. The purpose is to process the entire simulation configuration on each participating processor, and keep active only one domain closure while maintaining information of the neighbors of this domain.

2.3 Handling mobility of nodes

In GloMoSim, there are a number of ways to specify mobility[11]:

1. If MOBILITY is set to NO, than there is no movement of nodes in the model.

2. For the RANDOM-DRUNKEN model, if a node is currently at position (x, y), it can possibly move to (x-1, y), (x+1, y), (x, y-1), and (x, y+1); as long as the new position is within the physical terrain.

3. For RANDOM-WAYPOINT model, a node randomly selects a destination from the physical terrain. It moves in the direction of the destination in a speed uniformly chosen between MOBILITY-WP-MIN-SPEED and MOBILITY-WP-MAX-SPEED (meter/sec). After it reaches its destination, the node stays there for MOBILITY-WP-PAUSE time period.

4. For MOBILITY-TRACE, the mobility of nodes is specified in a trace file.

In the Genesis extension to GloMoSim, we concentrate on "mobility-trace" as it allows us to control the mobility of the nodes. In this case, the mobility of nodes is be specified in the configuration file. Thus, a node can schedule events (transmit and receive packets) when it is mobile. There are a total of 6 possibilities when a mobile node sends or receives a message from the nodes in the network. A node can send a message and move before an event is scheduled or during the event or after the event is scheduled to that node. Similarly, a node can receive a message and move before an event is scheduled for it or during the event or after the event is scheduled for it. However, instead of taking care of these "mobile" nodes at run-time, we can modify the scheduler during pre-simulation. Although such precomputing can be applied to all four methods of node mobility in GloMoSim, our current implementation deals only with "mobility-trace" method. In it, the user specifies the speed, start and destination locations of the nodes in a configuration file. Knowing the above parameters, we can calculate the time and location (x, y) when the node crosses the domain boundary (using the equation of the straight line - we assume that nodes always travel along a straight line). Using this information we modify the behavior of the scheduler to allow nodes to transmit data only when it is in the active domain. This gives rise to three cases:
a. The source and destination positions of the node both don't belong to active domain. In this case we calculate the times when the node enters and leaves the active domain. Based on these times, we schedule the time when the node can transmit packets.
b. The source position is in active domain while the destination location is not. In this case, we need to calculate the time when the node leaves of the active domain.
c. The source location is not in the active domain

while the destination position is in the active domain. Thus, in this case we need to calculate the time when the node enters the active domain.

The following example further clarifies this concept.

2.3.1 Example

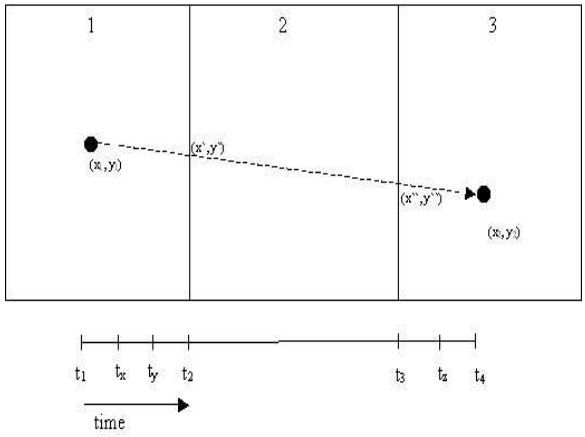

Figure 4: Example showing the mobility of a node

In Figure 4, we consider only one partition (as defined by GloMoSim) and 3 domains 1, 2, and 3 respectively. Now, consider a node 'A' at location (x_1, y_1) and moves at time to and moves to location (x_2, y_2) at speed, say s m/secs. Let A transmit data between times t_1 and t_x and t_y and t_z respectively.

Now, since we have both the start and the destination location, we can calculate the slope of the line path that A will take. Based on this slope, the speed at which it travels and the domain boundaries (also set in the network configuration file), we can calculate the points where A crosses the domain boundaries, (x', y') and (x", y") using the equation of the straight line. Based on this distance, we can calculate the times t_2 and t_3 when the node crosses these boundaries. Each instance of GloMoSim (that is, each active domain) will do this calculation to check is in its region during the time of simulation, thus giving rise to the 3 cases described above.

Using these times, the instance where domain 1 is active, the scheduler is not be changed as A completes its transmission before it crosses its boundaries. However, the instance of GloMoSim where domain 2 is active will change the time for which A starts and stops sending out messages to t_2 and t_3 respectively. Similarly, the instance where domain 3 is active, the scheduled start and stop times are changed to t_3 and t_z respectively.

Thus, only the active domain (an instance of GloMoSim running on a processor) checks if the mobile node crosses its boundaries during the simulation and if the node needs to send out data at the time when it is in that domain, the latter will modify the scheduler to make sure the node transmits data between the appropriate times. This entire process will take place during the initialization phase of the simulation and not during the run-time of the simulation.

