2005

An Interactive Distributed Simulation Framework With Application To Wireless Networks And Intrusion Detection

Oleg Kachirski
University of Central Florida

AN INTERACTIVE DISTRIBUTED SIMULATION FRAMEWORK WITH APPLICATION TO WIRELESS NETWORKS AND INTRUSION DETECTION

by

OLEG KACHIRSKI
M.S. University of Central Florida, 2001

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2005

Major Professor: Dr. Ratan Guha

ABSTRACT

In this dissertation, we describe the portable, open-source distributed simulation framework (WINDS) targeting simulations of wireless network infrastructures that we have developed. We present the simulation framework which uses modular architecture and apply the framework to studies of mobility pattern effects, routing and intrusion detection mechanisms in simulations of large-scale wireless ad hoc, infrastructure, and totally mobile networks. The distributed simulations within the framework execute seamlessly and transparently to the user on a symmetric multiprocessor cluster computer or a network of computers with no modifications to the code or user objects. A visual graphical interface precisely depicts simulation object states and interactions throughout the simulation execution, giving the user full control over the simulation in real time. The network configuration is detected by the framework, and communication latency is taken into consideration when dynamically adjusting the simulation clock, allowing the simulation to run on a heterogeneous computing system. The simulation framework is easily extensible to multi-cluster systems and computing grids. An entire simulation system can be constructed in a short time, utilizing user-created and supplied simulation components, including mobile nodes, base stations, routing algorithms, traffic patterns and other objects. These objects are automatically compiled and loaded by the simulation system, and are available for dynamic simulation injection at runtime.

Using our distributed simulation framework, we have studied modern intrusion detection systems (IDS) and assessed applicability of existing intrusion detection techniques to wireless networks. We have developed a mobile agent-based IDS targeting mobile wireless networks, and introduced load-balancing optimizations aimed at limited-resource systems to improve intrusion detection performance. Packet-based monitoring agents of our IDS employ a CASE-based reasoner engine that performs fast lookups of network packets in the existing SNORT-based intrusion rule-set. Experiments were performed using the intrusion data from MIT Lincoln Laboratories studies, and executed on a cluster computer utilizing our distributed simulation system.

Dedicated to my dear family.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Ratan Guha, for his invaluable help, support and guidance throughout my degree pursuit. I would also like to thank members of my research committee - Dr. Mostafa Bassiouni, Dr. Sheau-Dong Lang, Dr. Joohan Lee, and Dr. Damla Turgut - for their assistance and guidance in completing this work.

I acknowledge support of the Army Research Office under grants DAAD19-01-1-0502, W911NF04110100 and the National Science Foundation under Grant EIA 0086251. The views and conclusions herein are those of the author and do not represent the official policies of the funding agencies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES AND LISTINGS

LIST OF ACRONYMS

- BPNN – Back-Propagation Neural Network
- CBR – Case-Based Reasoner
- DIDS – Distributed Intrusion Detection System
- DIS – Distributed Interactive Simulation
- DMSO – Defense Modeling and Simulation Office
- FOM – Federation Object Model
- GUI – Graphical User Interface
- HLA – High Level Architecture
- HPC – High-Performance Computing
- IDES – Intrusion Detection Expert System
- IDS – Intrusion Detection System
- ISOA – Information Security Officer's Assistant
- MANETs – Mobile Ad Hoc Networks
- MBS – Mobile Base Station
- MIDAS – Multics Intrusion Detection and Alerting System
- NADIR – Network Anomaly Detection and Intrusion Reporter
- NID – Network Intrusion Detector
- NIDES – Next-generation Intrusion Detection Expert System
- NSM – Network Security Monitor

- OMT – Object Model Template
- PADS – Parallel and Distributed Systems
- PINS – Parallel Interactive Network Simulation architecture
- RPC – Remote Procedure Call
- RTI – Runtime Infrastructure
- SOM – Simulation Object Model
- WINDS – WIreless Network Distributed Simulation framework
- XML – eXtensible Markup Language

CHAPTER 1: INTRODUCTION

1.1 Overview

Developments in the field of computer networking in recent years have resulted in the availability of advanced technologies and models for researchers developing hardware, communication algorithms and networking protocols. These advances gave rise to fundamental questions concerning the security and the viability of many technologies introduced. Security concerns are particularly important when considering a wireless networked system. One type of wireless network in particular, the ad hoc wireless network, is more vulnerable to security violations and misuse than other wireless network topologies. It is often a difficult task to devise a routing protocol, a security sub-system, or an application-level system based on an ad hoc wireless network. Careful design and simulation of such a system is necessary to establish its feasibility under varied operational conditions. Network simulators have long been used for this purpose, saving development time and validating theoretical assumptions. However, due to a wide diversity in wireless network technologies, generic simulators often don't provide the required flexibility and range of features that are essential for wireless network simulation. One of the goals of this research work was to develop a network simulation framework that is geared towards simulation of wireless systems. We have developed an architecture that allows researchers to simulate a wide variety of wireless network topologies and view the network configuration and visualize simulation results in real-

time. Scalability and performance played a major role in the design of our framework. The framework is demonstrated in distributed simulations of intrusion detection systems for ad hoc wireless networks, which was the driving application that necessitated the development of such a simulation system.

Security in wireless networks is an area of major concern and concentrated research. Wireless transmissions are subject to eavesdropping and signal jamming. Physical security of each node is important to maintain integral security of the entire network. Ad hoc wireless networks are totally dependent on collective participation of all nodes in routing of information through the network. To solve these problems, security features have been incorporated into modern wireless networks, such as spread spectrum transmission technology, strong encryption mechanisms, vulnerability analysis, and intrusion detection and prevention systems. Intrusion detection is one of the key techniques behind protecting a network against intruders. While commonly used for wired networks, intrusion detection systems for ad hoc wireless networks are undergoing research investigation. We have developed an agent-based intrusion detection system for wireless networks that takes into account the limited resources and dynamic nature of ad hoc networks to provide a lightweight, modular, low-overhead intrusion detection mechanism based on the mobile security agent concept. Each module represents a lightweight mobile agent with certain functionality, making total resource consumption smaller by separating the functional tasks into categories and dedicating an agent to a specific purpose, with the ability to add new agents to expand the functionality. Since

IDS systems targeting ad hoc wireless networks are still a research problem, to test associated algorithms in our simulations, we have utilized a case-based reasoning engine with SNORT rules for detecting intrusions in wireline networks as a packet-monitoring component of our wireless IDS system.

1.2 Contributions

This dissertation presents the study of the application of distributed system concepts to the simulation of wireless networks and associated algorithms. As a result of this research, a parallel simulation framework was developed. This framework targets large-scale wireless network simulations running on cluster computers and cluster grids. Various studies have been conducted with the help of this simulation framework – specifically, the research into intrusion detection systems for mobile wireless networks and security optimization techniques targeting low-power, resource-conscious systems. Results of the intrusion detection systems research are also incorporated into this dissertation. Our accomplishments, described in detail in the following chapters of this dissertation, can be briefly summarized as follows:

- Developed an interactive cross-platform GUI-enabled Parallel Interactive Network Simulation Architecture (PINS) – with its implementation specializing in executing various WIreless Network Distributed Simulations (WINDS).

- Developed an agent-based intrusion detection system, targeting ad hoc wireless networks with low system resources, and incorporating a new clustering strategy for optimal agent allocation and a load-balancing support for wireless cluster-based packet intrusion detection.

In the first part of this dissertation, we describe the parallel interactive network simulation architecture (PINS) for large-scale network simulations that we have developed. We have implemented our architecture for simulations of wireless networks (WINDS simulation system), demonstrating its use in simulations of routing protocols in ad hoc wireless networks and mobility models in totally mobile wireless networks, as well as its scalability by simulating a large-scale wireless ad hoc network of 5000 nodes on a cluster computer. The framework performance was compared to several widely-used distributed simulation technologies (T-Spaces, High Level Architecture) as applied to parallel intrusion detection simulations on a cluster computer.

In the second part of this dissertation, we discuss the modern intrusion detection systems that we have studied and assess the applicability of existing IDS techniques to ad hoc wireless networks. We then present an intrusion detection system targeting ad hoc wireless networks that we have developed. In the context of this intrusion detection system, we have evaluated the efficiency, correctness and performance of our network packet-monitoring engine using network intrusion data from Lincoln

Labs and introduced a load-balancing mechanism for packet monitoring to further improve performance of our IDS system, followed by analytical verification of results using a queuing model. We have also evaluated hardware and software components for a high-performance parallel intrusion detection system implementation, as well as several popular network simulation packages from a wireless network simulation perspective.

1.3 Organization of the Dissertation

The rest of this dissertation is organized as follows. Chapter 2 introduces technological concepts referenced in this dissertation, describing briefly origins, principles and recent developments in the areas of intrusion detection systems, agent systems, and distributed simulation systems. Chapter 3 presents the distributed network simulation architecture (PINS), and an implementation of a scalable, high-performance wireless network distributed simulation system (WINDS) executing on multiple processors, either a symmetric multiprocessor computer or a network of heterogeneous computers. Guidelines on using our simulation system are also given in Chapter 3. Chapter 4 describes a case study of using WINDS to implement a distributed simulation of a totally-mobile wireless network, demonstrating the process, efficiency, ease of design and implementation and constraints of simulating wireless network systems and applications. Chapter 5 presents our work on intrusion detection systems, describing the intrusion detection framework for ad hoc wireless networks – an agent-based architecture

aimed to minimize the costs of network monitoring in a dynamic environment with limited computational resources. Various aspects of intrusion detection are studied in this chapter and simulated using WINDS. Our network packet-monitoring methodology is presented that uses SNORT rules and XML representation of network data to detect network intrusions. Intrusion detection system scalability issues are discussed, together with practical solution and analytical verification based on queuing theory model. Recommendations are also given for the choice of software and hardware components for constructing a high-performance parallel system. Performance comparisons are made between PINS framework and several widely-used simulation technologies - High Level Architecture/RTI and TSpaces in Chapter 6, using a back-propagation neural network training algorithm on intrusion datasets. Chapter 7 concludes this dissertation.

CHAPTER 2: BACKGROUND KNOWLEDGE

In this chapter we introduce the technological components used in this research. A brief history of key technologies is presented, including the definitions and terms used throughout this dissertation. In the first part of this chapter, we review current work in the area of distributed systems, distributed simulators and supporting mechanisms. We proceed to review intrusion detection systems, their underlying concepts, classifications, working principles and current limitations. We also introduce mobile agent systems, their uses, performance characteristics and security issues.

2.1 Distributed Simulation Systems

Developments in parallel and distributed computing led to the creation of high-performance distributed simulation systems. We first introduce the history of development of parallel and distributed systems, then describe the advances in parallel and distributed simulation research, and review several frameworks for parallel and distributed network simulations. Future research directions in the field are presented late in this section.

2.1.1 History of Development of Parallel and Distributed Systems

All of today's microprocessors (Pentium, MIPS) process data in parallel. The Pentium 4 processor can simultaneously process up to 126 instructions at different stages

of execution. The ideas used in modern parallel architectures were developed long before microprocessors existed. The origins of parallel and distributed computing are traced back to 1953, when IBM introduced the IBM 701 computer with parallel-access memory and arithmetic [36]. As processor speeds increased, researchers saw a bigger bottleneck – the I/O system. In the 1950's the main peripheral device – a magnetic tape – was 1000 times slower accessing data than a processor. In 1958, IBM developed a new computer, IBM 709, which incorporated 6 independent I/O processors that were capable of working in parallel, communicating with main processor [79]. The IBM STRETCH computer, developed in 1959, incorporated breakthrough research concepts [82], such as pre-decoding, memory operand pre-fetch, out-of-order execution, speculative execution based upon branch prediction, branch mis-prediction recovery, and precise interrupts. The first instruction-pipeline computer, ATLAS, was developed at the University of Manchester in 1962. It featured timesharing of several concurrent computing and peripheral operations, multi-programming, virtual storage, paging, and fixed storage (ROM) [5]. Control Data Corporation developed the CDC 6600 in 1964, which supported 10 independent functional units [64]. This machine is arguably the world's first super-computer. This design was followed up in 1969 with the CDC 7600, which included pipelining in every functional unit [65]. The concept of matrix processors was pioneered in ILLIAC IV in 1974, which incorporated 64 processing elements (PEs) [37]. Each PE had some local memory, as well as some memory shared with a neighbor PE. The main control unit could access the entire memory and control the 64 PEs, which were synchronized and would execute the same operation, but on a different set of data. The

CRAY-1 supercomputer [17] was introduced in 1976 and featured the first vector pipelined processor with vector operations, capable of performing over a hundred million arithmetic operations per second.

These computers laid the foundations of the principles of parallel computing and gave rise to four major directions of high-performance parallel and distributed computing technologies. One is vector-pipeline computers that feature pipeline functional units and vector processing of operations that work with arrays of data. An example is CRAY T90 [68]. Massively-parallel computers with distributed memory are another category. These have a large number of microprocessors with local memory, interconnected by a high-speed network, and are a highly configurable and flexible class of parallel computers. Examples are Intel Paragon [38], IBM SP1 [66] and CRAY T3D [16]. Shared-memory parallel computers form a third class. Due to physical constraints of shared memory access, these computers have a limited number of processors. The last class is a hybrid of the first three – cluster computer architecture. This approach has received the most attention in the past years, as a cost-effective and very scalable parallel solution.

Beowulf cluster implementation has raised parallel and distributed computing to a new level. Beowulf cluster computers were first introduced in 1994 as a result of the Beowulf Project at CESDIS [8]. A Beowulf system is a collection of personal computers constructed from commodity-off-the-shelf hardware components interconnected with a local-area network and configured to operate as a single unit, a parallel computing

platform, using an open-source network operating system (e.g., Linux). The driving design philosophy of a Beowulf system is to achieve the best possible price/performance ratio for a given computing problem. For many problems it's possible to achieve an order of magnitude improvement in price/performance compared with "conventional" parallel supercomputer designs. Common uses of cluster computers are traditional technical applications such as simulations, biotechnology, petro-clusters; financial market modeling, data mining and stream processing; and Internet servers for audio and games.

2.1.2 Principles of Parallel and Distributed Simulation Systems

A computer simulation or a computer model is a computer program which attempts to simulate an abstract model of a particular system. Computer simulations have become useful tools in modeling natural systems in physics, chemistry, and biology, human systems in economics and social science, and in the process of engineering new technology. Eventually, the processing power of a single computer system became inadequate for certain problems of global scale; and the use of super-computer wasn’t always an option for many researchers. With the advent of computer networks, and later on computing clusters, the field of parallel and distributed simulation has received widespread recognition.

Parallel and distributed simulation [23] is concerned with issues introduced by distributing the execution of a discrete event simulation program over multiple computers. Parallel simulation is concerned with execution on a multiprocessor computing platform

containing multiple CPUs communicating frequently over a high-speed interconnection network. Examples include parallel computers and cluster computers. Distributed simulation executes on a loosely-coupled system, with individual computers distributed geographically and connected via a wide-area network. An example of a large-scale distributed system is a collaboration system running over Internet 2 between several major universities. In both cases, the execution of a single simulation model is distributed over multiple computers. Subdividing a simulation model to be executed among multiple computers offers several benefits:

- Reduced execution times for certain simulations that can be subdivided efficiently to run on multiple computers, with final results combined.
- Geographic distribution of simulation components allows inclusion of data from multiple scattered sensors/participants, and utilizing diverse communication media among processors taking part in the simulation.
- Diverse computer architectures can be integrated into a single simulation, based on availability and applicability.
- Simulations can be made fault-tolerant by replicating one processor's tasks among several processors.

Subdividing a simulation among several computers poses certain problems. The synchronization problem raises the issues of synchronizing the events occurring on different computers during the joint simulation execution. Early works on conservative

synchronization algorithms that solve this problem include work by Bryant [11] and Chandy and Misra [12]. Optimistic synchronization algorithms originated from the Time Warp algorithm [40] introduced by Jefferson in 1985. Another problem lies in efficient subdivision of a sequential simulation program among several computers for parallel execution. According to Amdahl's law [1], the algorithm, and not the number of processors limits the speedup:

$$S \leq \frac{1}{f + \frac{(1-f)}{p}}$$

where S is the speedup of parallelized program execution, f is the fraction of non-parallelizable operations (f=1, means a completely sequential program), and p is the number of processors. The law doesn't explicitly take into account the speed of the interconnection network between processors. In the early days of computer networks, network communications were a limiting factor for distributed systems. Modern high-speed networks achieve gigabit communication speeds, thus drastically reducing the inter-processor communication delay, and improving the performance of distributed systems. Two strategies exist in dividing simulation systems among several computers. One is the space-parallel approach, where a number of concurrent simulation tasks is spread among multiple processors, executing these tasks in parallel, and combining the results of execution. Another one is the time-parallel approach, where the simulation time axis is divided into intervals assigned to different processors for concurrent execution. In this approach, processors handling every-but-first time interval assume some knowledge on the state of the simulation at the entry point of time, and proceed with the simulation

execution based on this assumption. If the boundary states don't match, certain adjustments may be performed. This approach, however, is less widely used, because of the constraints of the state-mismatch problem.

2.1.3 Modern Distributed Simulation Systems and HLA

Academic institutions and the defense community are the driving forces behind parallel and distributed simulation systems, placing emphasis on reduced simulation execution time and interoperability of heterogeneous simulation systems. The SIMNET project (1983 – 1990) was the first major leap of the defense community into the realm of distributed simulation systems [55]. This work led to the development of Distributed Interactive Simulation standards that encompassed integration of multiple simulation systems, sensors, and live participants. The most recent development in distributed simulation community is called High Level Architecture (HLA) [59], driven by DARPA, which has become a standardized framework (IEEE 1516) for modeling and simulation [87]. HLA is a component-based software architecture that incorporates subsystems for communication and data sharing, synchronization, and time management. This framework facilitates distributed and multi-platform computing in simulation systems by providing standard integration architecture for separate and remote applications, thus facilitating reuse of simulation components. Each component simulation application is called a *federate*, and a simulation system composed of two or more federates is a *federation*. The Run-Time Infrastructure (RTI) is the interconnection bridge between the simulators. Simulation objects have to be compliant with the data definition for the

simulation, which is described in the Object Model Template (OMT). The data shared between the federates within the federation is defined by the Federation Object Model (FOM). The Simulation Object Model (SOM) defines the data that federates share with the federation. The common RTI provides the following services to the federation: federation management, declaration management, ownership management, object management, time management and data distribution management. The architecture of HLA is presented in the diagram below.

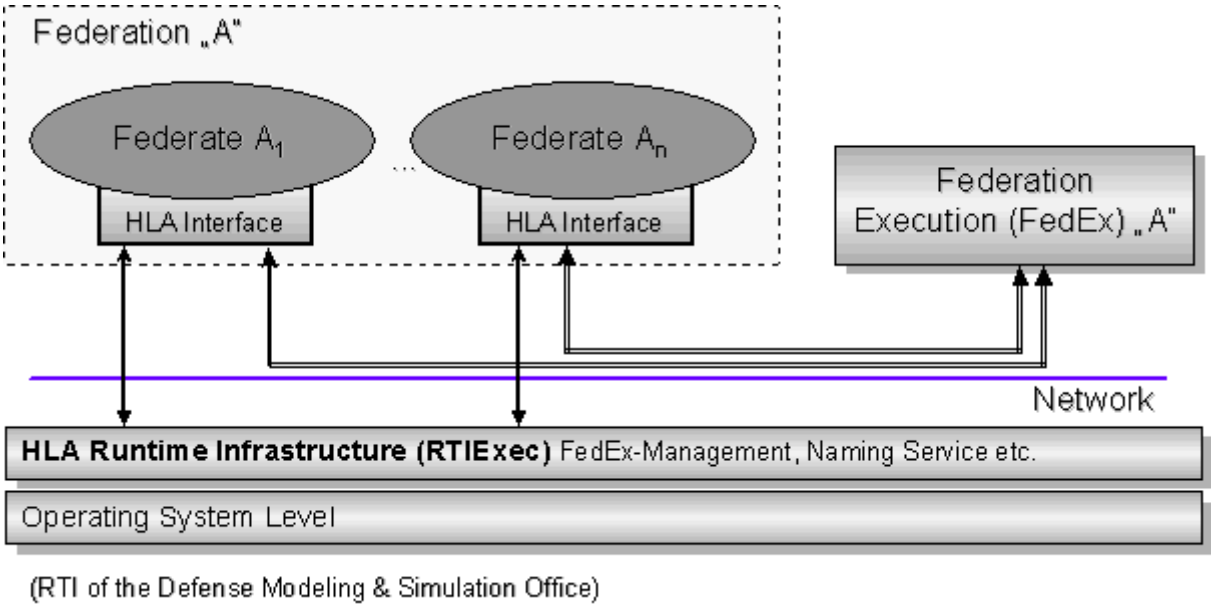


Figure 2.1. High Level Architecture Runtime Infrastructure.

