Appendix A

GloMoSim SIMULATOR FOR WIRELESS NETWORKS

- **PARSEC**

PARSEC (for Parallel Simulation Environment for Complex systems) is a C-based simulation language developed by the Parallel Computing Laboratory at UCLA, for sequential and parallel execution of discrete-event simulation models. It can also be sued as a parallel programming language. PARSEC runs on several platforms, including most recent UNIX variants as well as Windows.

PARSEC adopts the process interaction approach to discrete-event simulation. An object (also referred to as a physical process) or set of objects in the physical system is represented by a logical process. Interactions among physical processes (events) are modeled by time-stamped message exchanges among the corresponding logical processes. One of the important distinguishing features of PARSEC is its ability to execute a discrete-event simulation model using several different asynchronous parallel simulation protocols on a variety of parallel architectures. PARSEC is designed to cleanly separate the description of a simulation model from the underlying simulation protocol, sequential or parallel, used to execute it. Thus, with few modifications, a PARSEC program

may be executed using the traditional sequential (Global Event List) simulation protocol or one of many parallel optimistic or conservative protocols. In addition, PARSEC provides powerful message receiving constructs that result in shorter and more natural simulation programs.

- **GloMoSim Library**

GloMoSim is a scalable simulation library for wireless network systems built using the PARSEC simulation environment . Table 1 lists the GloMoSim models currently available at each of the major layers. GloMoSim also supports two different node mobility models. Nodes can move according to a model that is

generally referred to as the “random waypoint” model . A node chooses a random destination within the simulated terrain and moves to that location based on the speed specified in the configuration file. After reaching its destination, the node

pauses for a duration that is also specified in the configuration file. The other mobility model in GloMoSim is referred to as the “random drunken” model. A node periodically moves to a position chosen randomly from its immediate neighboring positions. The frequency of the change in node position is based on a parameter specified in the configuration file.

Layer:	Models:
Physical (Radio propagation)	Free space, Rayleigh, Ricean, SIRCIM
Data Link (MAC)	CSMA, MACA, MACAW, FAMA, 802.11
Network (Routing)	Flooding, Bellman-Ford, OSPF, DSR, WRP
Transport	TCP, UDP
Application	Telnet, FTP

Table 1: Models currently in the GloMoSim library.

In contrast to existing network simulators such as OPNET and NS, GloMoSim has been designed and built with the primary goal of simulating very large network models that can scale upto a million nodes using parallel simulation to significantly reduce execution times of the simulation model. In the next sub-sections, we explore the techniques of node and layer aggregation that are used to achieve this scalability.

As most network systems adopt a layered architecture, GloMoSim is being designed using a layered approach similar to the OSI seven layer network architecture. Simple APIs are defined between different simulation layers.

This allows the rapid integration of models developed at different layers by different people. Actual operational

code can also be easily integrated into GloMoSim with this layered design, which is ideal for a simulation model as

it has already been validated in real life and no abstraction is introduced. For example, a TCP model was implemented in GloMoSim by extracting actual code from the FreeBSD operating system. This also reduces the amount of coding required to develop the model. The simple APIs that are currently implemented in GloMoSim are also presented in a following sub-section.

- **Node Aggregation**

In PARSEC, a simple approach to designing a network simulation model is to create each network node as an entity. Although this approach is easy to understand, it has scalability problems. If an entity has to be instantiated

for each node, the memory requirements would increase dramatically for a model with large number of nodes because each entity requires additional memory to work as an independent process. The performance of the simulation would also degrade due to context switching overheads among many entities. Hence, initializing each node as a separate entity inherently limits the scalability and performance of the simulation.