2.4 Consistency of distributed domain simulations - handling conflicts

As per our definition of a domain-closure, there exists a region in the topology which overlaps in at least two domains. Thus, a node belonging to this area is active in both domains at the same time. We need to ensure that the domain (distributed) simulation faithfully represents the sequential simulation (on a single processor). To achieve this, the inter-domain messages only need to include information about communication (packets received and sent) by nodes lying in the domain-closure. More specifically, we take care of two cases described below. For each of these cases, assume that X is a node which lies in the overlapping region between two neighboring domains. Let X' be its "proxy" lying in the closure of the neighboring domain.

2.4.1 X and X' both receive data from different nodes in an overlapping time-interval

There are two sub-cases or overlapping receives, because a node can communicate only with nodes within its range. Thus, a node in a boundary region can have only nodes with a single domain on one side of its range, as shown in Figures 5– 6 .

1. In Figure 5, X receives from node Y in D, which does not have proxy in E and from a proxy Z' of node Z in E, while X' receives only from node Z. Here, the fact that X has two nodes sending and X' just one is not by itself a conflict. The conflict here is caused by the fact that Z' transmits to X after Y transmits to X in domain D. So actually, Y should have the right to transmit and not Z'. Here, X' incorrectly receives from Z, thus the conflict. In the correct scenario, both X and X' should behave consistently and receive from Y. In this case only X is aware of the conflict (between nodes Y and Z'), whereas X' is not(as it receives only from Z). Thus, when X' receives

the log from X (during the freeze interval), it becomes aware this conflict. Thus, domain E would need to re-simulate the freeze interval. During, re-simulation, X' would then take corrective action - when Z tries to transmit to it, it takes appropriate action of not allowing transmission of Z transmit to it.

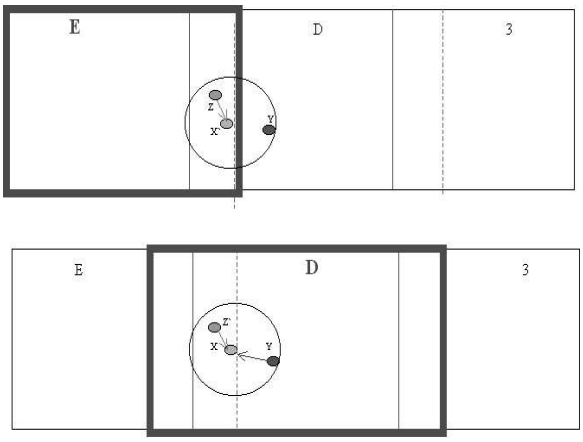

Figure 5: X and X' both receive data from different sets of nodes in an overlapping time-interval

2. As shown in Figure 6, X receives from node Y in D, while X' receives from proxy Y' in E and from Z in E, which does not have proxy in D. Here, the fact that X has two nodes sending and X' just one is not by itself a conflict. The conflict here is caused by the fact that Y' transmits to X after Z transmits to X in domain D. So actually, Z should have the right to transmit and not Y'. Here, X incorrectly receives from Y', thus the conflict. In the correct scenario, both X and X' should behave consistently and receive only from Z'(in other words Y should not be allowed to transmit in domain E). In this case only X' is aware of the conflict (between nodes Y' and Z), whereas X is not. Thus, when X receives the log from X' (during the freeze interval), it becomes aware this conflict. Thus, domain E would need to re-simulate the freeze interval. During, re-simulation, X would then take corrective action - when Y tries to transmit to it, X takes appropriate action of rejecting transmission.

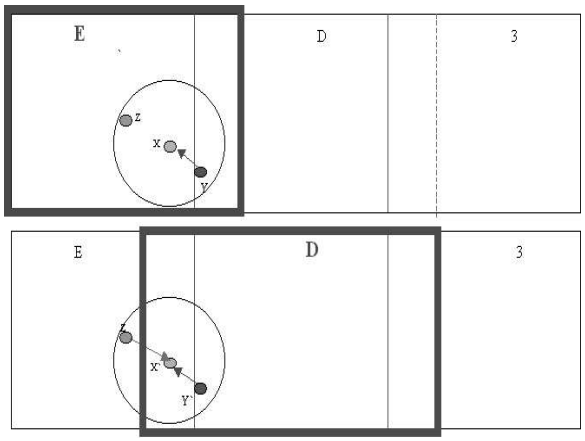

Figure 6: X and X' both receive data from different sets of nodes in an overlapping time-interval

2.4.2 X or X' but not both are receiving while the other node is transmitting at the same time

Again, here there are two sub-cases or overlapping receives, because a node can communicate only with nodes within its range. Thus, a node in a boundary region can have only nodes with a single domain on one side of its range, as shown in Figures 7– 8 .