The High Level Architecture is merely a standard and an abstracted framework for distributed simulation. It is neither a modeling tool, nor a simulator. It doesn't have capabilities for data display and analysis, or any sort of user interface. HLA does not eliminate the programming effort that is required to build a functioning simulation system.

The flow of simulation and its architecture, as well as providing data processing and meaningful presentation and interpretation of results, is still a task of the simulation developer. Even though HLA provides specifications for seamless integration of various simulation components, the efficiency of the entire simulation heavily depends on the implementation and operational performance of each individual simulator comprising the simulation.

2.2 Intrusion Detection Systems

2.2.1 Overview

Recent advances in networking technologies, algorithms and protocols made it possible to think of a network of computers as an information processing unit. The progress made in the area of computer networking gave rise to many new technologies – grid computing, distributed databases, 100% fault-tolerant systems, etc. With these advances came fundamental questions concerning the security of such systems. It's quite easy to construct a secure centralized system – providing physical security of the equipment, assigning simple admission procedures and authentication mechanisms, and providing operating system support for process level security. Making a computer network secure is much more difficult. Much of the equipment isn't in secure locations, and most of the communication goes over insecure data links. Common techniques of protecting computer network involve the use of cryptography and secure protocols. Physical network security is addressed by isolating access to security databases and

network equipment, and in some cases, protecting the network media itself. (An example of that is to enclose network cabling in gas-pressurized pipes and to monitor the pressure changes within the pipe system to prevent physical access to the network medium.) Still, we can only guarantee complete security of a system if its security mechanisms *prevent* unauthorized access to system resources and data. However, at present, the best we can do is to detect such intrusions into the system, so that action can be taken to stop the intrusion, repair the damage, and secure the system against similar future intrusions.

The concept of intrusion detection was introduced first in 1980 [2]. An intrusion attempt or a threat was defined as a deliberate unauthorized attempt to access information, manipulate information, or render a system unreliable or unusable. Since then, several methodologies of intrusion detection have been devised. The next sections discuss why intrusion detection systems are needed, intrusion detection approaches, evaluation, and limitations of intrusion detection systems.

2.2.2 Need for Intrusion Detection Systems

Users of computer systems assume properties of the system such as confidentiality, integrity and assurance of functionality to protect sensitive information. This information is the target of computer system intruders, who will attempt to exploit flaws in an operating system as well as application programs to gain access to such information. The consequences of such subversions can be quite dramatic. An example is the famous

internet worm of 1988 [74]. There are several ways to deal with such subversions, however, a few observations have to be made here:

- It is impossible in practice to build a completely secure system. One reason is that despite extensive software testing, bugs do occur in the code, and quite frequently so. An illustration of this fact is that software products carry disclaimers instead of warranties.
- Cryptographic methods have problems. The user component of security is perhaps a more vulnerable one (lost or stolen passwords).
- Even if protected from outside attacks, computer systems are targeted by insiders, and more often than expected. FBI records show that 80% of intrusions were due to insider attempts. One reasonable explanation is that firewalls are pretty effective at keeping outside intruders at bay.
- Efficiency of system operation is inversely proportional to the level of access control. Many choose a more efficient system thus sacrificing security.

Therefore, we should accept the fact that there will be no absolutely secure system for a while, and prepare for system attacks, detecting them as soon as possible and taking appropriate action. And this, in essence, is the task of an Intrusion Detection System (IDS).

One of the more utilized ways of detecting intrusions is by analyzing the audit data generated by system and application processes. An audit trail is a log of activities on a system that are recorded in a file as they occur. The logging process generates very large data files (depending on the system, log files can be hundreds of megabytes in size, e.g. [31]), which are difficult to analyze manually. IDS automates the task of analyzing the audit trail and pinpointing any suspicious activity which can be classified as a probable intrusion. A thorough analysis of the audit trail is important, since it allows tracing all activities of intruders, detecting the damage to the system, and helping reconstruct the damaged components. This enables us to use IDS to detect intrusions almost real-time from audit data, as well as analyzing the system vulnerabilities later on to prevent similar intrusions.

The following definitions from [2] are used throughout this dissertation:

- *Risk:* Accidental or unpredictable exposure of information, or violation of operations integrity due to the malfunction of hardware or incomplete or incorrect software design.
- *Vulnerability:* A known or suspected flaw in the hardware or software or operation of a system that exposes the system to penetration or its information to accidental disclosure.
- *Attack:* A specific formulation or execution of a plan to carry out a threat.

- *Penetration:* A successful attack - the ability to obtain unauthorized (undetected) access to files and programs or the control state of a computer system.

2.2.3 Classification of Intrusion Detection Systems

Intrusion detection techniques are generally classified into two categories – anomaly detection and misuse detection.

Anomaly Detection model makes use of operational profiles, which represent normal system activity for each user, application or a process. During the training period, regular system activities are recorded and a normal system behavior profile is established. During system operation, any deviations from a normal profile are considered anomalous, and an alert is raised. According to SANS Institute's taxonomy of anti-intrusion techniques [63], anomaly detection engine employs one or more of the following techniques:

- *Threshold Monitoring* sets values for metrics defining acceptable behavior (e.g., fewer than some number of failed logins per time period). Thresholds provide a clear, understandable definition of unacceptable behavior and can utilize other facilities besides system audit logs.
- *User Work Profiling* maintains individual work profiles to which the user is expected to adhere in the future. As the user changes his activities, his expected

work profile is updated. Some systems attempt the interaction of short-term versus long-term profiles; the former to capture recent changing work patterns, the latter to provide perspective over longer periods of usage.

- *Group Work Profiling* assigns users to specific work groups that demonstrate a common work pattern and hence a common profile. A group profile is calculated based upon the historic activities of the entire group. Individual users in the group are expected to adhere to the group profile. This method can greatly reduce the number of profiles needing to be maintained. Also a single user is less able to "broaden" the profile to which they are to conform.

- *Resource Profiling* monitors system-wide use of such resources as accounts, applications, storage media, protocols, communications ports, etc., and develops a historic usage profile. Continued system-wide resource usage - illustrating the user community's use of system resources as a whole - is expected to adhere to the system resources profile. Resource profiling is user-independent, potentially allowing detection of collaborating intruders.

- *Executable Profiling* seeks to monitor executables' use of system resources, especially those whose activity cannot always be traced to a particular originating user. Viruses, Trojan horses, worms, trapdoors, logic bombs and other such software attacks are addressed by profiling how system objects such as files and printers are normally used, not only by users, but also by other system subjects on the part of users. In most conventional systems, for example, a virus inherits all of the privileges of the user executing the infected software.

The software is not limited by the principle of least privilege to only those privileges needed to properly execute. This openness in the architecture permits viruses to surreptitiously change and infect totally unrelated parts of the system. User-independent executable profiling may also be able to detect collaborating intruders.

- *Static Work Profiling* updates usage profiles only periodically. This prevents users from slowly broadening their profile by phasing in abnormal or deviant activities which are then considered normal and included in the user's adaptive profile calculation. System administrator-controlled updates allow the comparison of discrete user profiles to note differences between user behavior or changes in user behavior.

- *Adaptive Work Profiling* automatically manages work profiles to reflect current (acceptable) activity. The work profile is continuously updated to reflect recent system usage. Profiling may be on users, groups, or applications. Adaptive work profiling may allow the system administrator to specify whether flagged activity is: 1) intrusive, to be acted upon; 2) not intrusive, and appropriate as a profile update to reflect this new work pattern, or 3) not intrusive, but to be ignored as an aberration whose next occurrence will again be of interest. Activity which is not flagged as intrusive is normally automatically fed into a profile updating mechanism.

- *Adaptive Rule Based Profiling* differs from other profiling techniques by capturing the historical usage patterns of a user, group, or application in the form

of rules. Transactions describing current behavior are checked against the set of developed rules, and changes from rule-predicted behavior flagged. As opposed to misuse rule-based systems, no prior expert knowledge of security vulnerabilities of the monitored system is required. "Normal usage" rules are generated by the tool in its training period.

Misuse detection essentially checks for activities that match the descriptions of undesired activity. This approach attempts to draft rules describing known undesired usage (based on past penetrations or activity which is theorized to exploit known weaknesses) rather than describing historical "normal" usage. Rules may be written to recognize a single auditable event that represents a threat to system security, or a sequence of events that represent a prolonged penetration scenario. The effectiveness of provided misuse detection rules is dependent upon how up-to-date the vulnerabilities database is. Misuse detection may be implemented by expert system rules, model based reasoning, state transition analysis systems, or neural nets, as described below:

- *Expert Systems* may be used to code misuse signatures as if-then implication rules. Signature analysis focuses on defining specific descriptions and instances of attack-type behavior to flag. Signatures describe an attribute of an attack or class of attacks, and may require the recognition of sequences of events. A misuse information database provides a quick-and-dirty capability to address newly identified attacks prior to overcoming the vulnerability on the target system.

- *Model Based Reasoning* attempts to combine models of misuse with evidential reasoning to support conclusions about the occurrence of a misuse. This technique seeks to model intrusions at a higher level of abstraction than the audit records. In this technique, intrusion descriptions are first developed at a high, intuitive level of abstraction in terms of sequences of events that define the intrusion. This technique may be useful for identifying intrusions which are closely related, but whose audit trail patterns are different. It permits examination of only portions of relevant data.

- *State Transition Analysis* creates a state transition model of known penetrations. In the Initial State the intruder has some prerequisite access to the system. The intruder executes a series of actions which take the target system through intermediate states and may eventually result in a Compromised State. The model specifies state variables, intruder actions, and defines the meaning of a compromised state. Evidence is pre-selected from the audit trail to assess the possibility that current system activity matches a modeled sequence of intruder penetration activity (i.e., described state transitions lead to a compromised state). The higher level representation of intrusions allows this technique to recognize variations of scenarios missed by lower level approaches.

- *Neural Networks* offer an alternative means of maintaining a model of expected normal user behavior. They offer a more efficient, less complex, and better performing model than statistical models of system and user behavior. Neural network techniques may be found to be more efficient and less computationally

intensive than conventional rule-based systems. A lengthy, careful training phase is required with skilled monitoring. After the training period, the network tries to match actual commands with the actual user profile already present in the net. Any incorrectly predicted events actually measure the deviation of the user from the established profile.

In reality, each approach individually doesn't yield an optimal level of intrusion detection. Therefore, two or more methods of intrusion detection are usually combined, resulting in a hybrid intrusion detection system.

2.2.4 Limitations of Existing Systems

Despite advances in research on intrusion detection technologies, the field of intrusion detection has still many problems to overcome. This section will demonstrate some of the limitations that certain ID technologies possess.

Starting with Anomaly Detection models, we can recall that all intrusive activities are assumed to be anomalous; however, in practice only a small intersection of sets of intrusive and anomalous activities would classify as intrusion attempts. This leads us to the conclusion that anomaly detection models potentially (a) flag non-intrusive anomalous activities as intrusive – known as a problem of false-positives, and (b) flag intrusive non-anomalous activities as non-intrusive, which gives rise to a problem of false-negatives (a potentially more dangerous problem). Another problem with statistical

methods is that they can be “trained” by intruders to eventually classify intrusive events as normal. Recalling section 2.1.2, anomaly detection models make use of thresholds – setting the threshold level too high or too low will result in one of the problems outlined above. Approximation of threshold levels can result in a high rate of false positives or a high rate of false negatives across a non-uniform user population. Profiling of users and groups is difficult due to irregularities in the user base and the set of underlying activities for seemingly similar users (i.e., users with similar job titles may have different work habits). Similarly, resource profiling requires a somewhat difficult task of correctly interpreting the changes in overall system usage. Rule-based profiling also introduces constraints in terms of the large number of rules required to be stored, which negatively impacts the performance and training time.

Misuse detection resembles the virus detection systems in the way known attack patterns are detected; however, misuse detection methodology is of little use when it comes to detecting unknown attacks. The expert system approach has a significant drawback – the expert system has to be formulated by security professionals, and therefore it is as strong as the person who programs it. In addition, certain rule interdependencies are introduced, which makes addition or deletion of new rules more complicated. State transition systems possess an array of problems. First, attack patterns can specify only a simple sequence of events. These patterns cannot describe and therefore detect such violations as denial of service attacks, failed logins or passive network listening.

Other problems with intrusion detection systems involve denial of service attacks on the IDS itself (an intruder can disable the IDS by issuing a large number of events resulting in alarms up to the point where an IDS can no longer cope with its tasks), and inability of the IDS to process encrypted or fragmented traffic. For instance, a well-known IDS Snort detects intrusions by matching the hexadecimal substring from the network packet payload with the existing intrusion database. If the intruder's traffic was encrypted or somehow modified, the attack would sneak past the intrusion detection system. Another point to consider is that when an alert is raised by the IDS, some intrusion activity has been going well before that, thus preventing us from discovering the exact starting point of an attack. In summary, intrusion detection systems cannot be relied upon as the only source of defense – rather, they should be used in unison with firewalls, strong encryption techniques, intrusion prevention systems and manual supervision.

2.2.5 Summary of Work on Intrusion Detection Systems

In 1980, James Anderson first proposed that audit trails should be used to monitor threats [2]. The importance of such data had not been comprehended at that time and all the available system security procedures were focused on denying access to sensitive data from an unauthorized source. Anderson also classified intrusions into several types, and separated internal intruders from external ones.

In 1987, Dorothy Denning presented an abstract model of an Intrusion Detection System [21]. This paper was the first to propose the concept of intrusion detection as a solution to the problem of providing a sense of security in computer systems. The basic idea of the model is to maintain a set of profiles for subjects (any entity taking part in system operation – e.g., users). When an audit record is generated, the model matches it with the appropriate profile, checking for abnormal behavior and reporting detected anomalies.

In 1988, the Internet worm (also known as the Morris worm) caused the Internet to be unavailable for about five days [74]. This incident brought the need for computer security into the spotlight. The same year, Teresa Lunt et al. refined the intrusion detection model proposed by Denning and created IDES (Intrusion Detection Expert System) [52]. This system was designed to detect intrusion attempts against a single host. An improved version was developed in 1995, called NIDES (Next-generation Intrusion Detection Expert System). Also in 1988, the Haystack system [72] was developed in order to assist Air Force Security Officers detect misuse of the mainframes used at Air Force Bases, and MIDAS (Multics Intrusion Detection and Alerting System) was created for the same reasons for the National Computer Security Center's Multics mainframe [70].

In 1989, Wisdom and Sense came out from the Los Alamos National Laboratory, and Information Security Officer's Assistant (ISOA) from Planning Research Corporation. A new concept was introduced in 1990, with NSM [33] (Network Security Monitor, now

called Network Intrusion Detector or NID): instead of examining the audit trails of a host computer system, suspicious behavior was detected by passively monitoring the network traffic in a LAN. NSM has several perceived advantages over audit trail-monitoring ID systems. First, the IDS gets instantaneous access to network data. Second, the IDS is hidden from the intruder because it is passively listening to network traffic. Therefore, it cannot be shut off or its data compromised. Finally, the IDS can be used with any system, because it is monitoring network traffic, protocols for which are standardized.

In 1991, a different idea was introduced with NADIR (Network Anomaly Detection and Intrusion Reporter) and DIDS (Distributed Intrusion Detection System) [73]: the audit data from multiple hosts were collected and aggregated in order to detect coordinated attacks against a set of hosts.

In 1994, Mark Crosbie and Gene Spafford suggested the use of autonomous agents in order to improve the scalability, maintainability, efficiency and fault tolerance of IDS [18]. Instead of a single large IDS defending the system, they proposed an approach where several independent, small processes operate while cooperating in maintaining the system. The advantages claimed for this approach are efficiency, fault tolerance, resilience to degradation, extensibility and scalability. The foreseen drawbacks include the overhead of so many processes, long training times, and the fact that if the system is subverted, it becomes a security liability. Their idea fit well with the ongoing research on software agents in other areas of computer science.

Another approach to address the scalability deficiencies in most contemporary intrusion detection systems was proposed in 1996, with the design and implementation of GrIDS [75]. This system facilitates the detection of large-scale automated or coordinated attacks, which may even span multiple administrative domains. In 1998, Ross Anderson and Abida Khattak offered an innovative approach to intrusion detection, by incorporating informational retrieval techniques into intrusion detection tools [3]. And as the research in the field continues, we see that this paradigm is proposed as an answer to the security requirements of other technological areas, such as mobile networks.

2.3 Agent Systems

2.3.1 Overview

Mobile Agents are processes that migrate under their own control in a heterogeneous network. When an agent is transported to a new host, its state is saved and sent embedded with the agent, then restored at a new host, allowing the agent to continue its execution. Mobile agents are divided into two kinds – strong mobility and weak mobility agents, the former being different from the latter in that the control state is transferred along with the data and code of the mobile agent to each host. The two types are used for different applications. Strong mobility enables agents to be independent decision-making units and is used, for instance, for load-balancing applications, whereas weak mobility is more suited towards event-driven agents and the technology behind this functionality is

consistently supported by Java virtual machines – a platform that most of mobile agent systems utilize.

2.3.2 Uses of Mobile Agent Systems

In our intrusion detection framework discussed in Chapter 5, we have chosen to rely on mobile agents for a number of reasons. There are certain properties that mobile agents possess and certain guidelines that they follow, which are useful to our system, as discussed below:

- A mobile agent doesn't have to actually move – instead it should follow certain guidelines to access specific resources and make intelligent decisions whether to access a remote resource or a local one.
- Mobile agents add to the complexity of systems. While having a number of advantages, they are not actually adding to a set of high-level operations to a service, but rather remotely invoke low-level operations.
- Mobile agents reduce bandwidth usage by filtering out only the best-matching results and transport them across the network, rather than the entire dataset. They can act as a server-side filter for the data.
- Multiple agents can be dispatched to a number of locations and concurrently obtain required information. This reduces total task completion time.

- Agents can replicate themselves and move to multiple locations from an optimally-selected position within the network, thus reducing latency. For example, instead of sending 10 mobile agents from the source workstation, one agent can be sent to the gateway of the home network, and then replicated on the gateway system to be sent to multiple networks.
- The user can dispatch an agent and close the connection link with the network, while agents independently gather the required results and return to the originating system whenever it becomes possible.
- Agent systems generally provide system usage statistics, which enables them to perform load balancing – by sending some agents off to systems with lower resource usage.