To circumvent these problems, node aggregation was introduced into GloMoSim. With node aggregation, a single entity can simulate several network nodes in the system. A separate data structure representing the complete state of each node is maintained within the entity. When the simulation code for a particular node is being executed

it does not have access to the data structures of other nodes in the simulation. The node aggregation technique implies that the number of nodes in the system can be increased while maintaining the same number of entities in

the simulation. In fact, the only requirement is that we need only as many entities as the number of processors on which the simulation is being run. Hence, a sequential simulation needs only one entity in the simulation. With the node aggregation technique, the memory and context switching problems are eliminated. In GloMoSim, each entity represents a geographical area of the simulation. Hence, the network nodes that a

particular entity represents are determined by the physical position of the nodes. For example, suppose we specify a geographical area of 100 by 100 meters in the simulation and set the number of x and y partitions to be 2 for a

particular simulation. There would be four partitions in the simulation, where each partition is represented by a single entity. Fig. 2 shows how the terrain would be divided into the four partitions. One particular partition in the simulation would encompass the area represented by the coordinates (0, 0), (49, 0), (0, 49), and (49, 49).

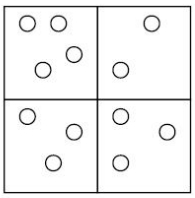

Fig. 2: A simulation with geographical area of 100 by 100 meters, which is divided into four partitions.

In a basic usage of GloMoSim, each entity represents a regular rectangular region (partition). Thus, a partition can have at most eight neighboring partitions. When a network node sends out a message, the message has to be sent to at most the eight neighboring entities in the simulation. This is much simpler than the simple design mentioned previously. If each entity represents a single network node, broadcasting a message from a node is very difficult. The first option to implementing broadcasting is that each entity constantly keeps track of the other entities that are within its power range. This option is difficult since the network topology would constantly change as mobility is introduced into the simulation. The second option is that when a node sends a message, it would be sent to all the other entities in the simulation. The receiving entity will accept the message as long as it is in the power range of the sender. This becomes highly inefficient as the number of nodes in the simulation increases.

Hence a simple message transmission becomes very complicated when node aggregation is not used. With node aggregation, each entity can examine which node can receive a packet within the entity and send messages only to the neighboring entities where the packet can be reachable.

- **Layer Aggregation**

Since GloMoSim is being built using a layered approach, the ability to rapidly integrate models developed at different layers by different people is very important. Hence, the simple approach would be that each layer in the simulation would be represented by a different Parsec entity. Some of the problems of using an entity to represent a single node reappear. As the number of layers in the simulation increases, the number of entities in the simulation also increases. This leads to scalability and performance problems in the simulation. This is not as dramatic as the case where an entity represents a single node since there are relatively few layers in the simulation. Aggregating all the layers into a single entity is still a compelling argument for other reasons.

There are times in a simulation when different layers need to access certain common variables. For example, the upper layers of the simulation need to use the CPU when they are executing any instructions. Since CPU is a shared resource among these layers, a layer has to make sure that the CPU is free before executing any instructions.

Hence the upper layers need to have access to common variables which will provide information about the state of the CPU. If these layers are kept as different entities in the simulation there is no elegant method for accessing shared variables. Global variables cannot be used in such a situation as they cause problems with concurrent access during parallel executions.

If the layers are kept as different entities, each layer also has to explicitly keep track of the entity name values for the upper and lower layers. These entity name values are needed for message passing among the various layers.
For the parallel conservative runtime, each entity also needs to specify the source and destination set of communicating entities as well as lookahead values. Specifying lookahead for an entity can be a very cumbersome and difficult task. This creates additional work for the protocol developer who is basically interested in modeling aparticular network protocol.
For these reasons, we decided to integrate the various GloMoSim layers into a single entity. Each entity encompasses all the layers of a simulation. Each layer is now implemented as three function calls by the protocol modeler. The developer has to provide an initialization function that will be called for each layer of each node at the
beginning of the simulation. The next function call provided by the developer is automatically invoked when a particular layer of a particular node receives an incoming packet/event. Based on the contents of the message, the appropriate instructions will be executed. Function calls are also provided for a layer to send messages to its lower or upper layer in the simulation. At the end of the simulation, another developer provided function call is invoked.
This can be used to collect any relevant statistics for that layer.
It might appear that the addition of node and layer aggregation would cause great difficulty for protocol developers who are only interested in developing the simulation model for their particular model. But this is not the case as we have created several layers of abstractions in GloMoSim. For the most part, the developer writes pure C code. The presence of the PARSEC runtime and interactions with the runtime are completely hidden from the user.
In the experience of our own group, modelers prefer to work within our structured environment of node aggregation rather than the alternative.