1. In Figure 7, X receives from node Y in domain D, which does not have proxy in E while X' transmits to Z in E. In the correct scenario, X' should be transmitting to Z at this time. This is because, the start time of transmission of X' to Z occurs before X starts receiving from Y. Thus, when X receives logs from X', it would now know of the time of transmission of X' to Z. Thus, domain D would need to re-simulate the freeze interval. Thus, during re-simulation of that freeze interval X would not allow the transmission of Y to it.

2. As shown in Figure 8, X' receives from node Z in domain E which does not have proxy in D when X transmits to Y in D. In the correct scenario, X should be transmitting to Y at this time. This is because, the start time of transmission of X to Y occurs before X' starts receiving from Z. Thus, when X' receives log information from X, it would now know of the time of transmission of X to Y. Thus, domain E would need to re-simulate the freeze interval. Thus, during re-simulation of that freeze interval X' would not allow the transmission of Z to it.

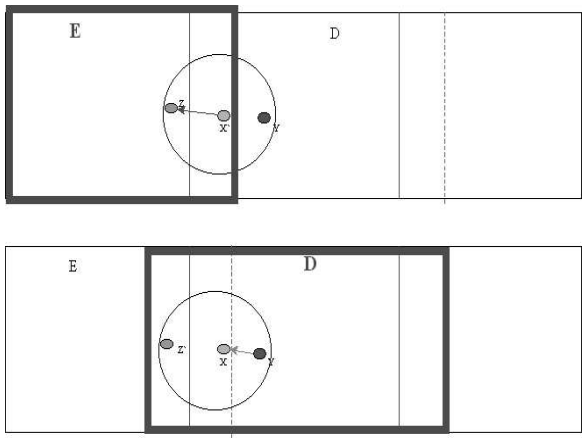

Figure 7: X or X' but not both are receiving when it is time to transmit

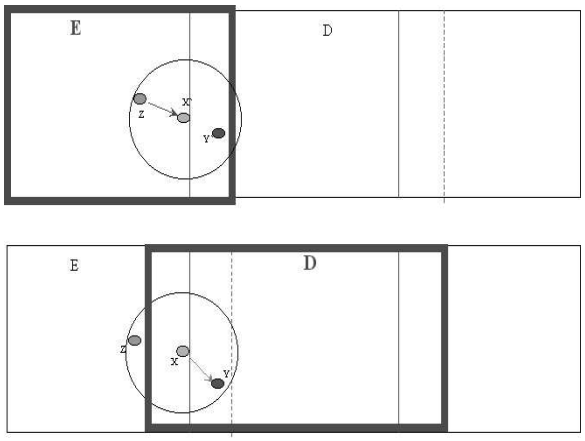

Figure 8: X or X' but not both are receiving when it is time to transmit

When the simulation freezes, the information of all nodes lying in the closure of a domain is sent to their respective neighboring domains. Each domain receiving this information validates the same for the two cases above. If any of the cases discussed above occurs, the domain decides (checkpoints) to resimulate that freeze interval after modifying the behavior of the node to make it consistent with its "proxy" in the neighboring domain. Diskless checkpointing enables the simulation to easily iterate over the same time interval.

2.4.3 Cascading Conflicts

In all of the cases above, when a conflict is caused, the domain in which the conflict is caused goes back and resimulates the freeze interval. During the resimulation, the events simulated by the nodes in conflict are corrected. However, this might lead to a chain of changes for all its neighbors and the neighbor's neighboring nodes causing an error in the neighboring domain as shown. For example, as shown in Figure 9, domain D needs to go-back. In the 1st iterarion of D, A transmits to B and Y transmits to X while the transmissions of Y to A and C to B are rejected over the same channel. However, due to a conflict, Y should not transmit to X. Thus, in the resimulation (2nd iteration), the transmissions of Y to A and thus C to B are allowed and go through. However, in the 1st iteration C(which is in the domain closure of Q) transmits to another node in Q. Thus, a cascading conflict is caused in the 2nd iteration of D.

In order to prevent such an error in simulation, the farmer broadcasts a re-check signal to all domains along the the domain id's of the domain that go-back. On receiving this signal, each domain will check to see if its neighbor has gone back. If yes, when it receives the logs of the events of the resimulation from its neighbor, it re-checks to see if the simulation is still faithful. If not, it will go-back to resimulate the same iteration after informing the farmer of the same. The farmer, in turn only broadcasts a sync signal to the workers when all of them are ready to go-ahead(that is, it receives a go-ahead signal from all workers as described in the algorithm above).

2.5 Genesis data exchange model for distributed wireless simulations

In the Genesis interface to GloMoSim, each domain has the knowledge of its domain boundaries as well as the domain id's of its neighbors (from the configuration file). Also, each domain has at most 8 domains as neighbors as shown in Figure 10. Thus, each domain needs to communicate information (about the activity of domains lying in its closure) to its corresponding neighbor only. We achieve this by establishing a peer-to-peer connection between domains. In other words, each domain receives data from at most eight of its neighbors during the freeze-interval.