Many arguments have been made against mobile agent systems. The primary reason for these arguments is that existing systems provide similar functionality without any need for modification. However, in many cases the use of mobile agent systems will dramatically improve system performance and reduce operational costs. The diagram below, for example, demonstrates the advantage of using agent systems as opposed to using the Remote Procedure Call mechanism:

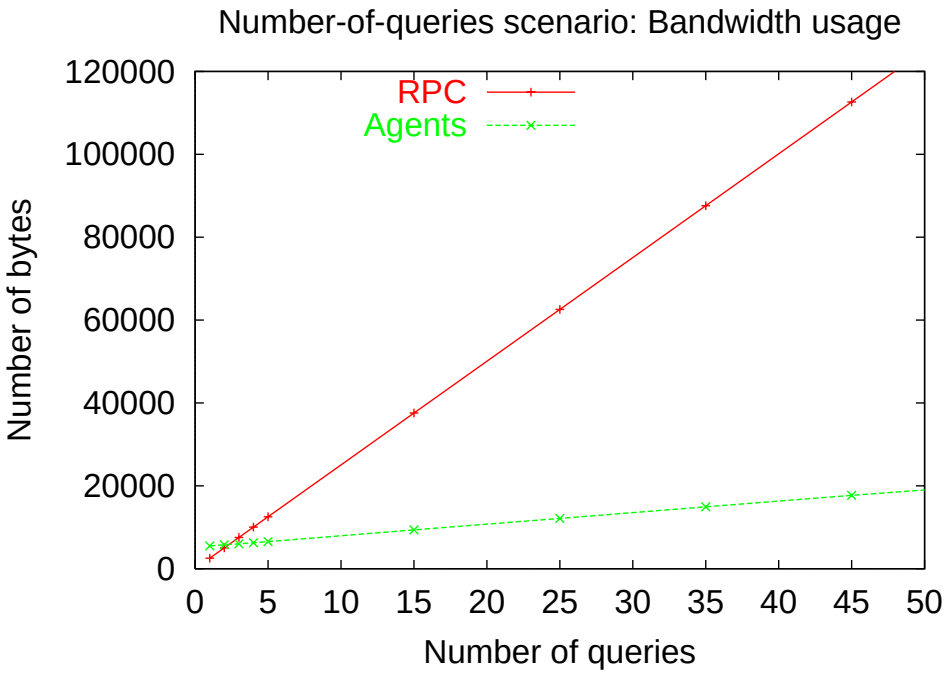


Figure 2.2. Advantage in Bandwidth Reduction when using Mobile Agents versus Remote Procedure Calls.

Agent systems possess a number of other advantages over traditional client-server and distributed systems, as discussed and illustrated in [48]. After overcoming certain technical challenges (e.g., performance, scalability, standardization issues), there still remains an issue of security – a major area of vulnerability of the early mobile agent systems. The issue of agent system security is addressed in the following section.

2.3.3 Mobile Agent System Security

Mobile agent systems have numerous points of potential security breaches [15]. Hostile mobile agents can attack hosts, threatening server resources. Agents can have

their data integrity and privacy jeopardized by a malicious host posing as an agent server. Insecure networks make attacks on mobile agents easier. Therefore, certain security measures must be maintained to ensure:

- Secure communication
- Secure agent transfer
- Protection of host resources from malicious data
- Protection of agent's code and data against tampering
- Secure control of remote agents
- Authentication of mobile agents

Since 1998, much work has been done in this direction. Current mobile agent systems use techniques discussed in [27, 81] to protect hosts against malicious agents. Agents can be protected by means of encryption and other techniques, discussed in [39]. Although an open environment such as the Internet can pose a threat to the use of mobile agents, their uses for private and corporate applications are increasing. Although certain advances (and user education efforts) must be made before the security problems are addressed adequately for all Internet applications, current work promises that shortly mobile-agent systems will be secure enough for general use.

CHAPTER 3: PARALLEL INTERACTIVE NETWORK SIMULATION SYSTEM – ARCHITECTURE AND IMPLEMENTATION

Research in computer networks is one of the major areas of Computer Science and the IT industry. A lot of time and money is spent on implementing network infrastructures in the industry, residential and educational sectors. Often, a computer network under consideration is very complex, and requires careful planning, modeling and analysis before deployment. Networking algorithms should be rigorously tested under a range of operating conditions prior to implementation and standardization. Various network simulators have been used for these purposes. However, due to a wide diversity in network technologies, generic simulators often don't provide required flexibility and range of features that are essential for network simulation. One of the goals of our research work was to develop a parallel network simulation architecture that is geared towards simulation of wireless systems. We have developed an architecture that allows researchers to simulate a wide variety of network topologies and view the actual network and simulation results in real-time. Unlike many other network simulation systems, in our simulation framework heavy emphasis is placed on intuitive user interface and clear presentation of simulation objects and their interactions. In this chapter we describe our parallel network simulator architecture, and then introduce an implementation of the architecture targeting wireless network simulations – WINDS (WIreless Network Distributed Simulation framework).

3.1 Existing Work on Network Simulation

Researchers continually work on developing new communication protocols, algorithms and methods for computer networks. Research papers in the network community propose new algorithms offering better performance, quality of service and other benefits. A technology of choice should be carefully evaluated before being adopted in practice. Primary method of such evaluation is to create a computer simulation of a particular topology, communication mechanism or an algorithm. Several generic network simulators offer convenient, robust simulation platforms for modeling large-scale networks. Our primary objective was to model an intrusion detection system for wireless ad hoc networks, which served as a driving force behind developing a parallel network simulation architecture, which was extended in its implementation to wireless network simulations. The key requirements were flexibility, extendibility and scalability, as our intention was to devise a simulation system capable of being extended to very large scale simulations, running on several processors. Several well-known general-purpose network simulators were considered to create our simulation model of a large-scale wireless network.

Primary choice of simulator packages was discrete-event simulators. Many existing event-driven simulators support inter-object communications by message passing and process events from the queue of events as they become available. Examples include MIT

Advanced Network Architecture group's NETSIM [35], UC Berkeley's INSANE [53], LBLN's NS, based on REAL [56]. Another popular simulator is NS-2 [56] - a discrete event simulator that provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless networks; however, the implementation incurs substantial memory overhead and increases the complexity of simulation code. Based on numerous published results, it is not easy to scale NS-2 beyond a few hundred simulated nodes; only recently, simulation researchers have shown it to scale, with substantial hardware resources and effort, to simulations of a few thousand nodes [60]. User-defined objects can be created in almost all of the above simulator packages; however, in majority of cases, only one specific language of choice (such as C) was available for implementing those objects, resulting in reduced portability of the simulation system and dependencies on a specific platform. While this may not be a problem for a single-computer simulation, or for a cluster computer with identical configuration of all nodes, a definite problem arises while deploying the simulation framework on a heterogeneous distributed system. Another problem with compiler-dependent approach is the reduced target clientele. Some simulators allow user-defined objects to be created and manipulated via scripts [53, 56]. A brief comparison of several of the above mentioned simulators is given in [13].

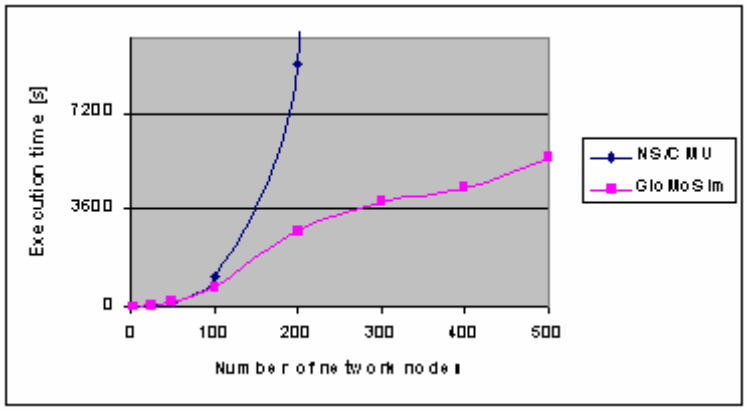
One of our requirements was an intuitive user interface support, which would allow us to visually demonstrate the concepts behind our intrusion detection algorithms in real time by getting snapshots of the simulation execution. NETSIM [35], NEST [46],

OPNET [13] and some others provided such a GUI, while many other simulator packages only provide numerical results. Some of the commercially-available simulation systems, such as MIL3 Software's OPNET, were also considered. OPNET has proven to be a well-composed, extensible and scalable simulator package. However, lack of source code, minimal support for experimental wireless network configurations and licensing issues prevented us from using it.

Intrusion detection system induces a high processing load on the simulator (especially for large-scale simulations). With this in mind, several parallel and distributed simulator packages were considered, such as CPSIM [26] and Columbia University's NEST and its extension REAL [46]. NEST has many protocols built-in, implements client-server model for distributed simulation and provides a design-time GUI for creating network topologies. NS-2 also can be extended to multi-processor systems using Akaroa-2 - a tool for running quantitative stochastic discrete-event simulations on UNIX multiprocessor systems or networks of heterogeneous UNIX workstations by creating multiple instances of an ordinary simulation program and running them simultaneously on different processors [57]. However, for our purposes, a specialized simulation framework for large-scale wireless network simulations is desired, as it provides basic out-of-the-box support for mobile wireless networks, including wireless routing, mobility support, and other specific features. For high-performance simulation, a tightly-coupled parallel system is required to minimize impact of slow network inter-processor communications. This necessitates two important requirements – the simulation

framework should target wireless networks, and should be efficiently implemented to execute on parallel architectures.

We have considered several simulators targeting wireless networks. SWiMNet [10] is a scalable framework developed for parallel simulation of wireless and mobile PCS networks; it eliminates most of the event dependencies by having a pre-computation stage before feeding events into the simulator, which can reduce the number of rollbacks in its parallel execution, but poses additional constraints on the simulation. WiPPET framework deals with issues of evaluating wireless propagation and various wireless communication protocols [44], and doesn't provide the level of abstraction in its object model that we require. An extended version of WiPPET, TeD, is a C++ implementation of a parallel wireless simulation framework that supports built-in wireless protocols such as GSM, TDMA and AMPS [43]. In [51], a conversion from sequential to a parallel wireless network simulator is presented, with considerable reduction in simulation execution time for time-consuming complex scheme simulations. Another popular wireless network simulator GloMoSim [25] targets simulations at multiple layers, concentrating on physical layer modeling. In terms of scalability these wireless simulators are limited, as the following figure demonstrates, comparing NS-2 and GloMoSim'99 for parallel simulations [77]:



The review of the network simulators mentioned above concludes that:

- Modeling of wireless nodes is not flexible enough to include desired functionality (i.e., mobility models for ad hoc networks) in the case of generic network simulators
- Extensive software development effort is required to implement commonly occurring features of the simulation
- Few simulators allow working with external data files (e.g., TCP trace files in binary format)
- Source code is sometimes not available to extend the design and functionality of the simulator
- Most simulators are relying on C/C++ libraries compiled for a particular architecture, thus making it non-portable between different computer systems
- Learning curve is usually steep even for simple simulations

Our simulation architecture was developed as a research tool to provide time-saving flexible simulation test-bed targeting a wide range of network architectures. The

implementation of this architecture – a parallel interactive simulation framework – is a cross-platform, interactive, portable, GUI-driven and can be used as a scalable parallel wireless network simulator for a variety of purposes, such as routing in ad hoc networks, mobility models of totally mobile wireless networks [7, 19], sensor networks, Bluetooth Pico-nets, etc.

In this chapter, we first present the simulator architecture design. Framework modules are described in detail in the next section, as well as a generic simulation process. We also list requirements and limitations of our simulation framework. Then, its design is extended to the parallel and distributed simulation environment, and simulation performance is established. Key design issues are also described in detail in this chapter.

3.2 Network Simulator Architecture

3.2.1 Design Philosophy

The objectives of this research and development project were to reduce redundant software design efforts in the area of network simulation, establish a framework general enough to be used for simulations of many network technologies, and provide for common base for the exchange of models relating to any network infrastructure. The object-oriented nature of the software and the use of a popular programming language (Java) for implementation allow researchers to easily modify, reuse and share whole systems or system components. Our framework also includes customizable user interface

that can be easily adapted to a specific problem. Very intuitive design and layout of the graphical environment allow the system to be used for demonstrational or educational purposes. GUI can be executed separately from the simulation engine and function as a visual demonstration of an algorithm or a system. 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
- Performance comparable to other distributed technologies
- Scalability

Following these guidelines, we have developed PINS architecture – a flexible, portable, network simulation framework that can be used to simulate a variety of applications for optical, wireless and wireline networks. We have implemented PINS architecture on a symmetric multiprocessor cluster computer to address the issues of scalability with large-scale simulations; results are discussed in the next section.

The design of our simulation framework 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 network is built from objects – network nodes, event generator, and

communication channels. Connections between network nodes are maintained by the connections module incorporated into the routing protocol object. Any network infrastructure is supported and can be configured - Ethernet, burst-switched optical, ad hoc wireless and hybrid (e.g., totally-mobile) 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.

Computer networks are particularly convenient to be modeled via object-oriented approach because they typically consist of discrete components, easily thought-of as interactive objects (for instance, network nodes, protocol stacks, packets) that interact with each other by message passing. By changing a particular module, simulations can be modified in a short time. As source code is provided, functionality of simulation event generator can be modified to accommodate new networking paradigms. Users can create shared libraries of swappable modules that describe their system for an easy experiment replication.

Popular network simulators aim at supporting every aspect of network communications, such as, for example, every layer of many communication protocols or every communication subsystem [25, 77]. This adds tremendous overhead to the simulation system, often resulting in scalability problems and slow execution times. Our simulation framework avoids these problems, by implementing only the core

functionality of a network – supporting simulation clock, network communication, routing and network object mobility (for wireless applications). These functions are highly-abstracted, allowing the user to include specific communication protocols, 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. Therefore, our simulation system choice was a time-stepped simulation system, as opposed to majority of discrete-event wireless simulation systems. The rationale behind this choice is the fact that in a network simulation with large network traffic, there is no particular advantage of running as-fast-as-possible simulation, jumping to the next event right away; however the issue of communication synchronization is complicated, when developing a parallel simulation system. The intuitive graphical user interface of our framework therefore reflects any changes in the simulation execution as they happen, in scaled real-time, allowing the user to adjust the simulation parameters at run-time. Another advantage or our framework is ability to use almost any data file as a source of network communications. Data file pre-processor converts binary packet data into the format used by our simulation framework, translating simulated network address space 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 simulation.

3.2.2 Simulator Components

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

- GUI (graphical user interface)
- Simulator Core Module
- Network Traffic Module
- Data Logging Module

The diagram below (Figure 3.1) shows the architecture of our framework. Some of the modules carry optional functionality and can be included into the simulation as necessary. The functionality of each module is described further on in this section. Architecture Use Case UML diagrams for the User and the System actors are presented in figure 3.2 (a) and 3.2 (b). Basic simulation flow is shown in figure 3.5.

Graphical User Interface Module

The graphical user interface (GUI) module shows the simulation execution in real-time (figure 3.1). 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 settings can be adjusted via GUI module, such as network communication parameters and simulation clock interval. GUI module also provides controls for simulation execution – user can pause simulation, step through the simulation one clock cycle at a time, add and remove

simulation objects. The GUI interface is generic enough to accommodate various network entities, but can be modified to reflect specifics of a wireless architecture being studied.

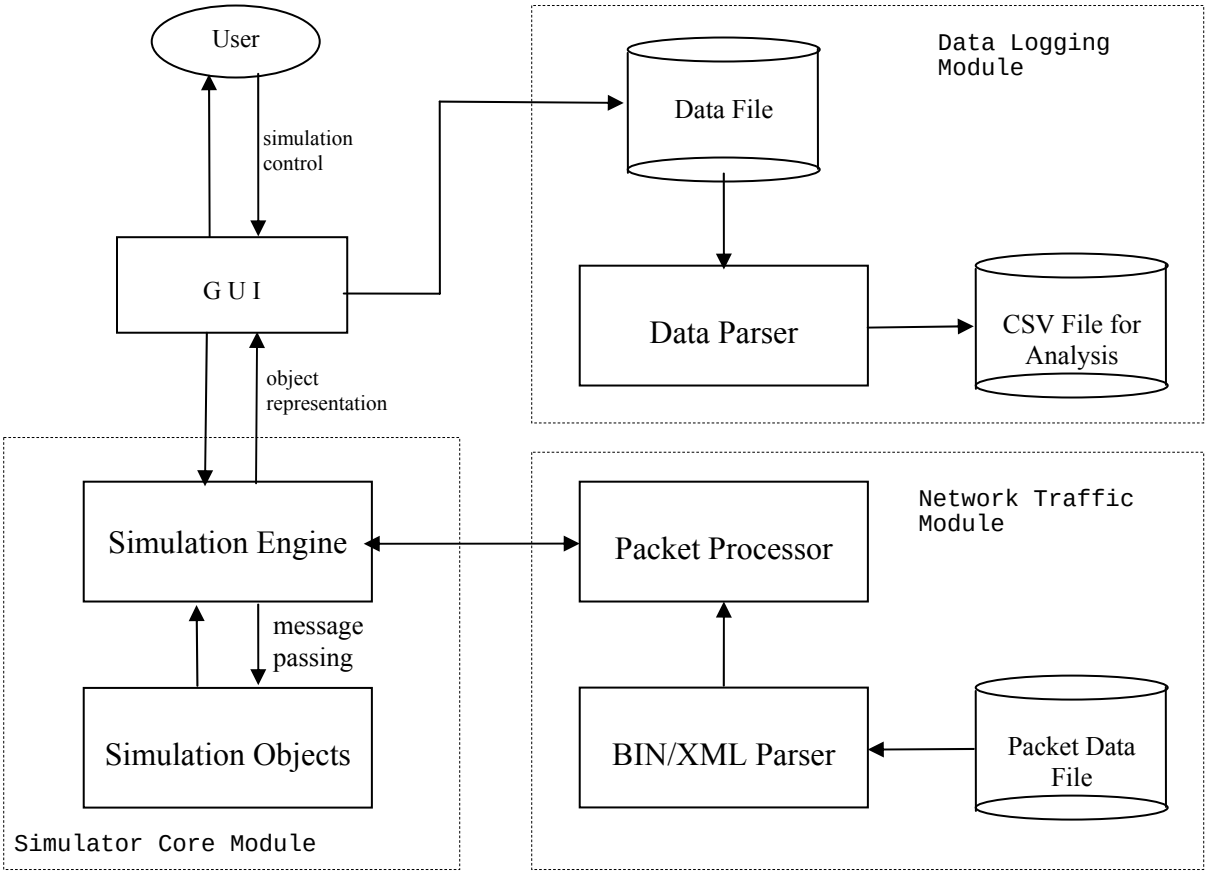


Figure 3.1. PINS Component Architecture.

Our simulation framework provides a flexible user interface component – for instance, you can use it to supply specific parameters used in simulations of ad hoc or infrastructure wireless networks, using the same interface. GUI module also records simulation statistics in a log window. Data logging module processes this data at the end

of simulation and saves the results in CSV format, which can be later processed by a mathematical analysis software package.