4.3 Simple APIs

Simple APIs between every two neighboring models on protocol stacks is predefined to support their composition. These APIs specify parameter exchanges and services between neighboring layers. The simplicity of the APIs allows developers to model their protocols rapidly in an independent fashion. The APIs currently defined

in GloMoSim are presented:

Channel Layer – Radio Layer APIs:

Data packet from Channel to Radio:

Fields: payload, packetSize

These fields refer to the actual data and size of data being received. They have similar meanings when used

subsequently for the reception or transmission of packets.

Data packet from Radio to Channel:

Fields: payload, packetSize

Radio Layer – MAC Layer APIs:

Data packet from Radio to MAC:

Fields: payload, packetSize

Data packet from MAC to Radio:

Fields: payload, packetSize

Request Channel Status from MAC to Radio:

Fields: (none)

This message is used by the MAC layer to request information about the current channel status.

Report Channel Status from Radio to MAC:

Fields: status, flag

This message is used by the radio layer to return the current status of the channel as well as the method by

which the information is being reported (passively or actively based on the request message sent by the
MAC layer).

MAC Layer – Network Layer APIs:

Data packet from MAC to Network:

Fields: payload, packetSize, sourceId

The sourceId refers to the previous hop from which the packet arrived.

Data packet from Network to MAC:

Fields: payload, packetSize, destId

The destId refers to the next hop where the packet will travel.

Network Layer – Transport Layer APIs:

Data packet from Transport to Network:

Fields: payload, packetSize

The IP header should be a part of the packet that is sent from the transport to the network layer.

Data packet from Network to Transport:

Fields: payload, packetSize, sourceId

The sourceId refers to the original source where the packet originated. For the packet sent from the network
to the transport layer, the IP header is no longer a part of the packet.

Network Layer – Application Layer APIs:

Data packet from Network to Application:

Fields: payload, packetSize, sourceId

Data packet from Application to Network:

Fields: payload, packetSize

These APIs, which are similar to the APIs used between the network and transport layers, are used for
communication between routing daemons (such as OSPF) that are running at the application layer and need

to communicate directly with the network layer.

UDP Transport Layer – Application Layer APIs:

Data packet from UDP to Application:

Fields: payload, packetSize, sourceAddr, sourcePort, destAddr, destPort

Data packet from Application to UDP:

Fields: payload, packetSize, sourceAddr, sourcePort, destAddr, destPort

In these APIs, the sourceAddr and sourcePort refer to the source address and port number where the packet
originates. The destAddr and destPort refer to the destination address and port number where the packet is
going.

TCP Transport Layer – Application Layer APIs:

Open Listen Socket from Application to TCP:

Fields: appType, localPort

This API is used by an application type (such as telnet server) to open a listen connection on the given port
number.

Connection Open from Application to TCP:

Fields: appType, localPort, remoteAddr, remotePort

This API is used by an application to inform TCP to try to setup a connection from the given local port
number to the given remote address and port number.

Data packet to send from Application to TCP:

Fields: payload, packetSize, connectionId

This API is used by an application to send a packet using on the given connectionId.

Connection Close from Application to TCP:

Fields: connectionId

This API is used by an application to close a particular connection.

Listen Socket Open Result from TCP to Application:

Fields: localPort, connectionId

This API is used by TCP to inform the application about the result of trying to open a listen connection.

Connection Open Result from TCP to Application:

Fields: type, localPort, remoteAddr, remotePort, connectionId

This API is used by TCP to inform the application about any connection that has been opened to a remote
address and port number and the associated connection id. The connection type can be passive or active.

Data Sent Result from TCP to Application:

Fields: connectionId, packetSize

This API is used by TCP to inform the application about the number of bytes that could be sent due to the
data sent request generated by the application for TCP.

Data Received from TCP to Application:

Fields: connectionId, payload, packetSize

This API is used by TCP to inform the application about any data that has been received on a connection.

Connection Close Result from TCP to Application:

Fields: type, connectionId

This API is used by TCP to inform the application about the connection getting closed as well as the type
of connection close (passive or active).