However, in order to prevent cascading conflicts (described above) and to make sure that all the domains simulate the same interval (some of them may need to go-back) we need to synchronize between domains. We have implemented a server (farmer) for this purpose. Synchronization between individual domains is based on messages sent to the server, which identifies the state of the simulation.

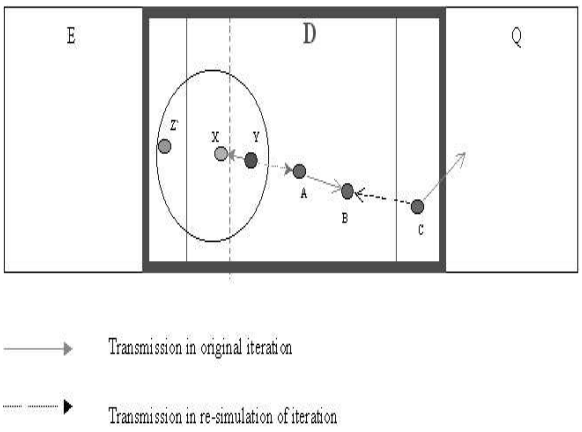

Figure 9: Chain of events in one domain causing a conflict in the neighboring domain - cascading conflict

3 Interoperability with other wired network simulation engines

As part of the Genesis project, we have previously demonstrated interoperability between Java-based SSFNet and C++/TCL-based ns2 which required the definition of a generic network model and a flow-based message exchange format [1]. We adopt a similar approach (described below) in order to demonstrate interoperability between SSFNet and GloMoSim. Our main objective is to create a scenario where we have mixed-mode traffic between a wired network (modeled using SSFNet) and a wireless network (modeled using GloMoSim).

In order to interoperate between the above mentioned simulators, our network configuration includes wired domains simulated by SSFNet and wireless domains simulated by GloMoSim. The SSFNet part of the network will view the wireless GloMoSim domains as a single node network, which is the fake source and sink for all traffic originating and destined respectively to the latter. Similarly, for GloMoSim, the SSFNet domains are represented by a single (fake) source and sink node. At each freeze interval, the information about delay-drop rates (just as in Genesis interoperability interface for ns2 and SSFNet) is exchanged for inter-domain traffic. This information about flows is exchanged only for cut-flows (as in ns-ssfnet interoperability). Based on this information and local conditions in the domain, a decision whether to go back or not is made by each of the domains (at each freeze interval). If all (wired and wireless) domains are convergent and can go ahead, the exchanged delay-drop

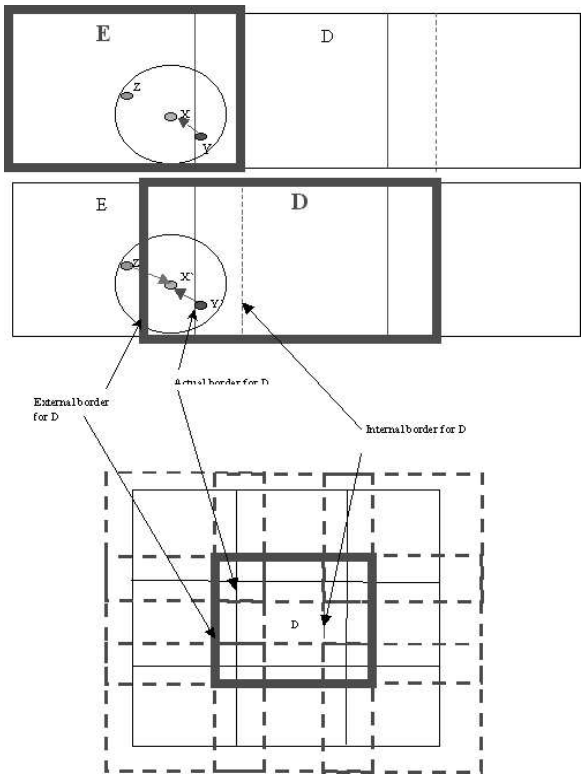

Figure 10: General Description of Domain D and its 8 neighbors

information is used by the the respective proxy sources to generate packets in the next freeze interval.

4 Experiments

In order to test the performance of the above devised interface, we configured a simple network topology. Initially, we considered a network with 5400 nodes over an area of 360,000 square meters. However, the sequential simulation ran out of memory for the above. Thus, we redesigned the network configuration to consist of 900 nodes over a geographical area of 225,000 square-meters and perimeter of 6000 meters (that is, it is a square of 1500 meters by 1500 meters). We assumed that the mobile nodes moved with a velocity of 50m/hour. The number of nodes which are mobile per second is 15 (for every node that left the domain, there would be a node that entered the domain from the corresponding opposite boundary). Figure 11 shows a sample scenario. Also for these experiments, we used CBR traffic. About 80 percent of the traffic was intra-domain, that is between nodes within a domain, and 20 percent of the traffic was

inter-domain. This was done in order to reduce cost of synchronization between domains.