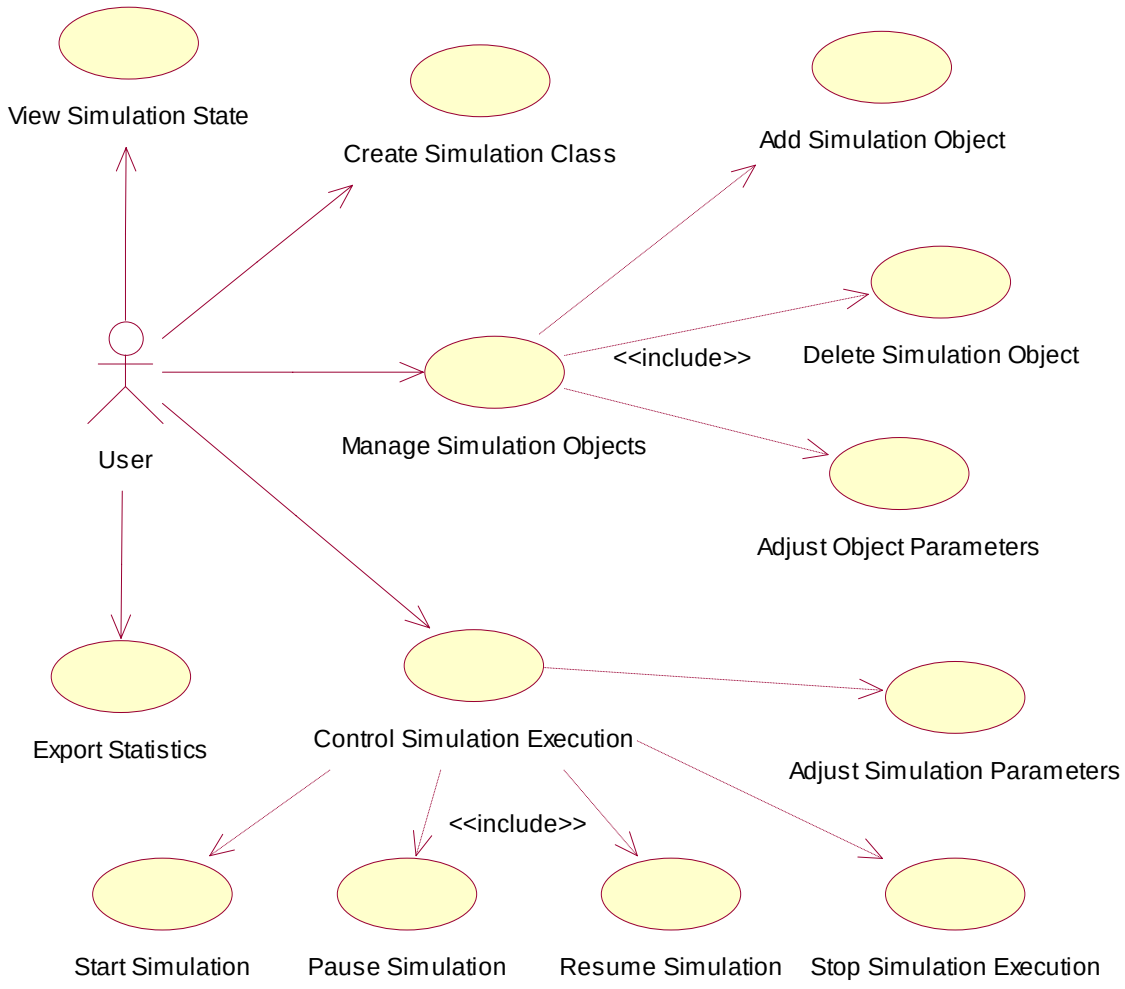


Figure 3.2(a). PINS User Use Cases.

Simulator Core Module

Simulation Engine: The heart of the PINS framework is a Simulation Engine. One of the design requirements is that PINS is a scaled-real-time time-stepped system. The

simulation is tied up to a graphical user interface, allowing users to monitor simulation progress in real-time and modify simulation parameters on the fly. This would not be otherwise possible with a discrete-event simulation system. The simulation engine runs an internal clock, the speed cycle of which can be controlled by the researcher 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 certain 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 class keeps track of object definitions of all user-defined objects taking part in a simulation (such as wireless nodes, routing algorithm used, mobile base stations and stationary routers), and instantiates them at runtime.

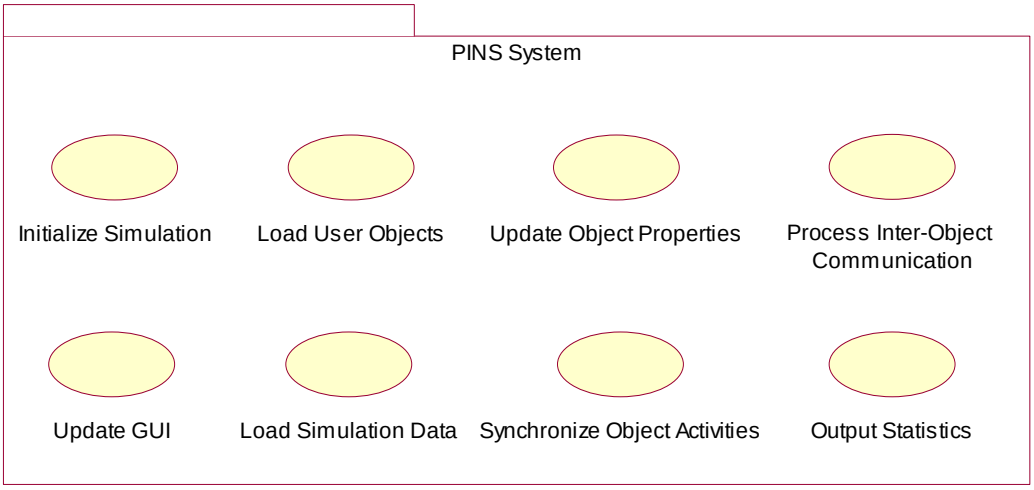


Figure 3.2(b). PINS System Use Cases.

Simulation Objects: The simulation objects execute independently and are time-synchronized via the simulation engine clock. Our framework has a few pre-defined simulation objects. One such object type is a wireless node object. Wireless nodes are members of any wireless network simulation, and can be either stationary or mobile. A smooth random-destination no-wait mobility pattern is embedded by default in a wireless node object. User can modify the motion pattern by implementing a different mobility algorithm (or read waypoints from a data file). This allows simulations to be flexible and account for many possible node mobility patterns. Each network node object also includes two methods used for inter-object communication – Send and Receive methods. Send method is invoked when a node is transmitting packets, and has a source, destination, protocol, port and payload as its arguments. When applied to wireless networks, 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 ad hoc and infrastructure wireless networks.

Routing protocol is another simulation object. The routing protocol included with our framework is a simple table-driven protocol, which is implemented programmatically as a separate routing class. To modify the routing protocol, user must include all necessary parameters in a network node object, and implement their own routing protocol class. Since the simulation framework serves communications in exact same way as a

real-life communication network, all existing routing protocols are supported. Depending on the system configuration in question, framework user can also instantiate a Base Station object, as used in the simulations of wireless infrastructure and totally-mobile wireless networks. The base station object supports Send and Receive methods (as described above), and is supplied with center-of-gravity motion pattern, which is used in the totally-mobile wireless network simulation. When base station object is used, the routing algorithm class must also be updated to route all network packets via appropriate base station.

Network Traffic Module

Network traffic for our framework is generated by the Network Packet Processor. The implementation of this 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

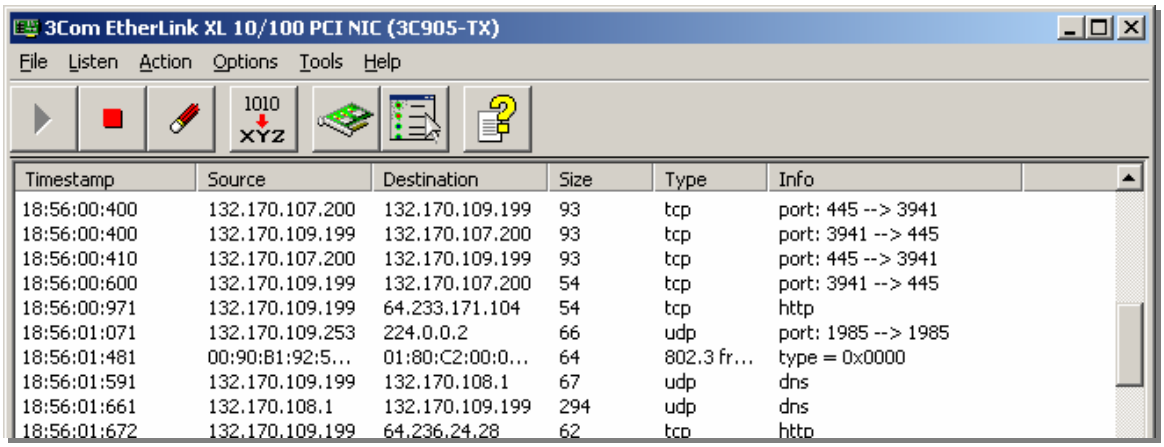


Figure 3.3. Sample Network Packet Capture.

network node (source). Pre-processing is performed on a binary data file obtained from

network packet capturing software (such as TCPDUMP). A sample capture of network packets is shown in figure 3.3 and a decoded packet example is shown in figure 3.4. Here, a network trace file is obtained and converted for the use in our simulation framework by substituting IP address space by a simulation network object address space. Each packet is decoded, and a new packet is created after processing addressing information. Further pre-processing may be done by stripping payload off the packets, adding data to the packets (and updating packet header), filtering packets for specific features (e.g., a timestamp, or a particular protocol).

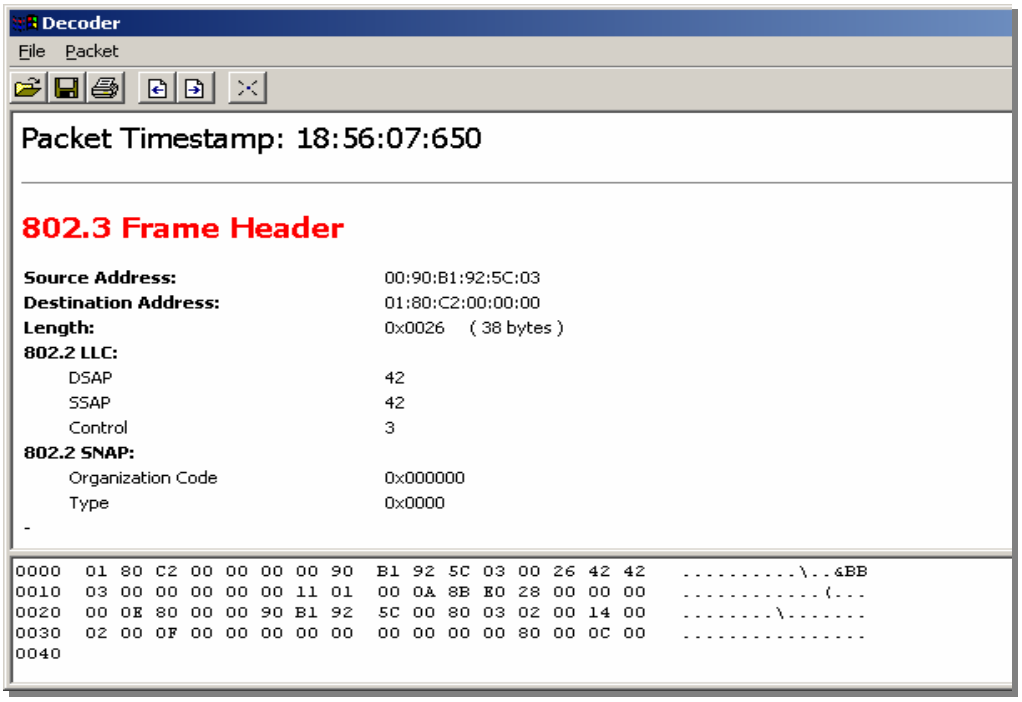


Figure 3.4. Decoded Packet Information.

Our simulator 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 offline. Packet processor object generates packets at times specified by

timestamps of each packet processed, and makes use of simulation engine's internal clock to time distribution of the packets. This allows us to speed 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 commonly used comma-separated data file format) by the CSV file generator.

3.2.3 Requirements and Limitations of Simulation Framework

Certain limitations exist in our single-processor simulation framework, and a few guidelines have to be strictly adhered to. Knowledge of Java programming language is required for developing user simulation objects, as well as clear understanding of principles of wireless communications. Object-oriented nature of our framework requires

all designs to comply with ideology of object-oriented programming and class definitions used in the simulation framework. The framework design imposes minimal hardware requirements. This, however, entails a restriction on the size of simulation supported. The number of wireless nodes is limited in framework specification, but can be changed to a higher value, if hardware of the computer used to run the framework is powerful enough – as the next section shows, where we have implemented the PINS architecture as a parallel simulation framework executing on multiple processors to address the issues of scalability and the speed of simulation execution.

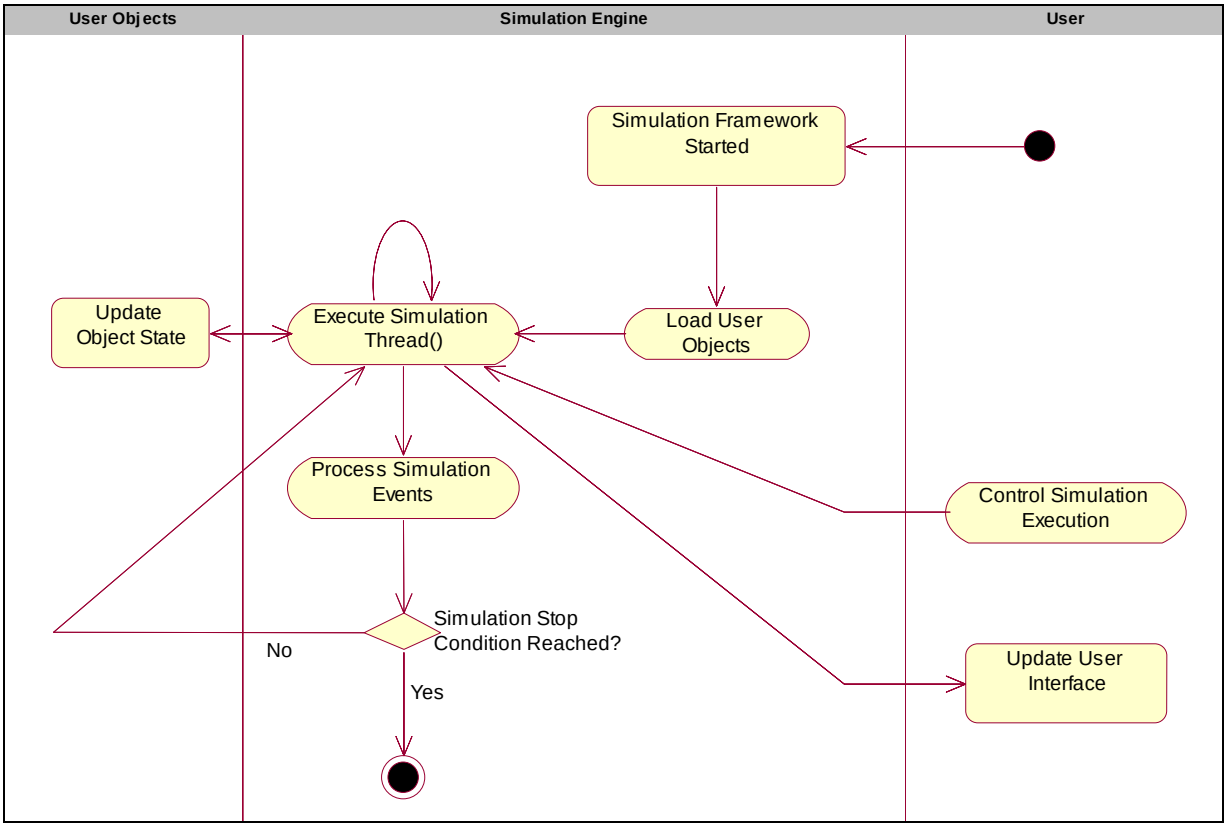


Figure 3.5. PINS Simulation Execution Activity Diagram.

The design of the framework may be further improved by making use of configuration files, where a user can specify required parameters, such as wireless communication range, initial number of nodes, etc. Optimally, a scripting support is beneficial to any simulation system, thus simplifying design and control of user simulations; however it is out of scope of the framework design for now. Graphical user interface communicates directly with the simulation engine and displays progress of the simulation derived from data supplied by the simulation execution via the main simulator class. This requires all simulations to be structured around the simulator class, and not directly with user modules. Finally, a Java run-time environment has to be installed on a client machine running the simulation. A Java compiler compliant with Sun's version 1.3 or greater is also required if a simulation incorporates user-defined simulation objects that need to be compiled.

3.3 Wireless Network Distributed Simulation System

In this section we extend the framework architecture to simulations of wireless networks in multiprocessor environments and describe the optimizations to further improve the performance of the simulation system for large-scale simulations of wireless networks. We call our distributed simulation framework WINDS – WIreless Network Distributed Simulation framework.

3.3.1 WINDS Design 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 performing simulations in excess of 200 wireless nodes. 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 concepts of a distributed simulation have been presented in Chapter 2. As for the practical implementation, first, the entire set of objects in the simulation is partitioned into several parts. For instance, we can divide 100 simulation objects evenly between 5 processors, assigning objects to processors in a round-robin fashion. As user interface provides capability to instantiate multiple user objects of the same type with one command, multiple types of objects are instantiated sequentially, resulting in a balanced allocation of simulation objects of all types between the processors. During simulation execution, each of the processors performs computations

relevant to objects assigned to it. As most of our single-processor simulations have exhibited computationally-intensive properties user objects versus smaller amount of inter-object communications, distributing simulation objects among multiple processors promised good scalability results. Communication between objects is handled by the communication broker – if two objects are handled by the same processor,

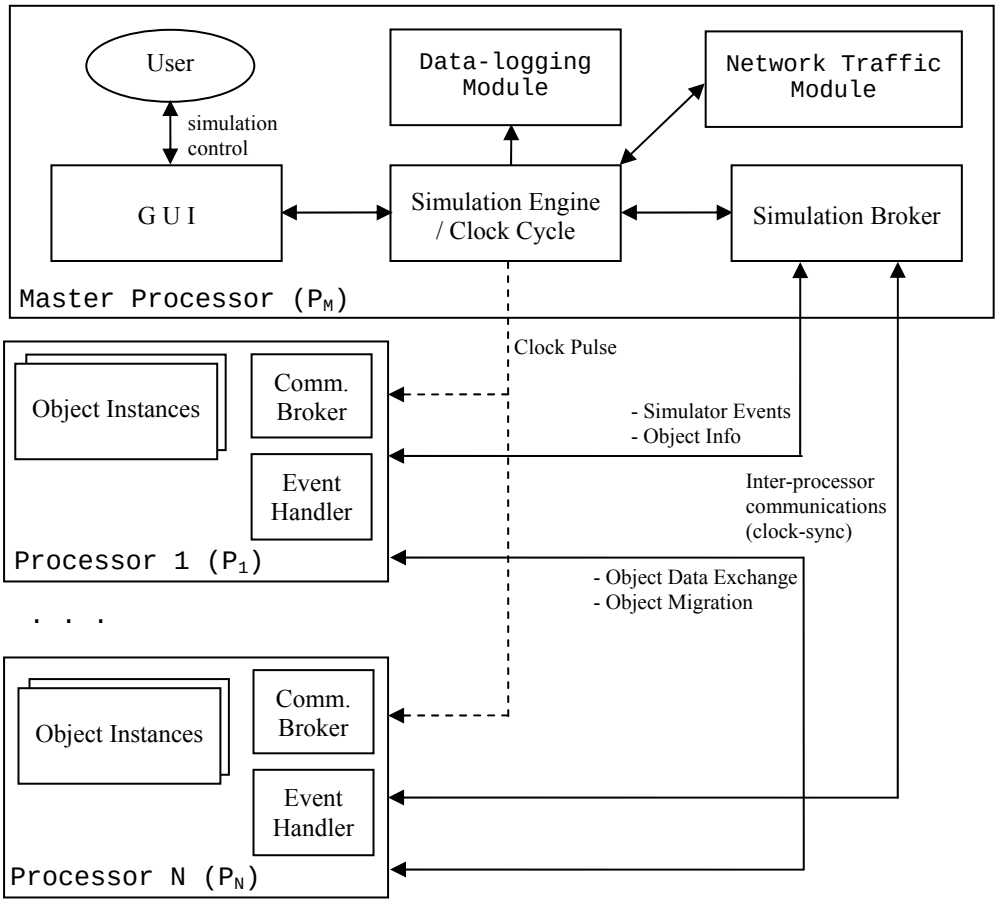


Figure 3.6. WINDS Architecture for Distributed Wireless Network Simulations in Multiprocessor Environments.

communication happens in exact same way as in the uni-processor system; if two (or

more) objects are assigned to different processors, inter-processor communication takes place. Our WINDS architecture [28] is presented in figure 3.6.