5. Scalability of GloMoSim

The node aggregation technique gives significant benefits to the simulation performance. As each entity needs
to examine packet receptions only for the nodes located in the region it is simulating, using many partitions reduce

the total search space for packet delivery. In Fig. 2, for instance, if a packet sent by a node located in Partition (0, 0)
cannot reach the border of the partition, no message needs to be sent to the other partitions. Therefore, the other
partitions do not have to examine the reception of the packet, which reduce the region to be examined for the packet
by a factor of four compared to using single partition. Fig. 3 shows the impact of multiple partitions for the models
with 2500 and 5000 wireless nodes. Both simulation models consist of wireless nodes running CSMA at the MAC
layer, each of which is randomly placed in 2000 x 2000m free space region. As seen in Fig. 3, the executions for
both models become faster as the number of partitions increases. The effect of multiple partitions is larger for the
model with 5000 nodes as the reduction in the execution time is related to the number of wireless nodes to be
examined for each radio transmission.

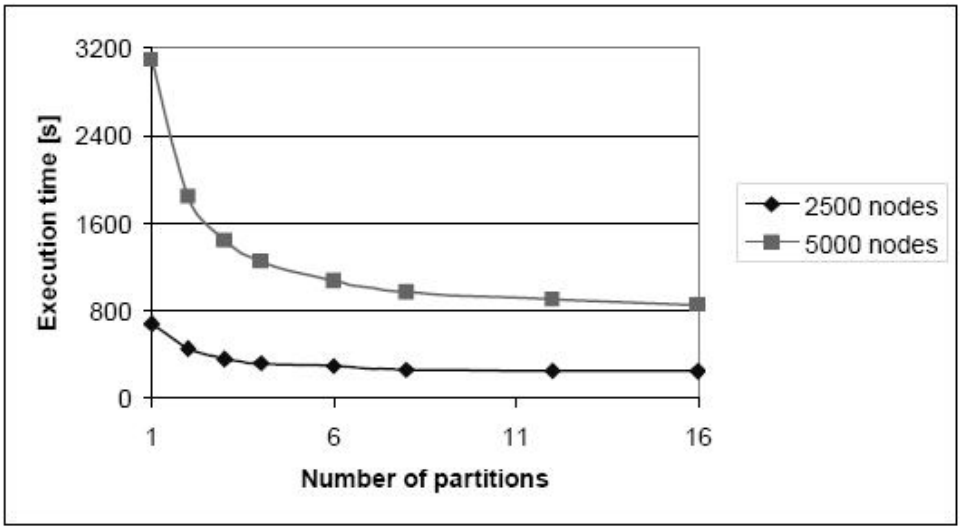

Fig. 3: Execution times against the number of partitions.

GloMoSim is aimed at simulating models that may contain as many as 100,000 mobile nodes with a
reasonable execution time. GloMoSim has already been used to simulate 10,000 nodes up to the MAC layer using parallel execution of the model on shared memory architectures. Fig. 4 indicates parallel performance of GloMoSim on a Sun SPARCserver 1000. Speedup rates are calculated based on the sequential execution of each model. The same configuration as the experiments on multiple partitions is used with different number of wireless nodes. Twelve partitions are used for all the executions to balance the workload of each processor. As shown in Fig. 4, GloMoSim achieved better parallel performance for models with higher number of wireless nodes because more activities occur concurrently in those models, which increase the parallelism of models.

Users, especially those who need to simulate large-scale models can benefit from this parallel simulation capability of GloMoSim. Fig. 5 shows increases of execution times against the number of mobile nodes in the model. With the same number of processors, the execution time increases dramatically as the number of mobile nodes in the model increases. However, the execution time with 6 processors for the 10000 node model is shorter than the sequential execution for the 5000 node model. This implies that the user can run the simulation for a model consisting of twice the number of mobile node in the same amount of time with 6 processors.

Parallel simulation requires synchronization of simulation clock among multiple processors. PARSEC simulation environment provides four variations of conservative protocols and an optimistic protocol for the synchronization. The current GloMoSim kernel has parallel execution directives for conservative protocols and will be capable of executing models using optimistic protocols in future. The experiments done in this section used the null message protocol, which is one of the most basic conservative protocols widely used for many applications.