4.1 Distributed Simulation Experiments

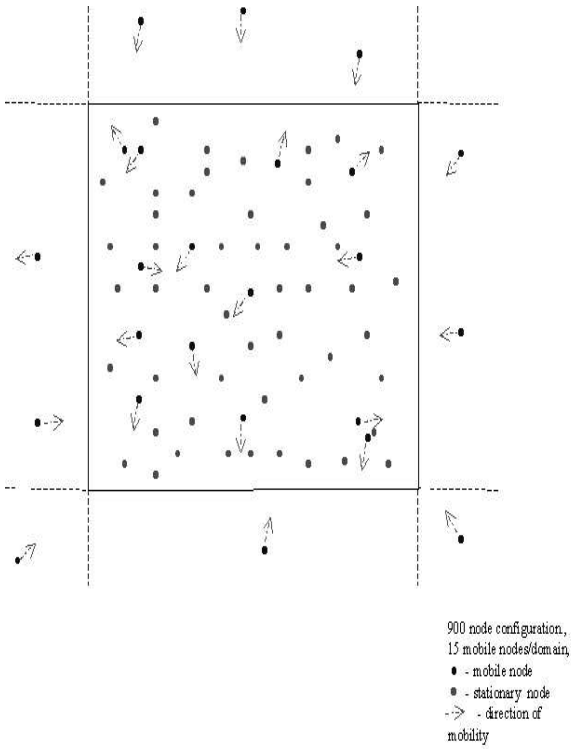

Figure 11: Sample network configuration showing 1 domain of the topology

We conducted tests for 3 different sizes of domains - 1 domain (vanilla GloMoSim), 4 domains and 16 domains. The simulation time was 200secs with a freeze interval of 20secs. The intensity of flows shown in the table 1 represents the time interval in seconds between the transmission of 2 packets. The graph in Figure 12 shows the results generated by these experiments. All of these experiments were carried out on a cluster of IBM Netfinity processors.

Figure 13, shows the speed-up for 4 and 16 domains respectively. The Genesis interface for distributed wireless simulations outperformed the sequential simulation to produce a speed-up of 4.2 and 14.7 for 4 and 16 domains respectively.

To measure the accuracy of the simulation runs, the flows of the sequential runs were compared with those of the parallel runs. We monitored the flow statistics of the distributed simulations with that of the sequential simulation. Since the number of nodes

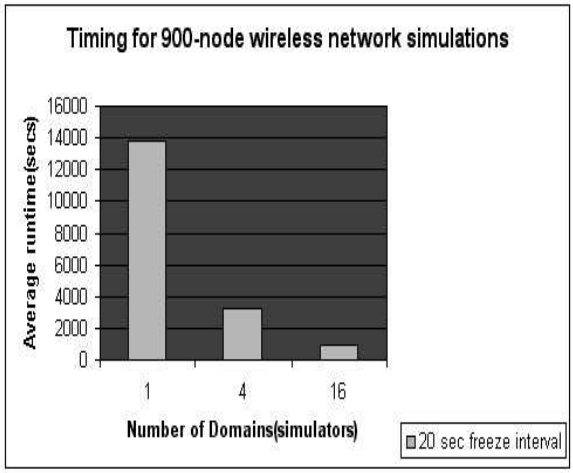

Figure 12: Real-Time of Simulation vs Domain Size

Number of Domains	1	4	16
Total number of flows/domain	1120	280	70
No. of internal flows/domain	1120	224	56
No. of External flows/domain	1120	56	14
Max./Min. Intensity of flows(s)	1/98	1/98	1/98
Avg. Time(s)	13827.6	3287.5	943.6
Speedup	1	4.2	14.7

Table 1: Measurements results on IBM Netfinities (times are in seconds)

in the closure of any domain is always small (these nodes are the ones which cause the inconsistency in the simulation), the differences in these statistics was observed due to these nodes was minimal too. Comparison of the throughput, drop-rates and delays indicated that the values of the distributed simulation differed by about 4.3 percent when compared to the corresponding of the sequential simulation as shown in table 2.

4.2 Experiments on Interoperability with SSFNet

To demonstrate interoperability between wired and wireless domains, we used the network configuration shown in Figure 14 that consists of 4 domains, each containing 4 nodes. Three of these domains are wired, simulated by SSFNet; while one domain contains wireless nodes, simulated by GloMoSim.

