AN ARCHITECTURE FOR DISTRIBUTED SIMULATION OF WIRELESS NETWORKS

Ratan K. Guha and Oleg Kachirski
School of Computer Science
University of Central Florida
Orlando, FL 32816, USA
E-mail: {guha, oleg}@cs.ucf.edu

ABSTRACT

In this paper we describe a simulation framework designed for simulating wireless network technologies. We provide a detailed architecture of the framework and illustrate all the necessary steps to develop simulations for any particular wireless networking application. We demonstrate the use of this framework by simulating network intrusion detection system for wireless ad hoc networks. We also extend the framework architecture to multiprocessor environments and describe the optimizations to further improve the performance of the simulation system for large-scale simulations of wireless networks.

INTRODUCTION

Wireless networks today are used in almost every organization. Researchers enthusiastically work on developing new communi-cation protocols, algorithms and methods. Many papers in the network research community propose new wireless algorithms, offer better performance, quality of service and other benefits. Technology of choice should be carefully evaluated before being adopted. Primary method of evaluating wireless networks is to create a computer simulation of a particular topology, communication mechanism or an algorithm. Our primary objective was to create a distributed simulation framework capable of running on a cluster computer, targeting wireless networks. The key requirements were flexibility, extendibility and scalability, as our final goal was to utilize a simulation system capable of being extended to very large simulations. Several well-known general purpose network simulators were considered (Heybey 1988; Groselj 1995; Mah 1998; Keshav 1997; VINT Project 1997; Chang 1999) to create our simulation model of a large-scale wireless network. After reviewing documentation and running sample simulations, we have come to the conclusion that none of the assessed simulators would meet our requirements due to several reasons, such as inflexible modeling of wireless nodes to include desired functionality (i.e., mobility models for ad hoc networks), and relying on C libraries from a particular architecture; thus making it non-portable between different computer architectures

WINDS Framework (Wireless Network Distributed Simulation Framework) was developed by our Networks Research Group at the University of Central Florida as a research tool to provide time-saving flexible simulation test-bed targeting a wide range of wireless network architectures. The framework is cross-platform, Java-based, GUI-driven and can be used as a generic wireless network simulator for a variety of purposes, such as routing in ad hoc networks, mobility models of totally mobile wireless networks, etc.

The rest of the paper is organized as follows. First, we present the WINDS architecture design in the next section. Framework modules are described in detail in this section, as well as a generic simulation process. We also list requirements and limitations of our simulation framework. In section 3, a practical example of WINDS framework application is given as a test-bed for our Agent-based Intrusion Detection System for Ad Hoc Wireless Networks (Kachirski and Guha 2002; Kachirski and Guha 2003; Guha et al. 2002). Network clustering algorithm and agent allocation algorithm are presented, along with result discussion. Section 4 discusses on-going work on extending the framework to multiprocessor environments and distributing simulation objects efficiently to minimize inter-processor communications. Finally, a summary and conclusions are presented, along with plans for future development and parallelization of WINDS framework.

WINDS DESIGN PHILOSOPHY

There was a direct need for us to develop a network simulator that specifically targets wireless networks of diverse configurations. Our goals were:

- Clean, easy-to-understand and modify design
- Object-oriented approach, general portability
- Use of a popular programming language
- Easy-to-use, GUI driven framework
- Adequate performance and scalability

Following these guidelines, we have developed WINDS framework – a flexible, portable, wireless-network oriented framework that can be used to simulate variety of applications of wireless and hybrid networks. The objectives of this project were to reduce redundant software design efforts in the area of wireless network simulation, establish a framework general enough to be used for simulations of many wireless network-related technologies, and provide for common base for the

Proceedings 18th European Simulation Multiconference
Graham Horton (c) SCS Europe, 2004
ISBN 3-936150-35-4 (book) / ISBN 3-936150-36-2 (CD)

experimentation of various wireless infrastructures. The object-oriented nature of the software and the use of a popular programming language for implementation allow researchers to easily modify, reuse and share whole systems or system components. Graphical environment allows the system to be used for demonstrational or educational purposes.

WINDS Architecture

The architecture of WINDS (simulator and interface) is based on a building-block approach. Researcher implements an algorithm or a prototype from modules that receive inputs in the form of events, process them, and then generate outputs (events, log entries, GUI updates). The entire wireless network is built from objects – wireless nodes, event generator, and communication channels. Connections between wireless nodes are maintained by the routing object. Any type of wireless network is supported, such as ad hoc and infrastructure wireless networks. Object-oriented approach is central to the generality and flexibility of the system and allows users of our framework to reuse, share and catalog simulation components by modifying or replacing appropriate classes.