The simulation procedure is as follows. 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 control module on that processor. Once all objects have been created, simulation execution commences. 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 uni-processor 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 synchronously from the master processor to each client processor to control simulation execution. The screenshots of the simulation system are shown in figures 3.7(a), (b). The base class diagram for the WINDS framework shown in the figure 3.7(c) illustrates the main Java classes of the framework and their associations.

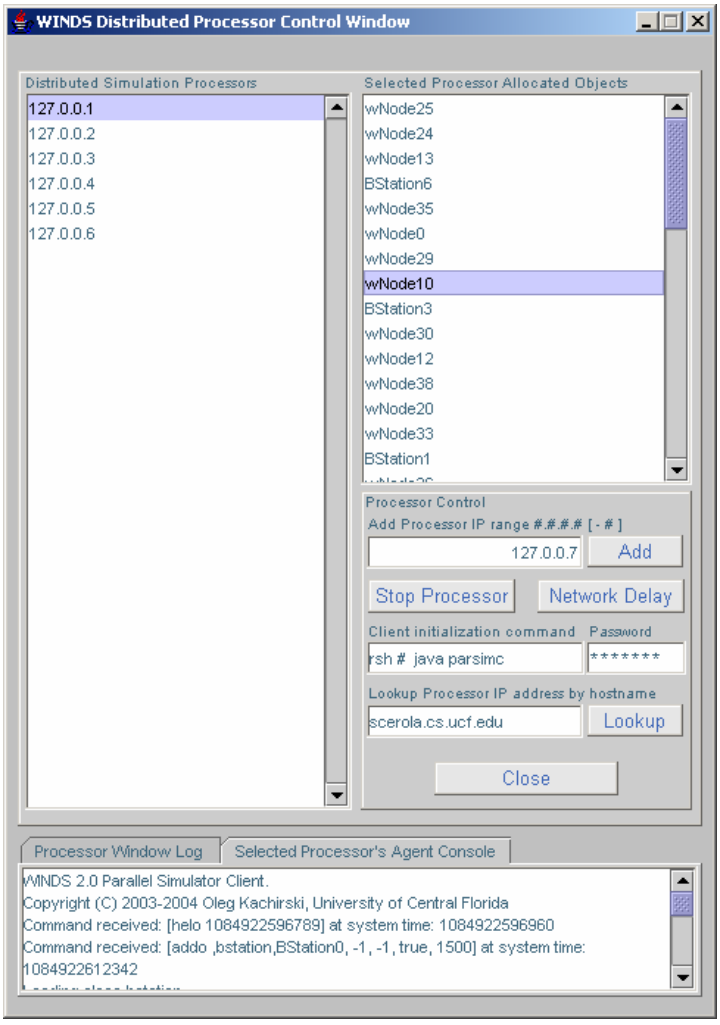


Figure 3.7(a). Processor window showing object allocation among processors.

Figure 3.7(a) shows (on a local host example with multiple aliases) the list of all processors taking part in the simulation, and the associated user objects. Each processor executes a processor agent that relays all console messages to the master node's simulator GUI screen. These messages may then be viewed by the user in regular or verbose mode, allowing him/her to obtain detailed account of simulation activities on each processor. This screen allows the user to simultaneously instantiate simulation agents on an entire

range of processors (e.g., all the nodes of a cluster computer) by specifying a range of IP addresses and a command for each processor. For example, setting IP range of 10.0.0.101-121 will result in instantiating simulation agents on all specified nodes. User can also look up IP address by machine's name, and specify an agent invocation command and optionally, user's password. An example of such a command on a Linux cluster computer would be: `rsh –n # | tcsh | cd ~/WINDS | java parsimc &`.

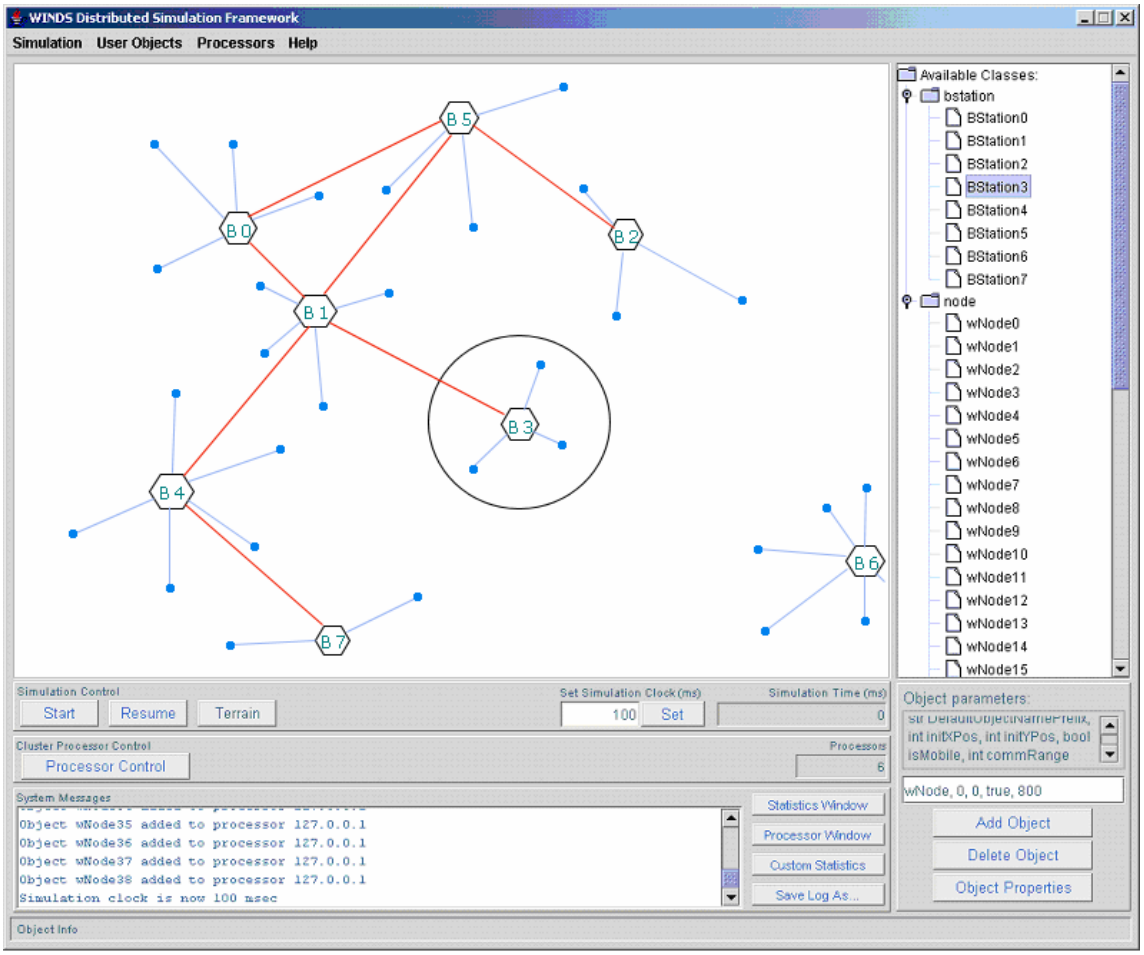


Figure 3.7(b). Main WINDS screen – Infrastructure Wireless Network Simulation.

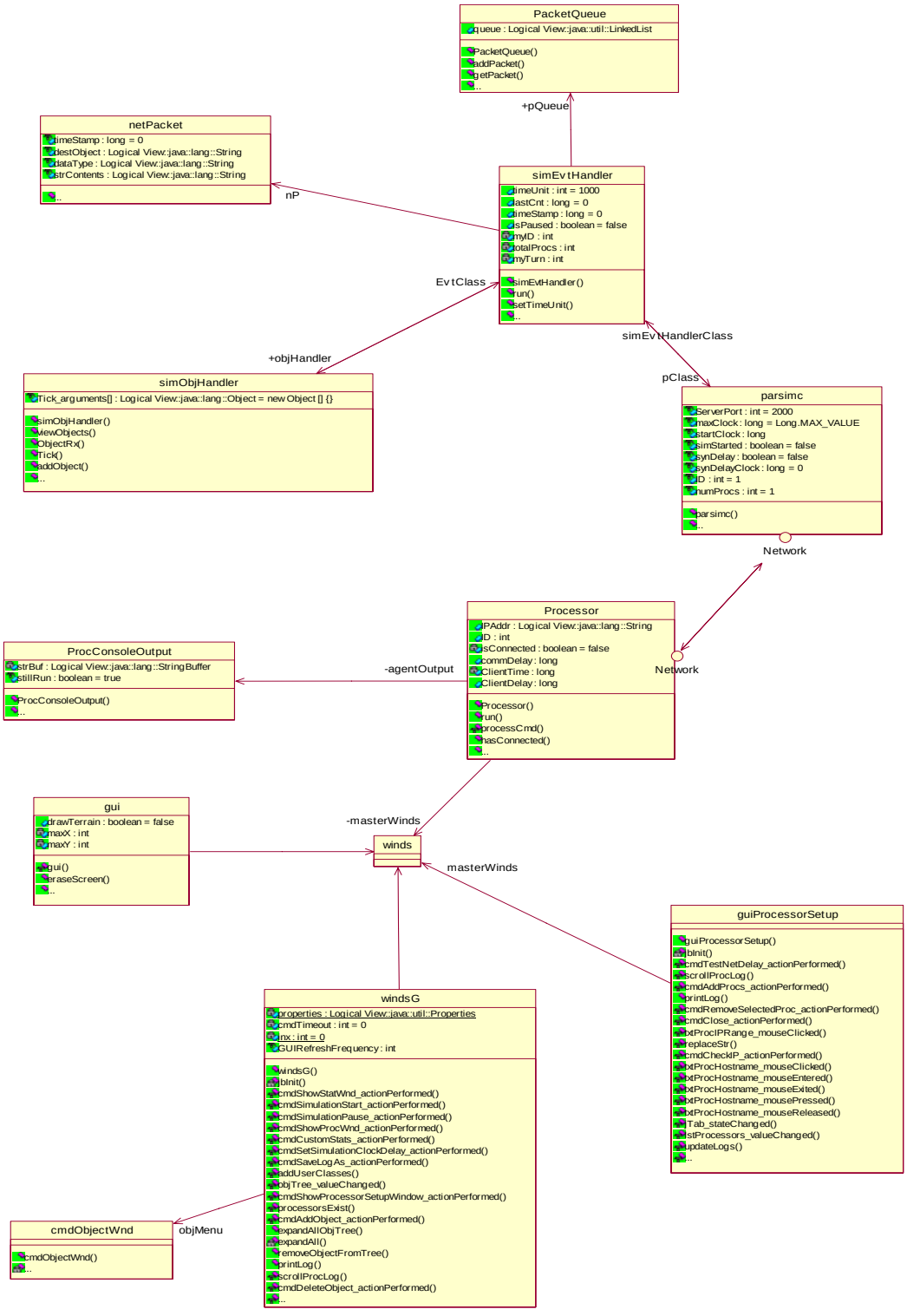


Figure 3.7(c). WINDS framework core class diagram.

Figure 3.7(b) demonstrates totally-mobile wireless network simulation in progress. Panel on the upper right shows all available user classes and instantiated objects for each class. Lower right panel provides means to instantiate user objects from class templates. Main window shows graphical presentation of simulation execution, with user object interactions in real time. Figure 3.8 shows a simulation of a wireless ad hoc network's mobility models for a large number of wireless nodes.

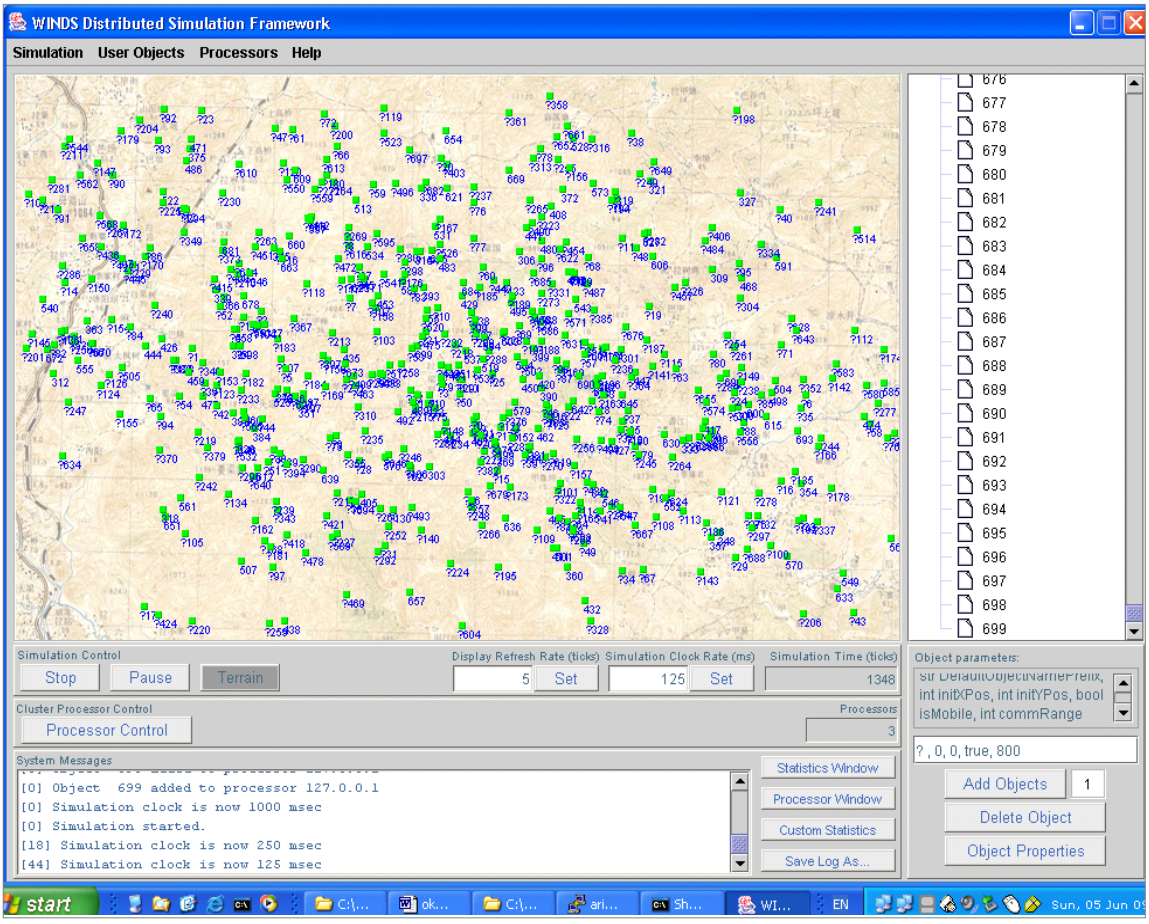


Figure 3.8. Simulating 700 user objects on 3 processors.

Here, 700 user objects are distributed among 3 processors – two on a local LAN, and a third one on the cluster computer, separated from the local LAN by a residential high-speed cable line. Simulation clock delay is set at 125 ms, which is possible despite substantial inter-processor communication latency of this geographically-distributed simulation, due to internal clock cycle advance on each processor and clock-synchronization mechanism discussed in the following section.

3.3.2 Simulation Clock Synchronization

The WINDS architecture was initially deployed on a Scerola cluster computer [67] with 128 900 MHz AMD Athlon processors. Each cluster node has dedicated memory space of 256 MB, and can access data concurrently from a replicated disk subsystem (15 GB each). Inter-processor communication is conducted via Fast Ethernet switch at 100 Mbps. More recently, our Distributed Systems Lab has acquired a new Ariel cluster computer [4] with 32 dual-processor 2.6GHz P4 nodes, each having a 2GB memory and 40GB hard drive space, and interconnected via Gigabit Ethernet. We have seen significant simulation execution time improvements over the previous cluster computer, as shown, for instance, in chapter 5. Initially, the simulation framework was relying on time synchronization mechanism of a distributed operating system running a cluster computer. However, for long-running simulations, the local clock discrepancy between multiple nodes of a cluster was too large. In an extended time period, physical clocks on these nodes would differ by 400ms or more between clock synchronizations.

Our simulation framework utilizes a distributed time synchronization algorithm supplemented by a centralized push time service from the master node. As with all time synchronization algorithms, there is an issue of measuring and predicting network delays affecting transmitted time-sync messages. Initially simulation clocks are synchronized via a centralized algorithm based on Network Time Protocol (NTP) [23]. A message is sent from the master processor to each simulation node, and returned. Four time stamps are assigned to this time-sync message, as shown in figure 3.9.

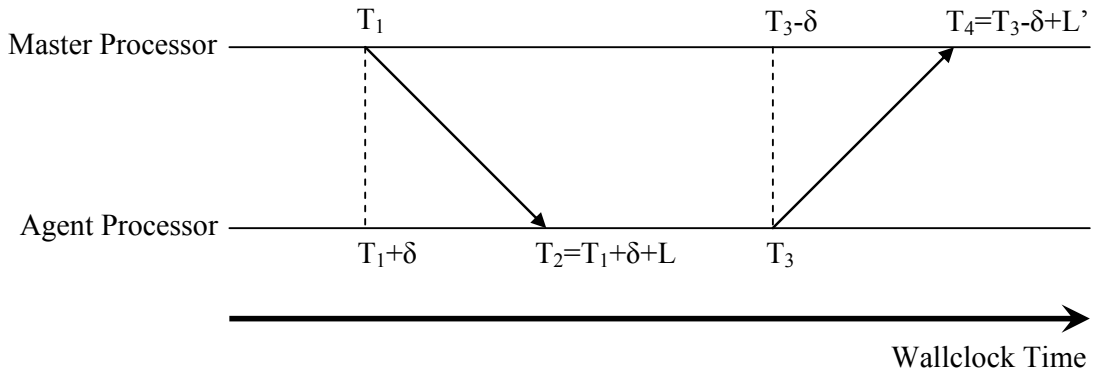


Figure 3.9. Determining offset and network delay in NTP.

Master processor (P_M) sends a time-sync message at local time T_1 to the simulation agent's processor (P_i) that at that moment has a local time of $T_1 + \delta$, where δ is the offset between two clocks (and is unique for each agent processor taking part in the simulation). The message is received by the agent processor at a time $T_2 = T_1 + \delta + L$, where L is the network latency to send a message from P_M to P_i. After processing the message, P_i returns it back to P_M at time T_3, with the assumed P_M's clock equal $T_3 - \delta$. The final timestamp is the P_M's clock value of T_4, which equals $T_3-\delta+L'$, where L' is the message latency from

P_i to P_M. The estimated clock offset for P_i is then computed to be $\delta_e = \delta + \frac{L - L'}{2}$, which is the midpoint offset between the wall clocks of P_M and P_i.