In order to demonstrate interoperability, we had 2 TCP flows, one each from a node in the SSFNet domain to the GloMoSim domain and vice-versa (as shown in Figure 14). For each flow, we short-cut it to

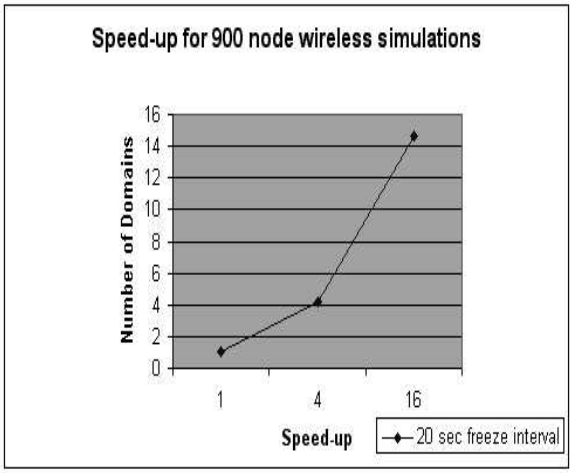

Figure 13: Speed-up vs Domain Size

Number of Domains	1	4
Avg. Throughput(bits/s)	2133	2183.7
Avg. Delays(s)	0.00273	0.00281
Avg % of packets dropped	2.10	2.19

Table 2: Comparison of parameters of sequential and parallel simulations to demonstrate faithfulness of distributed simulations

the proxy sink in the respective domain. During the freeze-interval, delay and drop information of these flows is communicated to the domain in which the destination domain lies. Each domain also makes a decision to go-ahead or back based on past and present parameters. Once all the domains can go-ahead, in the next iteration, the fake source in the destination domain makes use of this information to generate packets to the destination node.

For the described configuration, the average time the simulation took to execute for a stand-alone GloMoSim configuration was 10.52 seconds while the corresponding value for the SSFNet configuration was 8.43 seconds. The time of execution for the ineropearable configuration was 12.38 seconds as shown in table 3. Thus, the time of execution for the stand-alone configurations is comparable to that of the interoperable one. The latter result is greater because of the time taken for synchronization between the 2 different network simulators (this cost would be minimum when the configuration is scaled). We believe that equally good results would be achieved if the network is scaled in terms of the number of domains, number of flows

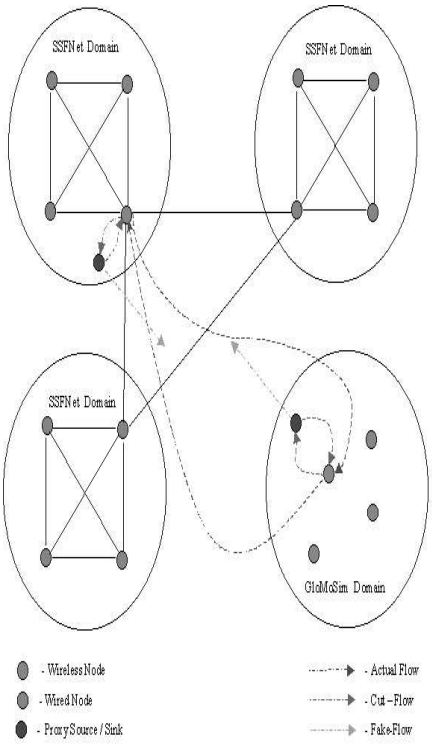

Figure 14: Network Topology for Interoperability of GloMoSim with SSFNet

Network Type	SSFNet	GloMoSim	SSFNet+GloMoSim
Avg. Time(s)	8.43	10.52	12.38

Table 3: Measurements results for interoperability on IBM Netfinities (times are in seconds)

and number of nodes per domain. The time of simulation execution is in direct relation to the latter three parameters.

5 Conclusion and Future Work

In this paper we present a novel approach to large scale wireless network simulation, which combines simulations of distributed domains and models it as a single system. The model is run until it converges to the fixed point solutions so each domain produces the required outputs based on the received inputs. Each model is fed by the data produced by the simulation (of the neighboring domains) and sends its output to other simulations (domains). This approach extends the Genesis technique to wireless simulations using peer-to-peer connections between neighboring domains. Using this approach, we observe a super-

linear speed up of a wireless network simulation on distributed computer architecture.

Some of the directions to improve/extend the current implementation are as follows:

- currently, this interface supports only for CSMA-CD protocol. We would like to extend it to 802.11-based MAC protocols,
- extension of the interface to support mobility types other than "mobility-trace",
- conducting extensive tests for interoperability between wireless domains and wired domains on the lines of the ones conducted by the GloMoSim group at UCLA,
- optimization of the peer-to-peer protocol used for communication between neighboring domains. This would help improve the speed-up of the simulation.

6 User manual

6.1 Overview of changes made to GloMoSim

The following section gives the syntax of the commands added to GloMoSim and also describes the high-level algorithms used to implement the above described components.

6.1.1 Freeze Scheduler

Freezes are used to pause the simulation at the points of time which are assigned by users in the GloMoSim network configuration file config.in. During the freezed time, the method glomofreeze() in file glomofreeze.pc is called. The method glomofreeze() is the right place for the GloMoSim users to perform their own functions. When return from the glomofreeze(), the simulation will resume. In the Genesis interface, we perform/ call all the other components from this method. Thus, this method is the core of all the changes made by us.