Many existing wireless network simulators (Boukerche et al. 2001; Kelly et al. 1998; Kelly et al. 2000; Liu et al. 1996) aim at supporting every aspect of wireless communications, such as, for example, every layer of many communication protocols used. This adds tremendous overhead to the simulation system, often resulting in scalability problems and slow execution times. The WINDS framework avoids these problems, by implementing only the key functionality of a wireless network – supporting simulation synchronization, wireless communications and node mobility. These functions are implemented on an abstract level, allowing the user to include specific wireless communication protocols, wireless routing algorithms and other required simulation parameters, as necessary. Another goal of our framework development was to integrate user interaction with simulation execution as closely as possible. The intuitive graphical user interface reflects any changes in the simulation execution as they happen, in real-time, allowing the user to adjust the simulation parameters at run-time. Another advantage of our framework is the ability to use almost any data file as a source of wireless communications. Data file pre-processor converts binary packet data into the format accepted by WINDS simulation system, converting network addresses into plain addressing scheme used with our simulation framework. As an example, intrusion data from the Lincoln Laboratories IDS tests was used to test our wireless IDS system.

WINDS architecture consists of four key modules, each comprised of a number of components, as described below:

- GUI (graphical user interface)
- Simulator Core Module
  - Simulation engine, Simulation objects
- Network Traffic Module
  - Packet pre-processor, Packet generator
- Data Logging Module
  - Data parser, CSV file generator

Figure 1 shows the WINDS architecture. Some of the modules carry optional functionality and can be included into the simulation as necessary. The functionality of each module is described below.

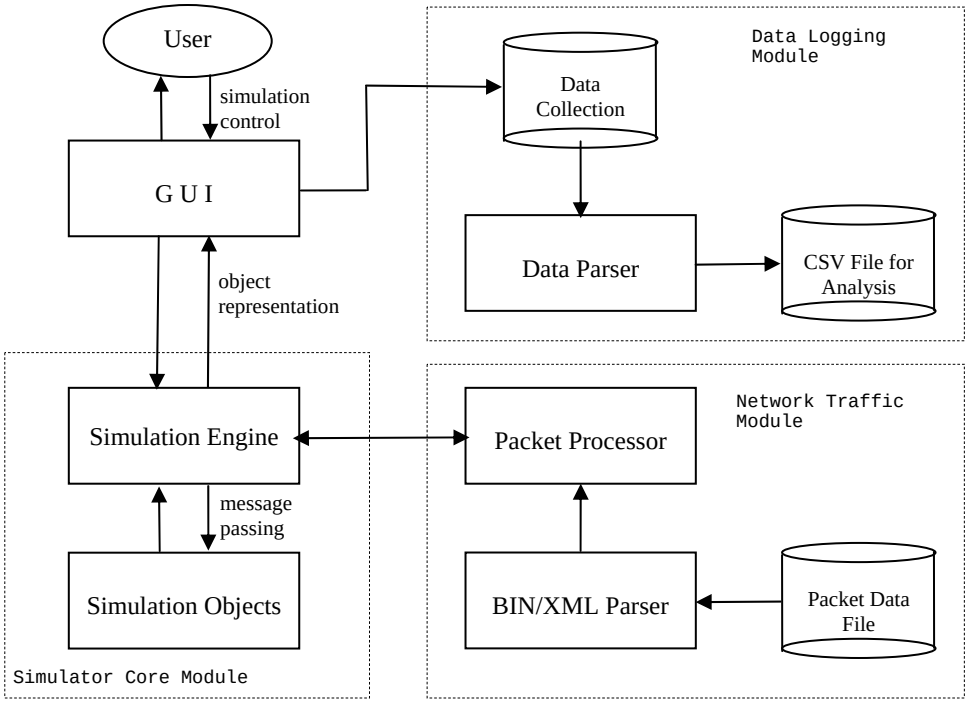

Figure 1: WINDS Architecture

Graphical User Interface Module

The graphical user interface (GUI) module shows the simulation execution in real-time. The GUI class is tied up to the simulation engine clock, and displays the required information every clock cycle. GUI class shows the simulation area with wireless node objects moving and communicating. Certain simulation parameters can be adjusted via GUI module. GUI module also provides controls for simulation execution flow. GUI module also records simulation statistics in a log window.