During simulation execution, each processor independently increments simulation time based on the wall clock time advance, and sends its current simulation clock to the master processor at a certain frequency. If a master processor doesn't receive a clock value from a particular processor within pre-defined time interval (equal to max (L, L')), the network delay increase is assumed, and that processor's clock value is discarded from synchronization step. The maximum clock value is then selected from all processors' simulation clock values, and is broadcast to all processors using a synchronous delay update command. To ensure simultaneous execution of simulation commands on all nodes of the cluster, the built-in simulation command synchronization mechanism has the following steps:

- Master processor (P_M) gets simulation time T_S
- P_M sends $T_S + \Delta T$ to $P_{1...N}$, where ΔT is the derived maximum inter-processor communication delay: $\Delta T = \max (\delta_{e,1}, \delta_{e,2}, \ldots, \delta_{e,N})$.
- Each processor sleeps until local simulation clock hits $T_S + \Delta T$
- Simulation command is executed on each processor simultaneously
- Each processor periodically sends internal current clock cycle to P_M
- P_M sends out current clock cycle to those processors that lag behind for clock cycle update via dedicated TCP connections used for simulation control.

The following sequence diagram (3.10) demonstrates the time synchronization procedure used by the WINDS framework.

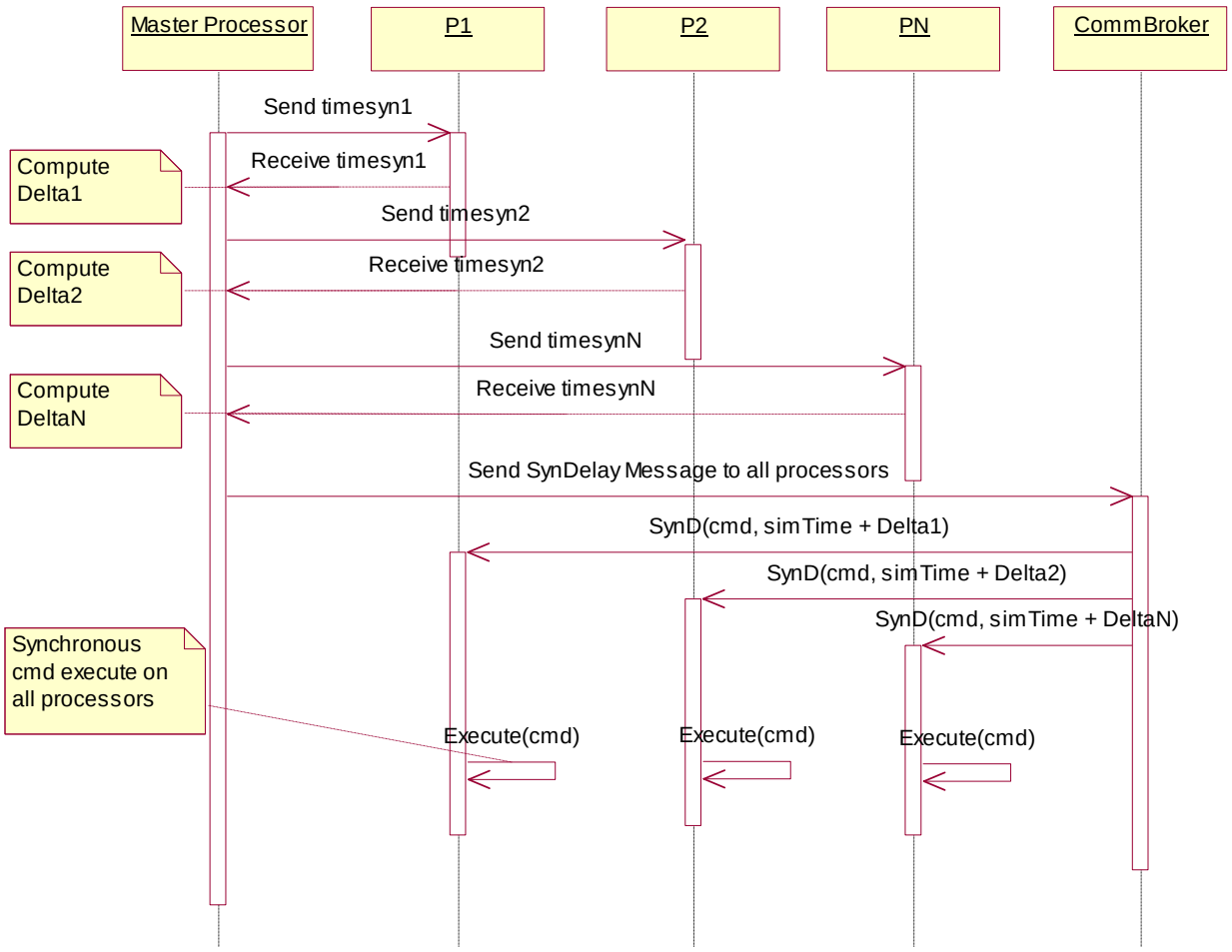


Figure 3.10. Command synchronization in WINDS – sequence diagram.

3.3.3 Parallel Optimizations

One of the main drawbacks of 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 to 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 certain 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 large number 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 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, where it is then restored in memory. 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.

Table 3.1. Simulation control parameters for a cluster-based WINDS.

Process / Setting	**Value**
Master-to-Processor simulation command propagation time	< 10 ms
Synchronous command broadcast to all processors (any number)	< 500 ms
Time to instantiate simulation object on remote processor (from template)	< 20 ms
Time to send object across network (via serialization)	< 40 ms
Typical simulation synchronization delay	1000 ms
Typical simulation clock cycle	50 ms
Time-scaled simulation multiplier (variable, dependent on architecture)	20x
Time to update GUI	< 100 ms

Table 3.1 lists the procedural performance numbers and typical simulation settings for the WINDS framework running on a cluster computer.

To minimize the overhead of exchanging large number of packets with small payload, we utilize the advantage of Nagle algorithm. Nagle algorithm is used to automatically concatenate a number of small buffer messages; this process (called *nagling*) increases the efficiency of a network application system by decreasing the number of packets that must be sent. For instance, a single byte of data originating from

one simulator component could result in the transmission of a 41 byte packet consisting of one byte of useful information and 40 bytes of header data. This translates into a large communication overhead. Nagle's algorithm instructs the sender to buffer data if any unacknowledged data is outstanding. Any data sent subsequently is held until the outstanding data is acknowledged (ACKed) or until there is a full packet's worth of data to send. Our simulation framework accumulates packets generated by each processor into a queue, which is then sent at once, before the simulation clock cycle advances (e.g., every 100 ms). Since the estimated time to send a certain-sized packed is known in advance, when the queue reaches an established threshold size, the packet is generated and sent out. As amount of inter-processor communications increases, two or more packets could be sent out before the expiration of the current clock cycle. Once the packet arrives to the destination processor, the events from the queue are individually analyzed and inserted in the Event Handler’s queue in an appropriate location based on a time stamp supplied. This mechanism facilitates elimination of out-of-order delivery of simulation packets and events embedded in these packets. For simulation control connections (e.g., simulation clock dissemination), nagling is disabled for all control sockets by specifying TCP_NODELAY option. Figure 3.11 shows an example of how nagling affects simulation control, when creating new user objects on multiple processors.

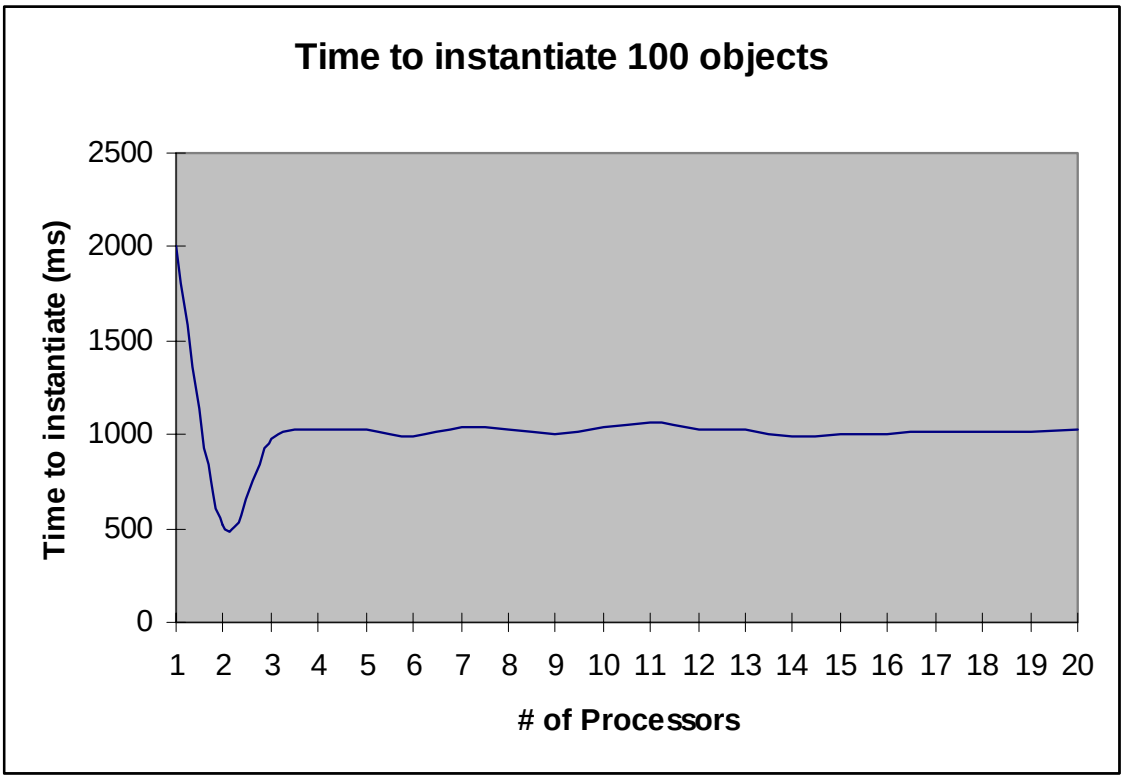


Figure 3.11. Effect of nagling on simulation control.

3.3.4 Serialization of Simulation Objects and Load Balancing

Originally, the Round-robin mechanism of simulation object distribution is used to allocate user simulation objects to the processors within the cluster. It does not provide the optimal assignment of nodes to processors, resulting in substantial inter-object communication overhead. We have studied the effects of various ways to distribute and organize simulation object hierarchies. The final layout of the simulation framework is still a client-server model, with a server residing on one cluster node, providing centralized time management and network routing for the simulation framework. Most heavy processing of data is still performed on individual processors, and the bottleneck

effect of the central node is negligible, when taking into considerations the high load of each individual processor. To reduce network traffic, we have implemented a serialization scheme to allow user objects to migrate from one node to the other throughout the simulation execution. The central node monitors the network traffic to detect user objects that produce frequent inter-node network communication. The simulation operator then has an option to move a particular user object across the network from one processor to the other, to minimize amount of inter-object communications.

In the case of a network with low node mobility, initial object assignment to processors may be optimized by deciding on the number of processors supporting the simulation and specifying each nodes' coordinates in accordance with the current node's expected processor assignment. For instance, if we utilize 4 processors (P_1, P_2, P_3 and P_4) in our simulation, we can geographically split the simulation area into 4 parts (A, B, C and D). Then, given the low mobility of user objects, we can minimize the inter-processor communication during simulation execution by instantiating, for example, every object 1, 5, 9, … with initial coordinates placing it in region A, objects 2, 6, 10, … placed in region B, etc. The Round-robin object allocation mechanism will then instantiate all objects in region A at processor P_1, all objects in region B at P_2, etc.

The object serialization is implemented via BOB (Better Object Binder) XML serialization library. An object is serialized at one processor, then is sent by the simulation communication broker to another processor, where it is de-serialized, using the following code:

```
// Serializing an object into XML representation:
new org.lucci.bob.DataBinder.XML().serialize(obj, outStream);

// Deserializing an object on another processor:
Object obj = new org.lucci.bob.DataBinder.XML().deserialize(inStream);
```

Listing 3.1. Serializing and De-serializing User Object.

Using BOB library allows the user to save object state in XML format and visually examine it if necessary. The library doesn't require implementing serializable interface in user objects, this making it less complicated for the user to create simulation objects.

To study the benefits of object serialization and its effects on load-balancing network traffic in distributed simulations, we will consider an algorithm from section 5.4 – load balancing packet distribution for the purpose of monitoring by an intrusion detection system. While the algorithm is described in detail in chapter 5, the main idea is to partition the wireless ad hoc network into clusters, then allocate the main control node (cluster head) to collect network packets. The packets are then split into batches and forwarded to other members of the same cluster for processing. If the wireless nodes are located on the same processor and have a property of low mobility, simulation network bandwidth can be saved by relocating simulation objects within a cluster onto the same processor. From table 3.1 and experimental results, the average time to send a batch of packets for processing to another user object located on a different processor is 5 ms. The operation of serializing an object and sending it across the network is 40 ms. The distributed intrusion detection algorithm achieves best performance (in terms of quality

of intrusion detection) when many helper nodes are used (see figure 5.8). As the number of nodes within a cluster is 5 on average, we can directly benefit by moving cluster members to the same processor. For the experiment, a simulation was created with a fixed number of nodes, and executed on a variable number of processors. Each graph in figure 3.12 shows results for low (uniformly distributed, with a maximum of < 0.1% of diagonal size of simulation area per second), and high (uniformly distributed, with a maximum of < 1% of diagonal size of simulation area per second) mobility of nodes. If two simulation nodes are located on the same processor, the time to forward a batch of packets (average size 10 packets per batch) between the nodes is assumed to be 0. All nodes are instantiated with a geographic distribution that initially allocates neighboring nodes to the same processor (as described at the beginning of this section), minimizing serialization efforts at the beginning of the simulation. Number of nodes is fixed to be 200 throughout the simulation. Network traffic is Poisson-distributed, with min and max rates of 8 and 92 packets per second, respectively. From the graph, we see that in case of low node mobility, serialization process significantly reduces the amount of traffic between the processors, compared to regular simulation execution. However, in the scenario where the simulation objects are highly mobile, the object serialization mechanism yields an increase in the inter-processor network traffic. We can infer from these results that a careful analysis of simulation execution is required, prior to utilizing object serialization mechanism.

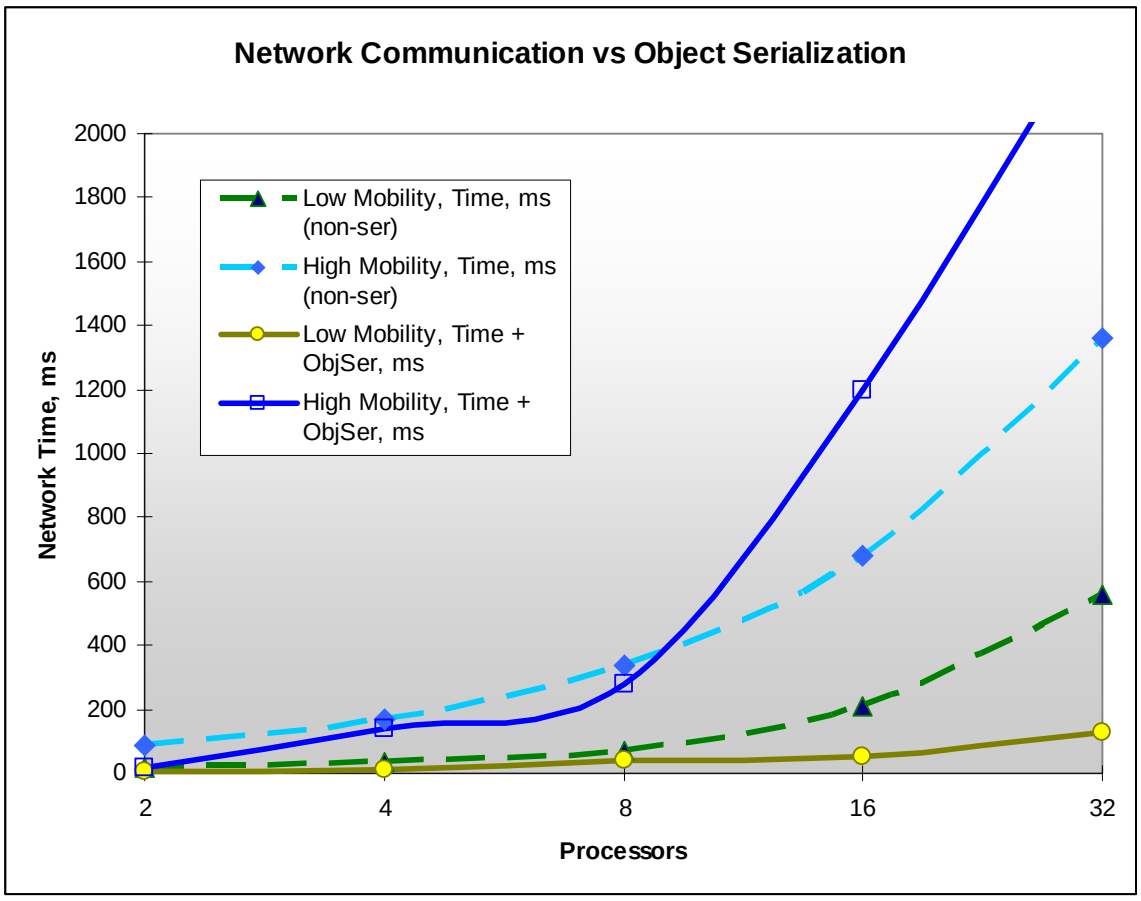


Figure 3.12. Using simulation object serialization for distributed DID problem.

3.4 Summary

In this chapter we have described the PINS architecture for parallel interactive network simulations. General framework design is presented, and can be used as a platform of extending the architecture to specific network simulation applications. The framework is cross-platform, scalable, and easily adjustable to particular simulation requirements. A

specific implementation of PINS for simulating large-scale wireless networks, called WINDS – is also presented. WINDS framework has been used in our research on the agent-based ad hoc network intrusion detection system, and later – as a research and development tool that incorporates a flexible test-bed targeting simulations for a wide variety of wireless network applications. WINDS is a generic wireless network simulator for a variety of wireless communication infrastructures, such as wireless networks, cellular networks, ad hoc networks, sensor networks, etc. We demonstrate the use of WINDS for several specific applications, such as an intrusion detection system simulation in wireless ad hoc networks. We have extended WINDS implementation to the multiprocessor environments for a transparent execution on a symmetric multiprocessor cluster computer, and a network of LAN-interconnected computers, and utilized inter-processor clock synchronization algorithms and object serialization mechanism, resulting in a substantially-increased scalability and performance of large wireless network simulations.

CHAPTER 4: CASE STUDY – WINDS SIMULATION

This chapter describes the process of designing and executing a simulation on our distributed simulation system, giving studies of efficiency and constraints of simulating wireless network systems and applications. A particular wireless infrastructure, called a totally-mobile wireless network, is presented in this study. Step-by-step simulation development process is presented, together with study objectives, simulation procedure, and interpretation of results.

4.1 Simulation of MBS Mobility in a Totally-Mobile Wireless Network

The concept of a totally-mobile wireless network was developed at the University of Central Florida [7]. It represents a crossover between infrastructure wireless and ad hoc networks – an infrastructure, where the transmitting base stations are mobile – mounted on mobile platforms and can be moved to a specific location to optimize wireless coverage offered. This scenario is of interest to the groups utilizing in-the-field communications, and similar entities requiring high level of fault-tolerance and redundancy, with an added benefit of improving wireless coverage (as shown in Fig. 4.1).