The syntax of the freeze command in the config.in file is as follows:

```
FREEZE-TIME     [freeze_time]S
FREEZE-INTERVAL [interval]S
NO-OF-FREEZES [no_of_freezes]
```

The algorithm for the freeze scheduler is described as follows:

```
static int i;
        if(i $< no_of_freezes)
        {
          print("Simulation frozen'');
          //...call all other components
          i++;
        }
```

6.1.2 Domain Deifnition

We divide the network into domains as described above. This divison of the network configuration into the respective domain components is a function of the geographical area the domain covers. Only one domain is active on each processor. In our extension to GloMoSim, we specify all the domains in a file called "domain.input". The active domain is specified in the config.in file. This file and the active domain information is parsed along with the other network configuration parameters in "glomo.pc". For each active domain, we maintain a list of which nodes is active in it. This list is created duting initialization by comparing the location of the node with the domain boundaries.

The syntax for the domain definition file domain.input is as follows:

```
[domain_id] [start_x] [start_y] [end_x] [end_y]
```

The syntax for specifying the active domain is as follows:

```
NO-OF-DOMAINS   [no_of_domains]
DOMAIN-PLACEMENT-FILE ./domain.input

ACTIVE-DOMAIN   [id]
ACTIVE-DOMAIN-START-X   [start_x]
ACTIVE-DOMAIN-START-Y   [start_y]
ACTIVE-DOMAIN-END-X     [end_x]
ACTIVE-DOMAIN-END-Y     [end_y]
```

Please note that the start and end x and y values in the domain.input and the config.in files respectively must match.

The algorithm for domain definition is as follows:

```
parse config.in to determine the number of domains
parse domain.input to determine the domain boundaries
parse config.in for active domain information
for each node in the network do
{
compare node location with domain boundaries
if node falls within boundary, mark it acive
}
```

6.1.3 Mobility of nodes

As described above, in the Genesis extension to GloMoSim, we concentrate on "mobility-trace" as it allows us to control the mobility of the nodes. In this case, the mobility of nodes is be specified in the configuration file "mobility.in". We take care of a total of 6 possibilities when a mobile node sends or receives a message from the nodes in the network - 3 for when a node sends a message and move before an event is scheduled or during the event or after the event is scheduled to that node. Similarly there are 3 possibilities for when a node receives messages - a node receives a message and move before an event is scheduled for it or during the event or after the event is scheduled for it. However, instead of taking care of these "mobile" nodes at run-time, we can modify the scheduler during pre-simulation. Although such precomputing can b

In the file mobility.in, the user specifies the speed, start and destination locations of the nodes in a configuration file. Knowing the above parameters, we can calculate the time and location (x, y) when the node crosses the domain boundary (using the equation of the straight line - we assume that nodes always travel along a straight line). Using this information we modify the behavior of the scheduler to allow nodes to transmit data only when it is in the active domain. This gives rise to three cases:
a. The source and destination positions of the node both don't belong to active domain. In this case we calculate the times when the node enters and leaves the active domain. Based on these times, we schedule the time when the node can transmit packets.
b. The source position is in active domain while the destination location is not. In this case, we need to calculate the time when the node leaves of the active domain.
c. The source location is not in the active domain while the destination position is in the active domain. Thus, in this case we need to calculate the time when the node enters the active domain.

All of the above is done in a method GLOMO_MobilityCalculateTime() in the file "mobility_trace.pc" for each mobile node. The crux of this method is as follows:

```
GLOMO_MobilityCalculateTime()
{
  -parse information of mobile node(mobility.in)
  -check for the 3 cases (a,b,c) above based on
   the source and destination positon of the
   node
  -if any of the 3 cases are satisfied, based
   on the source and destination location and
   speed of node, calculate the slope of the
   straight line traveresed by the node. This
   gives the equation of the line traversed
   by the node.
  -calculate the time(s) when the node
   crosses the boundary.
  -use this time to reschedule the
   transmission/receiving time for that
   node
}
```

6.1.4 Conflict detection and resolution

In the Genesis extension to GloMoSim, we take care of the 2 cases for conflicts described above. This is done in the Glomo_DetectConflict() in glomofreeze.pc. This function is called once the exchange of the log information between atmost 8 neighboring domains takes place. This function is called for each log of information received. The algorithm is as follows:

```
GLOMO_DetectConflict()
{
  - for each neighboring domain do
    {
     -receive the log of information for the
      last iteration.
     -compare the node_id of the received log
      with each node_id of the domains own
      log of nodes in the closure
     -if the node_id's match, compare the start/
      end times of transmission for the 2 nodes.
     -if these times do not match there is no
      conflict and go on to next log of data.
     -else check for case 1.1, 1.2, 2.1 & 2.2
      described above (basically here the mode
      of the node - transmitting/receiving - is
      checked for a conflict in the respective
      domain)
     if a match is found
        {
         -set the go-back flag to 1
         -resolve the conflict by toggling the
          mode of conflicting node in the
          schedule of events( so that in the
          goback this node behaves correctly)
      }

    }
}
```

6.1.5 Genesis data exchange model for distributed wireless simulations

We use a peer-to-peer data-exchange model and also a farmer-worker architecture in the Genesis extension to GloMoSim.