Simulator Core Module

Simulation Engine: the heart of the WINDS framework. One of the design requirements is that WINDS is a time-stepped simulation system. The simulation engine runs an internal clock, the speed cycle of which can be controlled at run-time. Programmatically implemented as a high-priority thread, the simulation engine runs in a loop continuously, driving execution of all other components of the simulation. Any events happening at a time must be processed by simulation objects at once, within a single simulation cycle. The simulation engine is common to all simulation models, and cannot be modified by the user. Simulation engine instantiates all simulation user objects from global definitions (such as wireless nodes, routing algorithm used, mobile base stations and stationary routers) at runtime.

Simulation Objects: execute independently and are time-synchronized via the simulation engine clock. Our framework has a few pre-defined simulation objects. One object type is a wireless node object. Wireless nodes are members of any wireless network simulation, and can be either stationary or mobile. User can modify the motion pattern by implementing a certain mobility algorithm (or even read waypoints from a data file). This allows simulations to be flexible and account for many possible node mobility patterns. Each wireless node object also includes two methods used for inter-object communication – Send and Receive methods. Send method is invoked when a wireless node is transmitting packets, and has a source, destination, protocol, port and payload as its arguments. Send method determines all the neighbors of the current wireless node, and broadcasts the packet to its neighbors by invoking Receive method on each neighbor node. Receive method first checks the packet destination, then depending on the routing algorithm used, forwards the packet to the destination or simply drops the packet. This allows the user to simulate both ad hoc and infrastructure wireless networks.

Routing protocol is another simulation object. The routing protocol included with WINDS framework is a simple table-driven protocol, which is implemented as a separate routing class. To modify the routing protocol, user must include all necessary parameters in a wireless node object, then add their own routing protocol class. Since the WINDS framework operates in exact same way as a real-life wireless network, all existing wireless routing protocols are supported. For the simulation of totally-mobile or wireless infrastructure networks, routing protocol is modified to route all messages via base stations.

Network Traffic Module

Network traffic for WINDS framework is generated by the Network Packet Processor object. The implementation of the object is common to all the simulations and includes reading a pre-processed data file in XML format, and forwarding each packet to the appropriate wireless node (source). Pre-processing is performed on a binary data file obtained from network packet capturing software (such as TCPDUMP). WINDS uses a flat addressing scheme to reduce communication overhead, and therefore all network addresses are converted to compatible notation by the BIN/XML Parser module before running the simulation. Packet processor object generates packets at times specified by scaled timestamps of each packet processed. This speeds up simulation execution, limiting the simulation speed only by the hardware specifications and the maximum packet broadcast rate. Simulation can automatically be stopped when the end of the data file is reached.

Data Logging Module

Data logger class saves the simulation results for future analysis. During simulation execution, results are stored in memory and displayed in a human-readable form via GUI data display window for performance reasons (frequent disk I/O operations reflects negatively on simulation performance). The representation of results can be tailored to particulate simulation requirements and is defined in the GUI class.

Data parser: At the end of the simulation run, these results are first pre-processed by a data parser to format data suitable for import into the mathematical analysis software.
CSV file generator: The pre-processed output is saved as a CSV file (a widely-used comma-separated data file format) by the CSV file generator.

Simulation Process

The simulation procedure is as follows. First, a simulation diagram is devised by the user, which includes the list of all objects taking part in the simulation and interaction mechanisms between the objects. Packet and data output formats, as well as simulation stop conditions should also be specified at this point. Next, the implementation is written for all of the above objects, following the guidelines included with the simulation framework. Data and XML parsers should also be re-written to reflect the new data structures. The objects are then placed in the simulator core directory and compiled. The GUI is then started, which then instantiates all simulation objects and displays them on-screen (for those objects that have visual behavior defined). Objects then can be manipulated via graphical interface during simulation runtime. User can get a snapshot of the simulation

environment by pausing the simulator and stepping through simulation execution. If the network traffic module is selected, network packets are input from the packet data file, then converted to the appropriate format by the parser and fed into the simulation by the packet generator at specified times. Either actual time stamps may be used to pro-rate packet delay to simulation time scale, or a statistical distribution may be used, as specified in the packet generator class. Packets are then passed to appropriate objects via messages and processed. All collected statistics are displayed in a log window, which then can be saved into a simulation log file. The results of the simulation can be pre-processed for further analysis or graph plotting by the Data Parser module. The supplied data parser converts log file into a CSV file, which then can be imported into a variety of statistical programs.