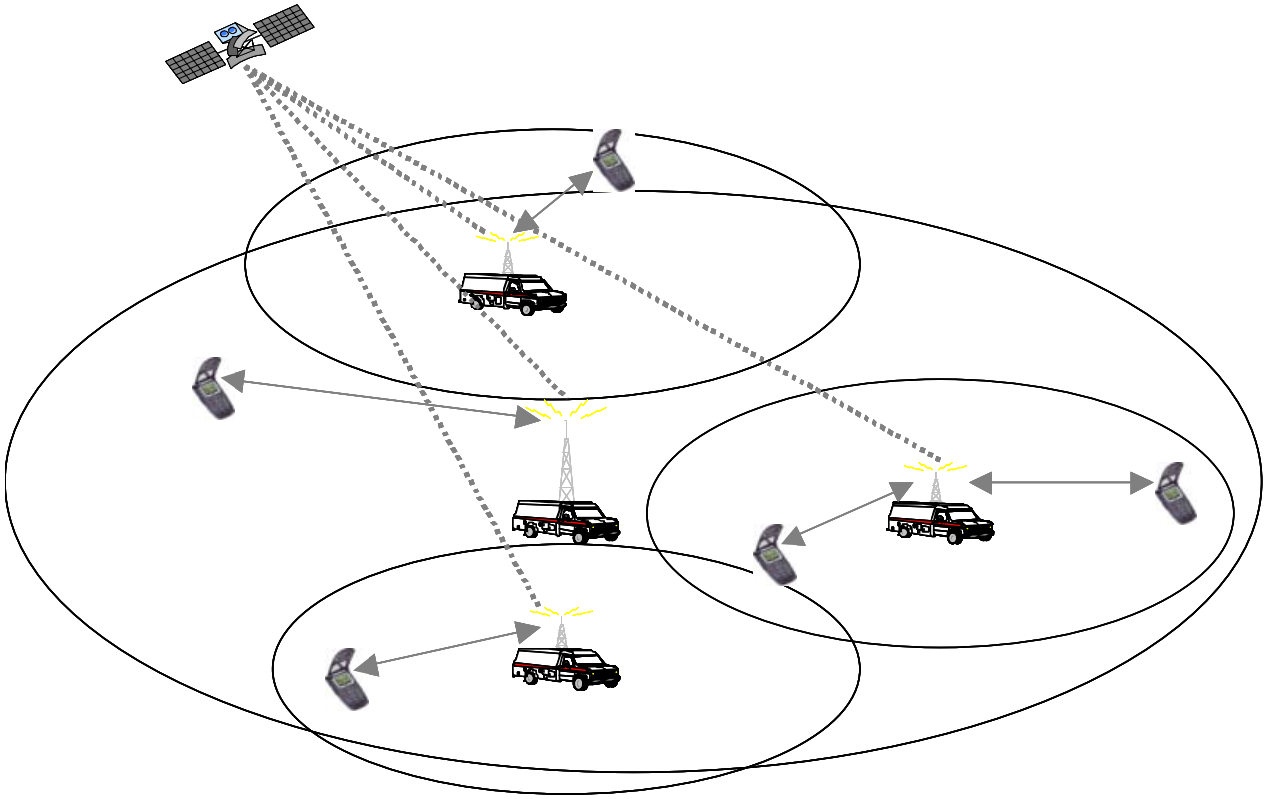


Figure 4.1. Totally-mobile wireless network concept.

Several fault-tolerant designs have been considered, and various experiments were conducted to decide on the framework that provides high level of redundancy, yet is economically efficient [7]. The architecture was devised, where active mobile base stations would be supplemented by auxiliary base stations positioned in the vicinity of the respective base station. If an active base station ceases to function, all wireless connections are switched to the auxiliary base station that will re-position itself to optimize wireless coverage and preserve communication links of existing clients. Furthermore, umbrella coverage is provided by a mobile base station with a larger area of wireless coverage. The base station providing this coverage will frequently re-position itself to accommodate wireless clients outside the scope of coverage by the smaller base

stations. Multiple positioning algorithms have been devised to optimize the coverage area of the entire wireless network under study. Some are based on signal strength; others assume the knowledge of mobile node positions, where a global positioning system (GPS) provides precise user coordinates to the base station, allowing it to make calculated decisions on its own positioning. For the purpose of this simulation, we will consider *Center-of-Gravity* algorithm for positioning mobile base stations in this section and compare it with the average-case scenario of a uniformly-distributed group of communicating nodes.

4.1.1 Objectives

- To simulate the mobile base station center-of-gravity positioning algorithm for optimal wireless coverage for the totally-mobile wireless network model.
- To compare the mobile MBS coverage with the stationary base station scenario, in the case of uniform density of mobile node distribution.

4.1.2 Simulation Design

The simulation *pre-conditions* are as follows:

1. Wireless node mobility is governed by Random Waypoint mobility pattern with zero waypoint wait time.
2. Mobile base station mobility is governed by Center-of-Gravity motion vector mobility pattern.

3. The ratio of each MBS signal area coverage to the total simulation area is 1:12.
4. The ratio of umbrella MBS signal area coverage to the total simulation area is 1:6.
5. Number of mobile base stations is varied from 1 to 8 for the same simulation area.
6. Channel capacity of each MBS is assumed unlimited.
7. Network density (number of nodes per unit area) is varied from 10 to 100, and the average coverage data is computed at the end of simulation.

The simulation proceeds according to the following scenario:

1. Wireless nodes are created and communication parameters initialized.
2. Mobile base stations (MBS) and an Umbrella base station (UBS) are created.
3. Each wireless node is assigned to a MBS, and the rest of mobile nodes within the coverage area of UBS are assigned to the UBS.
4. Each wireless node is mobile throughout the simulation.
5. Each node broadcasts its position to all MBS in vicinity.
6. Each node has an ownership index, indicating which MBS it is covered by.
7. Each MBS and a UBS is mobile throughout the simulation.
8. Each MBS is assigned to a group of nodes – but services all other wireless nodes within its vicinity up to capacity.
9. Each MBS moves towards the center-of-gravity of the assigned group of nodes, computing its position based on the coordinates of each wireless node within a group.
10. A UBS moves towards the center-of-gravity of the unassigned group of nodes.

11. Network density (number of nodes per unit area) is varied from 10 to 100.
12. Number of MBS is varied 1 to 8, and results are recorded for each setting.
13. Simulation executes for 10 minutes (wall time) at each setting of MBS number.
14. Simulation clock is set to 100ms (10x faster-than-real time execution).
15. Simulation stops after the final run has been completed, at the number of MBS of 8.

The only communications in the simulation are the messages from mobile nodes with their positioning information sent to MBSs, and the ownership update messages sent from MBSs to wireless nodes. The results are compared with the average-case scenario (not described in this section, but is a separate simulation), where wireless nodes are moving randomly, and the base stations have fixed positions and are uniformly distributed throughout simulation area.

Table 4.1. Simulation objects for ad hoc wireless network clustering simulation.

Object	Properties
1. Wireless node (wNode)	1. Mobility: Random Waypoint algorithm 2. Send(): send packet method 3. Receive(): receive packet method 4. myMBS: integer – associated MBS index; 0 if not associated 5. myPos: XY – current node coordinates 6. cntAssociated: long integer – total number of sampled times, when a node was covered by MBS (samples taken every 10 simulation clock cycles) 7. cntDisAssociated: total number of sampled times, when a node was not covered by MBS (samples taken every 10 simulation clock cycles)
2. Mobile Base Station (mStation)	1. Mobility: Center-of-Gravity motion vector algorithm 2. CommRange: integer – MBS's communication range in simulation distance units (e.g., meters) 3. Send(): send packet method 4. Receive(): receive packet method 5. ComputeToVector(): compute motion vector method 6. coveredNodes[]: XY-array – set of all covered wireless nodes and their positions 7. myPos: XY – current MBS coordinates 8. toPos: XY – calculated motion vector

The following figure 4.2 shows an activity diagram of a Wireless Node object activities throughout the simulation, and figure 4.3 shows the activities of a Mobile Base Station.

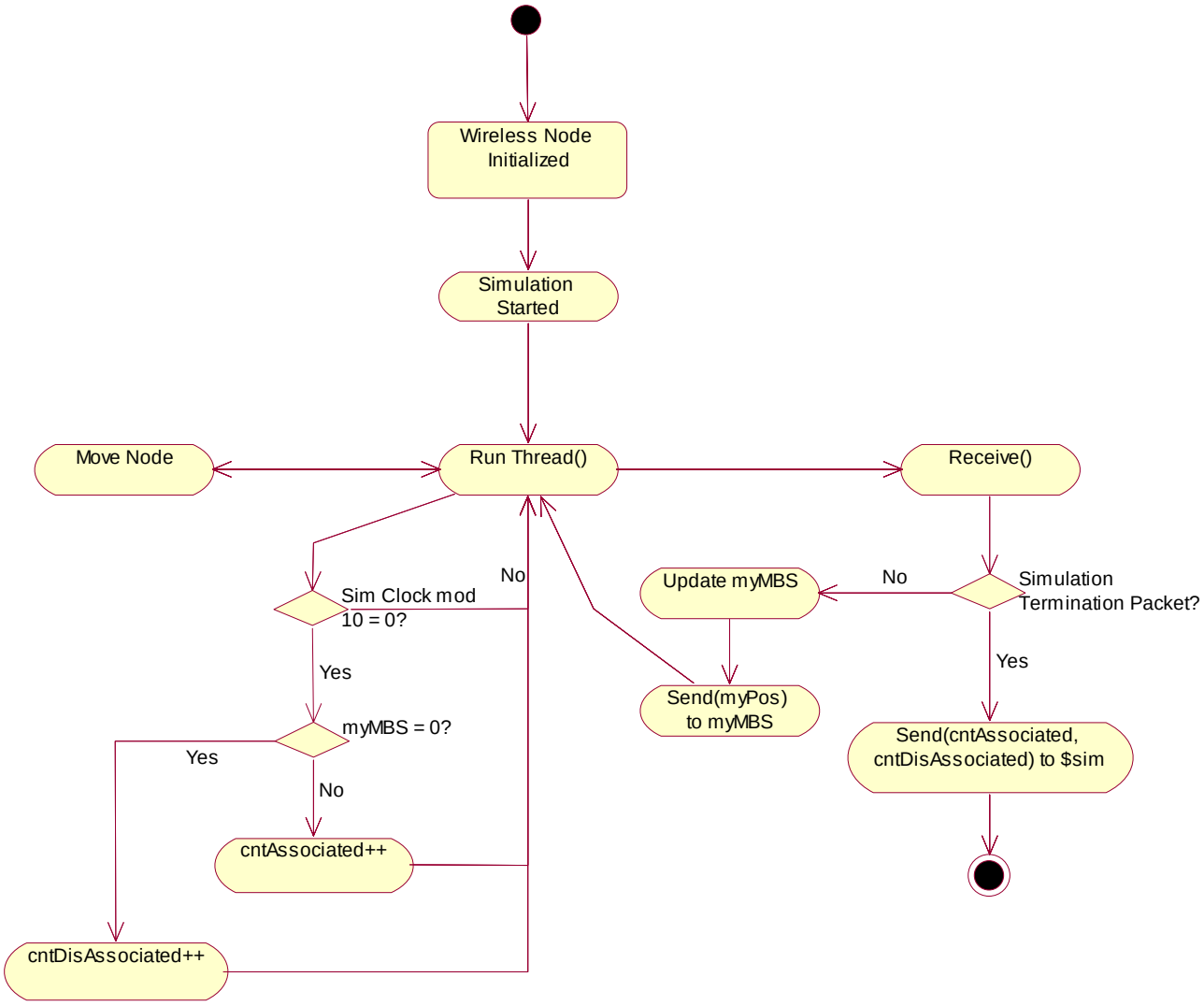


Figure 4.2. Wireless node in a totally-mobile network activity diagram.

In figure 4.2, when a node receives a packet, the source of the packet is always a MBS. Due to the absence of network communication other than for the purpose of MBS assignment, the node parses the incoming packet and gets assigned to a new MBS, sending back the node's current position.

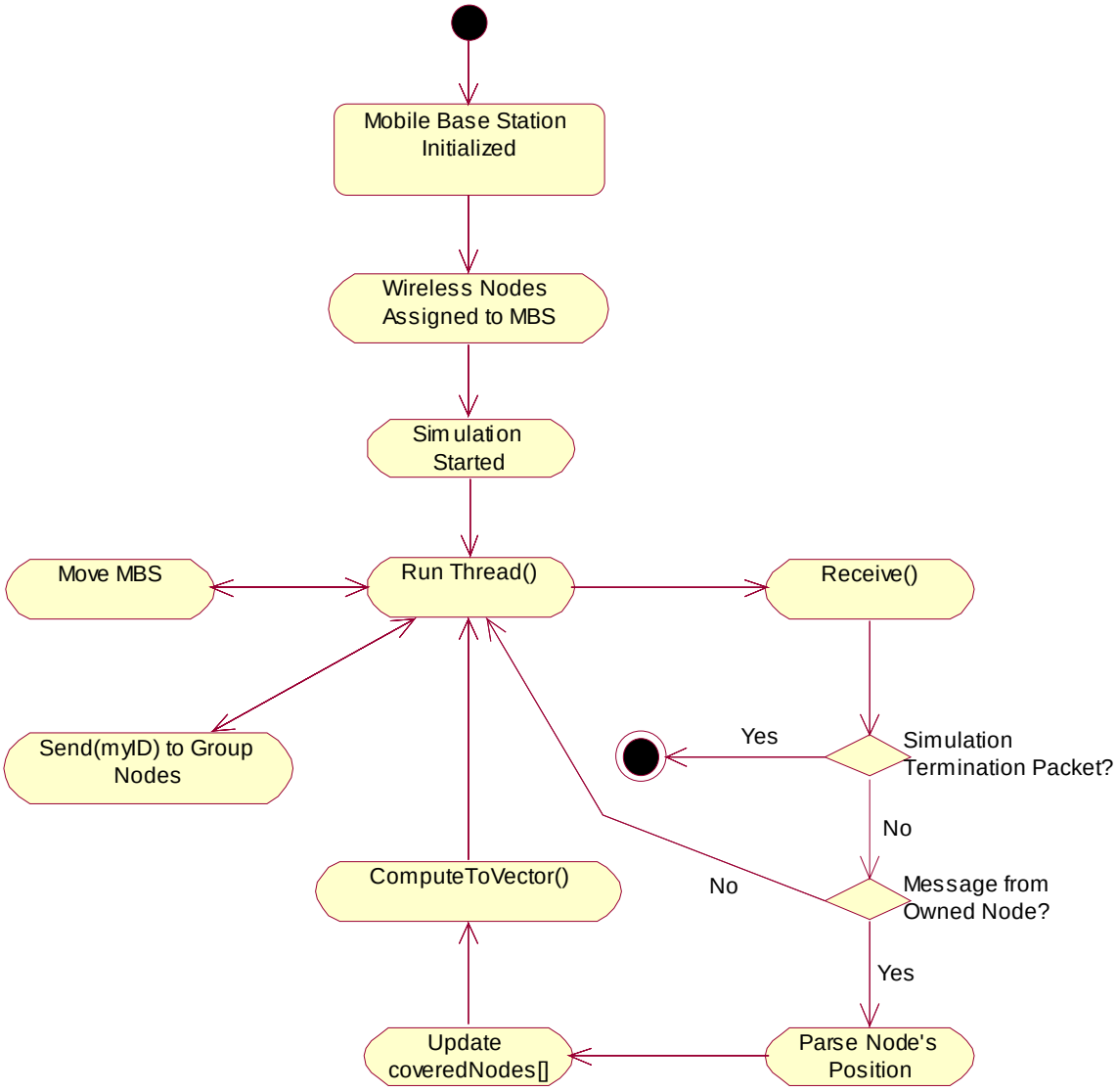


Figure 4.3. MBS in a totally-mobile network activity diagram.

Simulation framework traffic routing module determines what wireless nodes are within communication range, based on user-supplied *CommRange* property of the MBS.

The wireless node and mobile base station objects are implemented as Java classes following the WINDS object design template, and have the structure (in pseudo code) depicted in the following listings.

```
// Wireless Node class
// Object parameters and default parameter values:

//#P str DefaultObjectNamePrefix, int initXPos, int initYPos, bool
isMobile

//#D wNode, 0, 0, true


public class wnode {

  int ID;

  String objName;

  simEvtHandler ParentClass;

// Mobility study parameters

  int cntAssociated = 0;

  int cntDisAssociated = 0;

  int myMBS = 0;

  XY myPos = param(initXPos, initYPos);


// Methods

  wnode() // Wireless node object constructor - initialization

  void Tick() // Simulation clock pulse – every 10th clock tick here,
              // update the cntAssociated and cndDisAssociated

  void Receive(Long simClock, String dataType, String strPayload)

  void Send(Long simClock, String dataType, String strPayload)

  void wMove() // Wireless node random waypoint mobility


}
```

Listing 4.1. Wireless node in a totally-mobile network Class.

```
// Mobile Base Station class
// Object parameters and default parameter values:

//#P str DefaultObjectNamePrefix, int initXPos, int initYPos, bool
isMobile, bool isUBS, int myID

//#D mStation, 0, 0, true, false, 1


public class mStation {

  int ID;

  String objName;

  simEvtHandler ParentClass;

// Mobility study parameters

  int myID = param(myID);

  XY myPos = param(initXPos, initYPos);

  XY toPos;

  XY coveredNodes[];

// Methods

  mStation() // Mobile base station object constructor - initialization

  void Tick() // Simulation clock pulse – calls mbsMove

  void Receive(Long simClock, String dataType, String strPayload)

  void Send(Long simClock, String dataType, String strPayload)

  void mbsMove() // Mobile base station center-of-gravity mobility

  XY ComputeToVector() // Computes new MBS motion vector based on
                       // the center-of-gravity mobility model

}
```

Listing 4.2. Mobile base station in a totally-mobile network Class.

4.1.3 Simulation Execution and Results

The *wNode* and *mStation* classes were placed in the *simulation root/objects* directory, then dynamically loaded by the simulation framework. The simulation results are sent to the central *Simulation Statistics* window of the user interface, and are saved into a CSV file for analysis. Figure 4.4 shows totally-mobile wireless network simulation in progress.

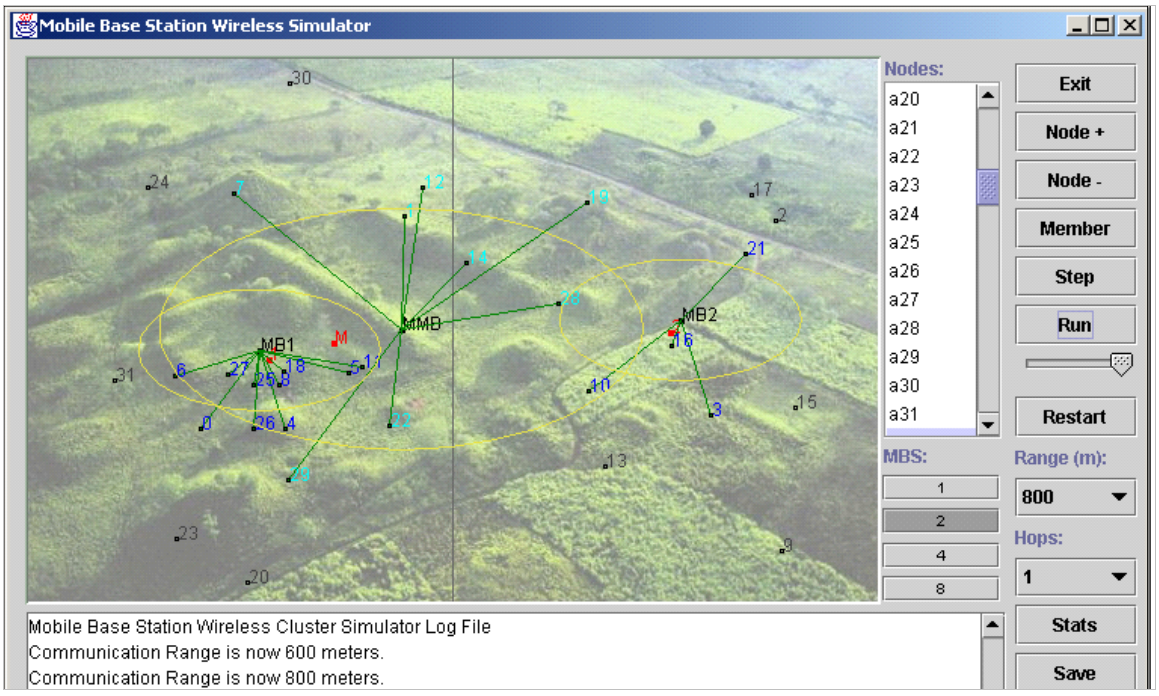


Figure 4.4. Totally-mobile network simulation in progress.