Peer-to-Peer data exchange

As described above, each domain has a maximum of eight neighbors. Thus, at each freeze, each domain will communicate to at most 8 neighbors. These changes can be found in communication.c and glomofreeze.pc. The algorithm for peer-to-peer data exchange is as follows:

```
Each domain does the following at the beginning
of simulation
{
establish connections via sockets to each of its
  neighboring domains.
}
At each freeze, each domain does the following
{
   - send up to 8 messages to its neighboring
   domains (as each domain has at most 8
   neighbors).
   - receive logs from all of its neighbors
     and compare the received data with the
     corresponding logged data.
   - make a decision to go-back/go-ahead
   - if domain needs to go-back, it does so
   - if it can go-ahead, it sends a sync to
     the farmer indicating the same after which
     it waits for signal from farmer to go-ahead.
}
At end of simulation
{
  close all socket connections and exit
}
```

Farmer-Worker Architecture In order for workers to prevent cascading conflicts (described above) and also to synchronize between freezes - that is all workers to be in the same iteration - we implement a server (farmer). This implementation can be found in communication_farmer.c. This implementation is similar to the Genesis extensions for UDP and TCP on NS and SSFNet (except the need for broadcasting a recheck signal used to prevent cascading conflicts).

The algorithm for the farmer-worker architecture is af follows:

```
The farmer does the following till the end
of simulation:
 -Wait for a go-ahead/go-back signal from
  all workers(domains)
 -If any domain needs to go-back , the farmer
  sends a recheck broadcast to all the domains
  along with the domain id's of the domains
  that need to go-back.
 -Else the farmer busy waits until all workers
  can go-ahead.
 -Only when  all workers can go-ahead, the
  farmer broadcasts a sync message to all
  workers to simulate the next interval.
```

6.1.6 Checkpointing

We use diskless checkpointing as described above in order to go-back and resimulate a freeze interval. This code can be found in glomofreeze.pc. The algorithm (implementation) for checkpointing is exactly the same as that the Genesis extension to NS and SSFNet. The documentation for the same can be found elsewhere.

6.2 Location of source code and documentation

The additions and modifications to GloMoSim for GENESIS have been done on FreeBSD IBM Netfinity machines. This section gives the paths for the source code and the documentation. All the source code and documentation is under madnak/projects under various sub-directories specified below.

Paths on FreeBSD:

Source code:

- /glomosim-2.0/glomosim,
- /glomosim-with-interoperability

Scripts

- Distributed Simulation Experiments: madnak/projects/glomosim-2.0/glomosim/tests

Configuration	Subdirectory
900 nodes, 1 domain	1-domain
900 nodes, 4 domains	4-domain
900 nodes, 16 domains	16-domain

Table 4: Scripts and their directories

Below every subdirectory for the configuration there are scripts for starting the configuration of the respective distributed simulation, for example, under tests/4-domain, there is 4-domain.sh, which starts the simulation of the 4 domains. The scripts to

start the 4-domain server and 16-domain server is present in 4-domain/4proc_server.sh and 16-domain/16proc_server.sh.

- Experiments with Interoperability:
  /glomosim-with-interoperability
  The test scripts for interoperability are present under
  /glomosim-with-interoperability/glomosim/bin.

- Documentation:
  /documentation/ms-report.tex,
  /documentation/ms-report.ps,
  /documentation/ms-report.pdf

References

[1] Szymanski B., Gu Q., Liu Y, "Time-Network Partitioning for Large-Scale Parallel Network Simulation under SSFNet," to appear in *Proc. Applied Telecommunication Symposium*, SCS Press (to be presented at ATS2002, April 2002).

[2] Carothers C., Szymanski B., "Linux Support for Transparent Checkpointing of Multithreaded Programs," to appear in *Dr. Dobbs Journal*, August 2002.

[3] M. Gerla and J.T.-C.Tsai, "Multicluster, mobile, multimedia radio network," *ACM/Baltzer Journal of Wireless Networks*, vol. 1, (no 3) 1995, p.244-265.

[4] 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.

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

[7] 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.

[8] Szymanski, B., A. Saifee, A. Sastry, Y. Liu and K. Madnani, "Genesis: A System for Large-scale Parallel Network Simulation," *Workshop pf Parallel Network Simulation*, Washington D.C., May 2002.

[9] Xiang Zeng, Rajive Bagrodia, Mario Gerla, "GloMoSim: a Library for Parallel Simulation of Large-scale Wireless Networks," *Proceedings of the 12th Workshop on Parallel and Distributed Simulations*, May 26-29, 1998, Banff, Alberta, Canada.

[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.

[11] GloMoSim website: "http://pcl.cs.ucla.edu/projects/glomosim/"