WINDS FOR IDS SYSTEM DESIGN

Our first use of WINDS architecture was to develop a test-bed for a wireless intrusion detection system (IDS) (Lippmann et al. 1998; Haines et al. 2001) simulation prototype. Therefore, IDS is used as a tool to demonstrate an implementation of WINDS architecture. This section covers the architecture of our IDS system, step-by-step design process, and implementation of several IDS-related algorithms.

Wireless node object embodies mobile agent functionality for packet-monitoring and decision-making agents of an IDS system, as well as mobility pattern and clustering algorithm functionality. Send and Receive methods of a wireless node object are utilized to support voting scheme required by the cluster membership decision algorithm. These methods are also used by the network traffic generator module to propagate packets across wireless network. Packet-monitoring functionality is built-in the wireless node object for this simulation. Packet-monitoring agent processes the incoming network traffic, running it through the CASE-based intrusion detection mechanism, which bases its decisions on a library of XML rule sets, covering various communication protocols. Database lookup is performed for each network packet, classifying the packet as part of normal or intrusion traffic.

Our IDS system (Kachirski and Guha 2003; Guha et al. 2002) takes into account the specifics of wireless networks to provide a lightweight, low-overhead intrusion detection mechanism for wireless networks based on mobile security agent concept. Essentially, an agent is a small intelligent active object that travels across network to be executed on a certain host, then returns with results to the originator. All the decisions, including network traversing, are left to agents. Agents are dynamically updateable, lightweight, have task-specific functionality and can be viewed as components of a flexible and dynamically configurable IDS.

We have utilized mobile agents at several intrusion-monitoring levels and processed their response in cluster heads – special nodes that are dynamically elected within a cluster using a real-time distributed algorithm. One advantage of our approach is the efficient distribution of mobile agents with specific IDS tasks according to their functionality across a wireless ad hoc network. Another advantage is restriction of computation-intensive analysis of overall network security state to a few key nodes. These nodes are dynamically elected, and overall network security is not entirely dependent on any particular node, as in the case of a monolithic system. We have also proposed a load-balancing solution that efficiently distributes traffic monitoring and intrusion detection tasks among the wireless nodes, improving the accuracy of intrusion detection system without sacrificing the overall performance of a wireless network and functionality of each node participating in the network. At the network-monitoring level, we have developed a case-based approach to network intrusion detection, and incorporated case-based reasoning engine for detecting intrusions at the packet level in our modular IDS system. The IDS system implementation discussed here is targeted at ad hoc wireless networks.

WINDS FOR CLUSTER COMPUTER

The single-processor version of the simulation system suffices for small-scale simulations of wireless networks. However, we have had significant reduction in performance when simulations of 200 or more wireless nodes were in progress. Scalability is an important factor of every simulation system, as computer networks grow in size and become more complex in functionality. A distributed simulation is the answer to scalability problems. The idea of a distributed simulation is as follows. First, the entire scope of objects in the simulation is partitioned into several parts. For instance, we can divide 100 simulation objects equally between 5 processors. During simulation execution, each of the processors performs computations relevant to objects assigned to it. Communication between objects is handled by the communication broker – if two objects are handled by the same processor, communication happens in exact same way as in the uniprocessor system; if two (or more) objects are assigned to different processors, inter-processor communication takes place. Our distributed WINDS architecture is presented in figure 2. The distributed WINDS architecture is currently implemented on a cluster computer with 64 processors. Each processor has dedicated memory space, and can access data concurrently from a replicated disk subsystem. Inter-processor communication is achieved via high-speed Ethernet switch.

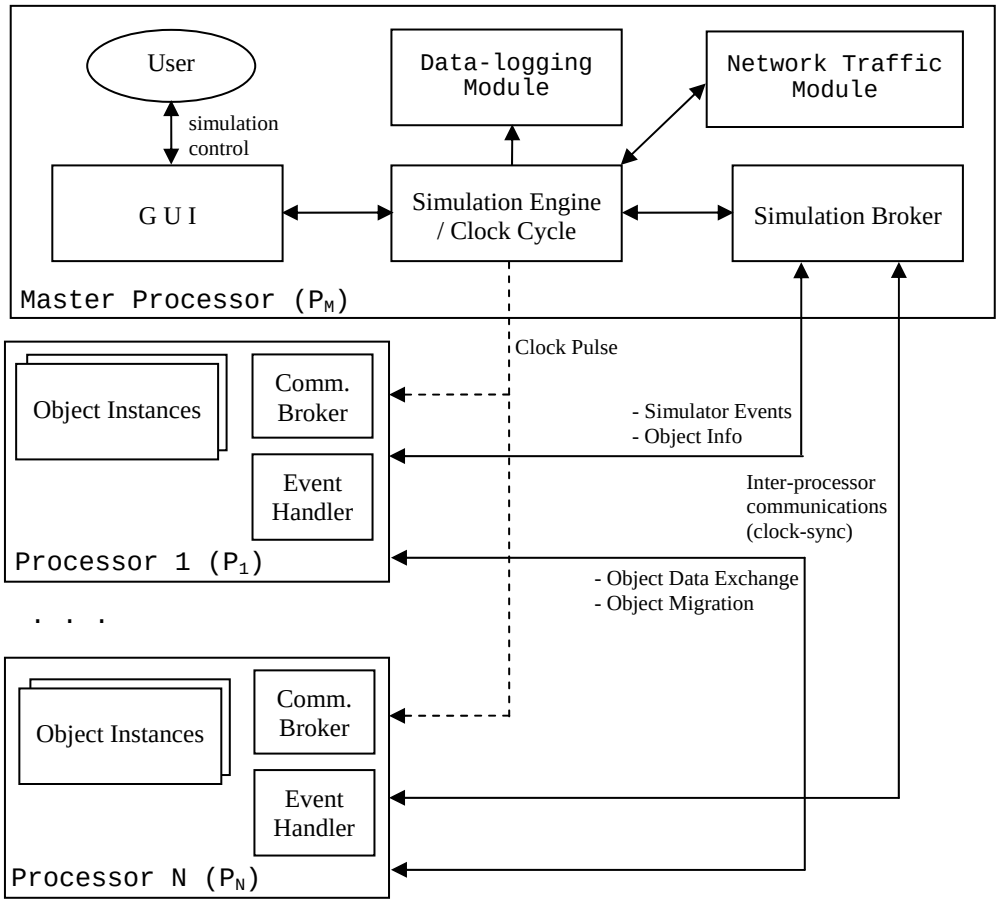

Figure 2: WINDS Architecture for Distributed Simulations in Multiprocessor Environments

Simulation Process

After object definitions have been devised and placed in a shared object directory, user starts the simulation framework on a master processor node. This in turn remotely starts simulation clients on each of the processors. Initially, user adds new objects to the simulation via graphical user interface. The objects are associated with processors in a round-robin manner. Instructions are sent to a respective processor from the master processor node to create an instance of an object and load it in memory. Object template is then read from disk by that processor, and a new object instance is created in its memory space. From now on, this object is handled by a local simulation module on that processor. Once all objects have been created, simulation run starts. Simulation engine sends a clock pulse out to every processor, and all communication between objects is clock-synchronized. Simulation broker located on the master processor keeps track of locating a specific object and serves as a routing module for the cluster communications. Other modules (like data logging module) behave in the same way as described for the single processor version of WINDS. When a packet needs to be sent from one object to another in the course of the simulation, communication broker on the node containing source object first determines if both objects reside on that node. If this is the case, then communication is performed locally by invoking the Receive method on destination object (same as for the uniprocessor case). If the destination object resides on a different processor, first a proper destination network address of that processor is determined by consulting simulation broker on a master processor node. Then, a network communication is initiated between the local processor, and the destination processor, handled by communication brokers of both processors. When a message is received on the destination processor, it is parsed for parameters, and Receive method of the destination object is called. Apart from exchanging messages between objects, all the processors also communicate with the master processor once every few clock cycles to ensure consistent state of the simulation and to report on the state of each object taking part in a simulation (i.e., to update the GUI information for each object). Commands are also sent from the master processor to each processor to control simulation execution.

Future Work - Simulation Optimizations

One of the main reasons against distributed implementation of many network simulators is inefficient inter-processor communication. In the case of a network of computers taking part in a simulation, network delay can be a significant obstacle to the goal of improving performance and scalability through the distributed simulation. Other traffic exists on the network, affecting simulator communications. Unless network nodes are dedicated for the simulation purpose,

node stability is an issue that can disrupt simulation entirely in the event of a single node crashing during the simulation run. Only specific computation-intensive algorithms can benefit from distributing the simulation among multiple processors. We have considered these and other issues when developing distributed WINDS framework. The optimal choice of computer hardware was computing cluster, where all processors communicate via high-speed switched network connections but own independent memory space and an instance of an operating system. This allows us to simulate very large wireless networks of diverse configurations.

Still, concerns exist for certain scenarios where inter-processor communication delay is of an issue. This can happen when one object repetitively communicated with objects located on a different processor, or in the event of a lot of broadcast communications taking place. Therefore, we have considered a number of optimizations that target the problems associated with the distributed simulation system. In one such optimization, during a certain period of time, all communication patterns are recorded, and allocation of objects is then optimized. For example, if an object A communicated with object B much more frequently than other objects, and these two objects are located on different processors, then one object is serialized and migrates via the network to the processor managing another object. In many cases, especially when object functionality is sparse, the size of the object is small, justifying such a migration. In another optimization, all communications between objects are concatenated together and sent as a single network packet between a pair of processors once every few clock cycles, and then locally time-synchronized. As we haven't completed implementing the entire range of planned optimizations, little can be said here regarding the actual performance figures.

SUMMARY

In this paper we have described the WINDS architecture for wireless network simulations. It has been developed to aid our research on the agent-based ad hoc network intrusion detection system, and later used as a research tool that incorporates a flexible test-bed targeting simulations of a wide variety of wireless networks. The framework is cross-platform, easy to learn, use and modify to adjust particular requirements. WINDS is considered a generic wireless network simulator for a variety of wireless communication applications, such as wireless networks, ad hoc networks, sensor networks, etc. We have demonstrated the use of WINDS for several specific simulations, such as intrusion detection system simulation in wireless ad hoc networks. We have also extended WINDS implementation to the multiprocessor environments (such as a cluster computer), improving scalability of large wireless simulations. The WINDS project is ongoing and will be available soon in its final implementation for use by researches worldwide.

ACKNOWLEDGEMENTS

This work was supported by the US Army Research Office, grant number DAAD19-01-1-0502. The views and conclusions herein are those of the authors and do not represent the official policies of the funding agency.

REFERENCES

A. Boukerche, S. K. Das, A. Fabbri, "SWiMNet: a scalable parallel simulation test-bed for wireless and mobile networks", ACM Wireless Networks, 2001, v. 7, Issue 5, pp. 467-486.

X. Chang, "Network simulations with OPNET", Proceedings of Winter Simulation Conference, 1999, pp. 307-314.

B. Groselj, "CPSim: a tool for creating scalable discrete event simulations", Proceedings of Winter Simulation Conference, 1995, pp. 579-583.

R. Guha, O. Kachirski, "Intrusion Detection Using Mobile Agents in Wireless Ad Hoc Networks", Proceedings of the IEEE Workshop on Knowledge Media Networking, KMN'02, pp. 153-160.

R. Guha, O. Kachirski, D. G. Schwartz, S. Stoecklin, E. Yilmaz, "Case-Based Agents for Packet-Level Intrusion Detection in Ad Hoc Networks", Seventeenth International Symposium On Computer and Information Sciences, Orlando, FL, October 28-30, 2002

J. Haines, L. Rossey, R. Lippmann, R. Cunningham, "Extending the DARPA Off-Line Intrusion Detection Evaluations", Proceedings of DARPA Information Survivability Conference & Exposition II, Volume: 1, 2001, pp. 35-45.

A. Heybey, "MIT Network Simulator", MIT Laboratory for Computer Science, 1988.

O. Kachirski, R. Guha, "Effective Intrusion Detection Using Multiple Sensors in Wireless Ad Hoc Networks", Proceedings of 36th HICSS Conference, 2003, pp. 57-64.

O. Kelly et. al., "Scalable parallel simulations of wireless networks with WiPPET: modeling of radio propagation, mobility and protocols", Mobile Networks and Applications, 2000, v. 5, Issue 3, pp. 199-208.

O. Kelly et. al., "Parallel simulations of wireless networks with TED: radio propagation, mobility and protocols", ACM SIGMETRICS Performance Evaluation Review, 1998, v. 25, Issue 4, pp. 30-39.

S. Keshav, "REAL 5.0", Cornell University, 1997, http://www.cs.cornell.edu/skeshav/real/overview.html

R. Lippmann et. al., "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-Line Intrusion Detection Evaluation", Proceedings of DARPA Information Survivability Conference & Exposition II, Volume: 2, 1999, pp. 12-26.

W. W. Liu et. al., "Parallel simulation environment for mobile wireless networks", Proceedings of the 28th conference on Winter simulation, 1996, pp. 605-612.

B. Mah, "INSANE Users Manual", UC Berkeley, 1998, http://www.employees.org/~bmah/Software/Insane

"NS-2 Simulator", VINT Project, 1997, http://www.isi.edu/nsnam/ns/