Wireless nodes are numbered and connections between the nodes and the base stations denote communication links within range. Blue nodes indicate that a node has been assigned to the mobile base station, cyan – a node is assigned to an umbrella base station and gray – node is not connected to the wireless network. Mobile base stations are

labeled as MBn, and a motion vector computed according to center-of-gravity algorithm is shown in red for each base station. Upon simulation execution, the following results were obtained.

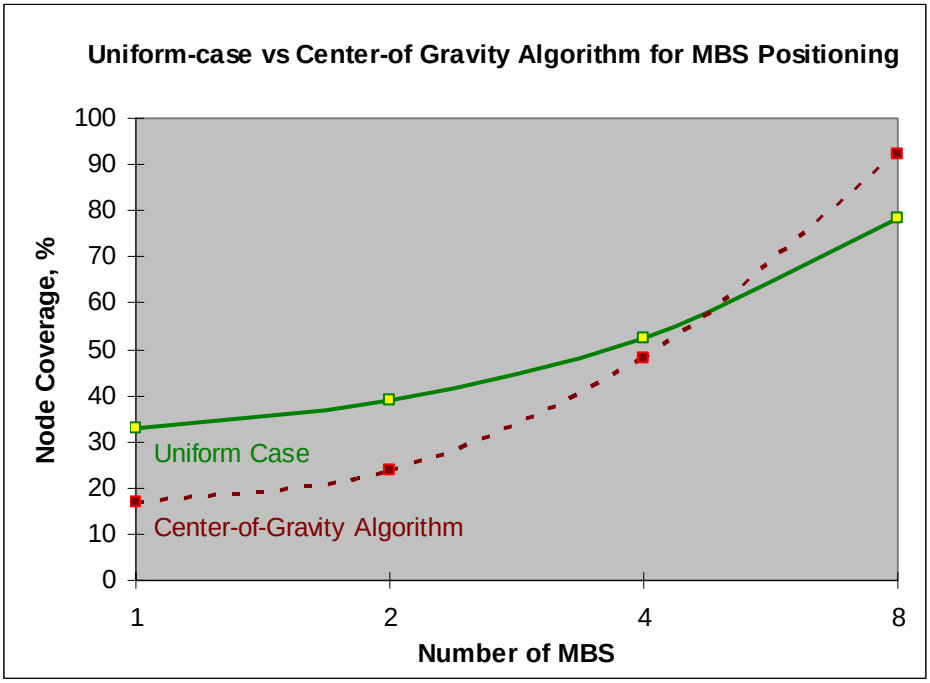


Figure 4.5. Network coverage of wireless nodes in a totally-mobile network simulation.

Figure 4.5 demonstrates that applying center-of-gravity positioning policy to mobile base stations yields worse-than-expected network coverage results, in addition to the base station repositioning requirement (an average of results at various network density levels is presented here). However, as we increase the density of the wireless network, as well as the number of mobile routers supporting the network, the overall network coverage is improved. The results demonstrate that a simple center-of-gravity algorithm for positioning mobile routers is far not the best possible solution. It can, however, be used in cases where wireless clients move in closely-congregated groups. Such a use would be of

primary interest to the military communications, tours where mobile clients are used for guidance and informational purposes, and sensor networks.

4.2 Summary

In this chapter we have demonstrated the use of WINDS distributed simulation framework for a specific scenario of parallel-simulating a totally-mobile wireless network, and specifically a mobility pattern study for such an infrastructure. This chapter may serve as a template for designing parallel wireless simulations, including simulation objectives, pre-conditions, design of the simulation execution, a choice of statistics, and interpretation of results. The entire process from simulation design to execution and result analysis is very concise, due to hierarchical object-oriented nature of the simulation framework and the high interactivity and flexibility of the user interface. Another study is presented in chapter 5, in the context of a wireless network intrusion detection system. In the following chapter we describe our work on an IDS framework for wireless ad hoc networks, and use WINDS simulation framework to develop the IDS-associated algorithms.

CHAPTER 5: INTRUSION DETECTION FOR WIRELESS AD HOC NETWORKS

Ad hoc wireless networks are a widely utilized type of wireless networks today. More than any other network topology, they are vulnerable to intrusions, as they operate in an open medium and use cooperative strategies for network management and communications. In this chapter we summarize our current research on a distributed intrusion detection framework for ad hoc wireless networks based on mobile agent technology. Intrusion detection processing is minimized using a real-time clustering algorithm and various load balancing strategies, while maintaining high degree of intrusion detection accuracy. A case-based reasoning approach to our network-level intrusion detection engine incorporates sophisticated artificial intelligence techniques that help overcome some of the limitations of other rule-based intrusion detection systems. In contrast to many intrusion detection systems designed for wired networks, we develop an efficient and bandwidth-conscious framework that targets intrusions at multiple levels and takes into account distributed nature of ad hoc wireless network management and decision policies.

5.1 Rule-Based Intrusion Detection

With rapid development of wireless network applications, security became one of the major problems that wireless networks face today [62]. While firewalls may prove to be an efficient first line of defense in wired networks, this is certainly not the case in the wireless world. Wireless transmissions are subject to eavesdropping and signal jamming. Physical security of each node is important to maintain integral security of the entire network. Ad hoc wireless networks are totally dependent on collective participation of all nodes in routing of information through the network. These are some of the major problems that wireless networks face today. As the uses of such networks grow, users will demand secure yet efficient, low-latency communications.

Intrusion detection is one of key techniques behind protecting a network against intruders. Existing intrusion detection systems and techniques have been reviewed in chapter 2. In this chapter, we will concentrate our presentation on ad hoc wireless networks. Ad hoc wireless network is a collection of mobile nodes that establish a communication protocol dynamically. The nodes may join the network at any time and communicate with entire network via the neighboring nodes. There are no base stations, and each member of such a network is responsible for accurate routing of information, and takes part in routing decisions. Due to arbitrary physical configuration of an ad hoc network, there is no central decision making mechanism of any kind – rather, the network employs distributed mechanisms of coordination and management. What really makes a difference between fixed wired and mobile wireless networks is the fact that mobile

nodes have a very limited bandwidth and battery power. Network packet monitoring is performed at gateways in a fixed network, but a concept of a gateway in a wireless network is very vague, depending on the type of network and routing algorithms used. Efficient host-based monitoring requires large amounts of CPU processing power, and hence is energy consuming.

Our work on intrusion detection system for ad hoc wireless networks takes into account the above considerations to provide a lightweight, low-overhead mechanism based on mobile security agent concept, described in chapter 2. Agents are dynamically updateable, lightweight, have task-specific functionality and can be viewed as components of a flexible and dynamically configurable IDS. These qualities make them a choice for security framework in bandwidth and computation-sensitive wireless ad hoc networks. We utilize mobile agents at several intrusion-monitoring levels and process 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. The other 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. We also propose 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 (discussed in the next section), and incorporated case-based reasoning engine for detecting intrusions at the packet level in our modular IDS system.

5.1.1 Case-Based Reasoning Systems

Case-based reasoning systems are designed for a given application domain. It is possible to abstract out the common aspects of CBR from the domain specific aspects [69]. This leads to a generic case-based reasoner from which any arbitrary domain-specific case-based reasoner can be created as a specific instance. In order to build the desired generic case-based reasoner, it is required to generalize both the notion of a case and the notion of a similarity metric used for determining the degree of similarity between cases. Since cases are described by their features, the first task is to describe the generic notion of a case feature. The framework proposed in [69] makes it possible to define virtually any type of feature and any type of case. The similarity metric for cases can be defined as a collection of feature comparison results with a rule specifying how these intermediate results are combined. Moreover, feature comparisons can be generalized into the generic notion of feature comparator of which each specific comparator is an instance. Although each case feature may require a different type of comparison, the result of the comparison should be a similarity assessment between the same case feature of the problem specification and of a case from the case archive.

Therefore for each new type of case feature, it is required to define a comparator that determines the degree of similarity between the problem situation feature and the corresponding feature in the case archive. Since different case features may use the same comparator, the number of comparators in the system will be much smaller than the number of different features. The modular distinction between comparators and case features simplifies the adaptation of the system into different problem domains.

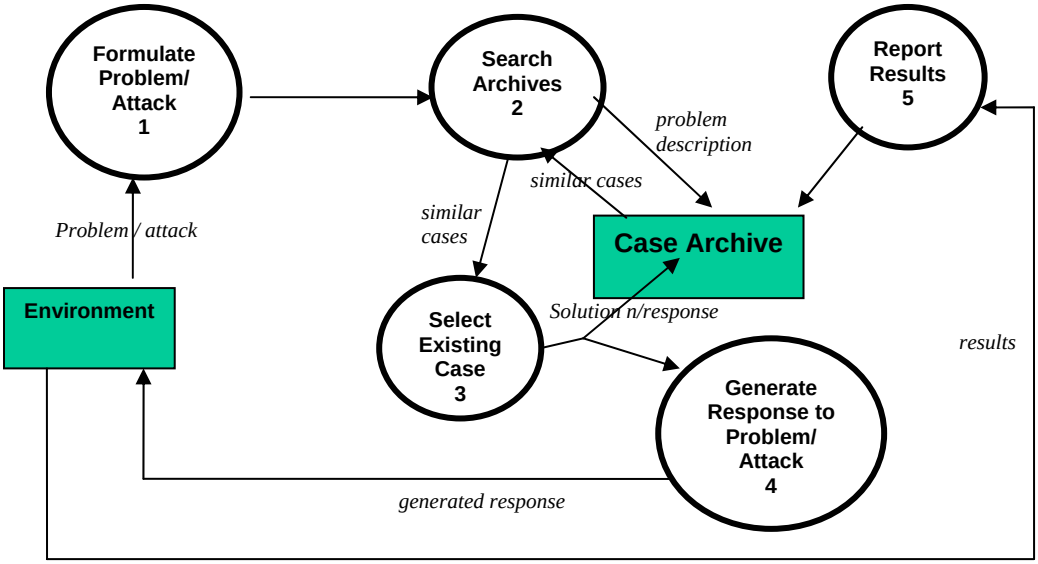


Figure 5.1. Case-based Reasoning Process.

The basic features of a general case-based reasoning (CBR) system are depicted in figure 5.1. The most important component of the system is the case archive where the previously experienced problems are stored with their solutions. Each entry in the case archive is called a "case" which contains (i) the features describing the problem, and (ii) the action or actions that were taken to solve the problem. When a problem is detected in the surrounding environment, it is formulated as a set of case features (step 1). Then, this

problem description is transferred to a search engine that extracts the similar cases from the case archive, where similarity is measured by the similarity between the matching features of the problem description and the case features of actual cases in the case archive (step 2). The returned cases are ranked according to their degrees of similarity to the given problem. At this moment two different scenarios are possible: either some of the selected cases are decided as a solution to the problem or a new case is formulated to solve the problem based on the returned cases. In either case, the actions recommended by the returned case or cases are taken (step 4). Furthermore, the measure of success or failure of the result of the action is reported along with the case into the case archive (step 5). This information is taken into account in the similar-case extraction process so that the performance of the system improves over time.

5.2 Network Intrusion Detection System for MANETs

This section introduces our multi-sensor intrusion detection system employing cooperative detection algorithm. A mobile agent implementation is chosen, to support such features of the IDS system as mobility of sensors, intelligent routing of intrusion data throughout the network and lightweight implementation.

5.2.1 Modular IDS Architecture

The proposed Intrusion Detection System (IDS) [41] is built on a mobile agent framework. It is a non-monolithic system and employs several agent types that perform specific functions, such as:

- *Network monitoring:* Only certain nodes will have sensor agents for network packet monitoring, since we are interested in preserving total computational power and battery power of mobile hosts. Network monitoring agents feature misuse detection engine (described in the following section).
- *Host monitoring:* Every node on the mobile ad hoc network will be monitored internally by a host-monitoring agent. This includes monitoring system-level and application-level activities for anomalies.
- *Decision-making:* Every node will decide on the intrusion threat level on a host-level basis. Certain nodes will collect intrusion information and make collective decisions about network-level intrusions.
- *Action:* Every node will have an action module that is responsible for resolving intrusion situation on a host (such as locking-out a node, killing a process, etc).

Each module represents a lightweight mobile agent with certain functionality, making a total network load smaller by separating the functional tasks into categories and dedicating an agent to a specific purpose. This way, the workload of a proposed IDS system is distributed among the nodes to minimize the power consumption and IDS-

related processing time by all nodes. A hierarchy of agents has been devised in order to achieve the above goals. Hierarchical IDS systems have been proposed in [20, 9, 34]. However, we will adapt our own hierarchy for our purposes. There are three major agent classes – monitoring, decision-making and action agents. Some are present on all mobile hosts, while others are distributed to only a select group of nodes, as discussed further. Monitoring agent class consists of packet, user, and system monitoring agents. The following diagram shows the hierarchy of agent classes.

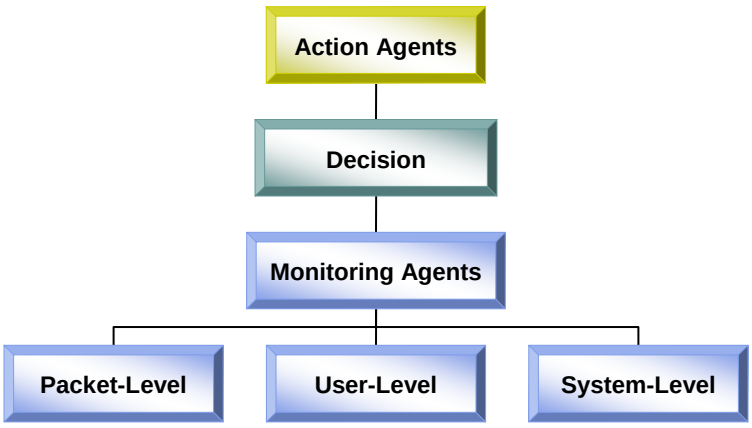


Figure 5.2. Layered Mobile Agent Architecture.

5.2.2 Mobile Agent Distribution across Wireless Ad Hoc Network

As mentioned above, not all the nodes on a wireless ad-hoc network will host all types of IDS agents. To save the resources, some of the functionality must be distributed efficiently to a (small) number of nodes while providing an adequate degree of intrusion detection. While all the nodes accommodate host-based monitoring sensors of an IDS, we

use a distributed algorithm to assign only a few nodes to host sensors that monitor network packets, and agents that make decisions.

The idea is to logically divide a mobile network into clusters (similar to Clustered Gateway Switch Routing protocol described in [61, 14, 58] and used in [80] for authentication purposes) with a single cluster head for each cluster, and to monitor the packets within the cluster by only one node. The algorithm is presented below, along with an example.

Clustered Network-Monitoring Node Selection Algorithm:

1. Hop Selection Step: based on security requirements, a certain number is selected as a number of hops. This step is important in choosing decision agent-hosting nodes, as well as network monitoring nodes, as selected number is the maximum number of hops from any node in the ad-hoc network to the Decision Node. Selection of this number greatly affects the network monitoring range, as only those nodes taking part in a decision process host network monitoring agents, resulting in lesser area of the network being monitored.

2. Let C_i denote the number of established connections (reachable nodes) for node i at the time of cluster setup, with a total of N nodes in the entire network. Each node sends its C_i value to all its reachable neighbors.

3. Upon receiving C_j values from its neighbors j, where $j \neq i$ for all $i = 1 \ldots N$, a node i sums up the total as S_i (connectivity index), which upon completion is broadcast to all nodes with a time to live (TTL) equal the number of hops selected in step (1):

$$S_i = C_i + \sum_j C_j$$

4. Each node then has to vote to select cluster head node, that will accommodate network monitoring and decision agents. Every node sends a vote packet to the node it selects based on highest connectivity index received as a result of a broadcast in step (3). If a node receives a vote from a node with equal S*i* value, it doesn't send a vote to the source node. In case two nodes have equal S*i* values and send votes to each other simultaneously, the node with the largest total of S*i* values sends a "discard vote" message to the other node. This will ensure that the minimal number of nodes is selected for hosting packet-monitoring agents. Note that in step 3, a node will decrease TTL count and broadcast the packet containing S*i* to all its reachable neighbors, resulting in every node receiving the information about the maximum S*i* within the hop distance.

5. Each node that received at least one vote loads and runs Network Monitoring and Decision Agents. Steps (4) and (5) are shown on a diagram 5.3, giving scenarios for (a) one-hop and (b) two-hop ad-hoc wireless networks.

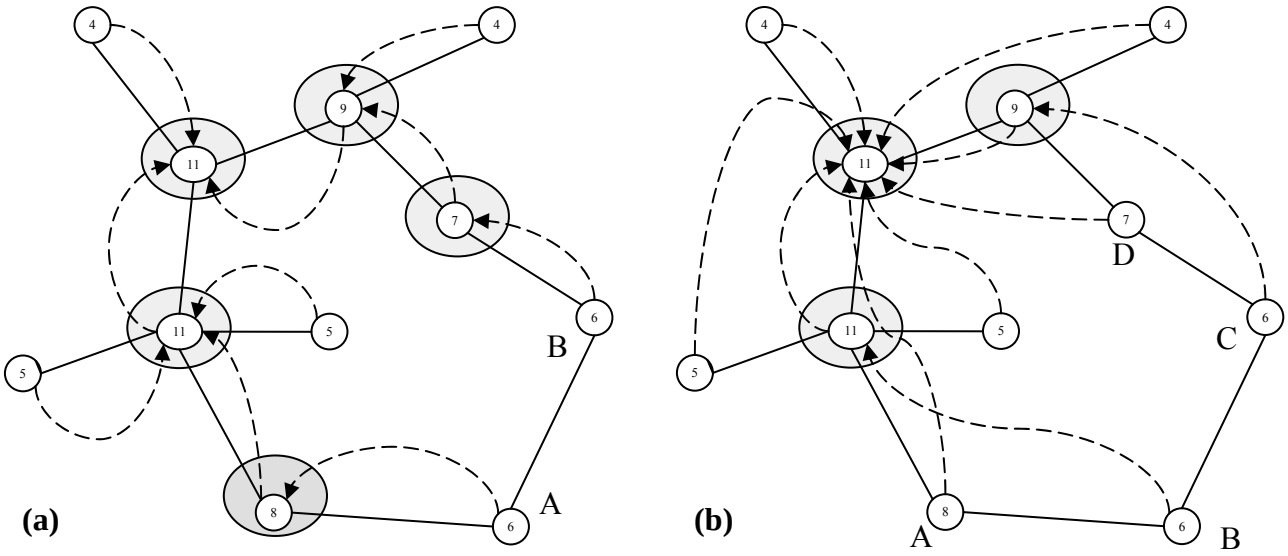


Figure 5.3. Network monitoring node selection with (a) one-hop radius, and (b) two-hop radius. Dashed lines indicate a vote packet route. Nodes selected to host network monitoring and decision agents are highlighted.

The selected nodes host network-monitoring sensors that collect all packets within communication range, and analyze them for known patterns of attacks. Parameters such as per-protocol statistics, number and frequency of certain packet types and consistency with the model are verified. The main advantage of the allocation algorithm above is that overall packet-monitoring task is limited to a small subset of nodes, thus conserving power and processing capabilities for many nodes in the ad hoc network. In contrast, in a monolithic IDS system where each node hosts packet-monitoring module, a message sent from node *B* to node *A* will be received and checked by node *C* as well as node *A*, thus

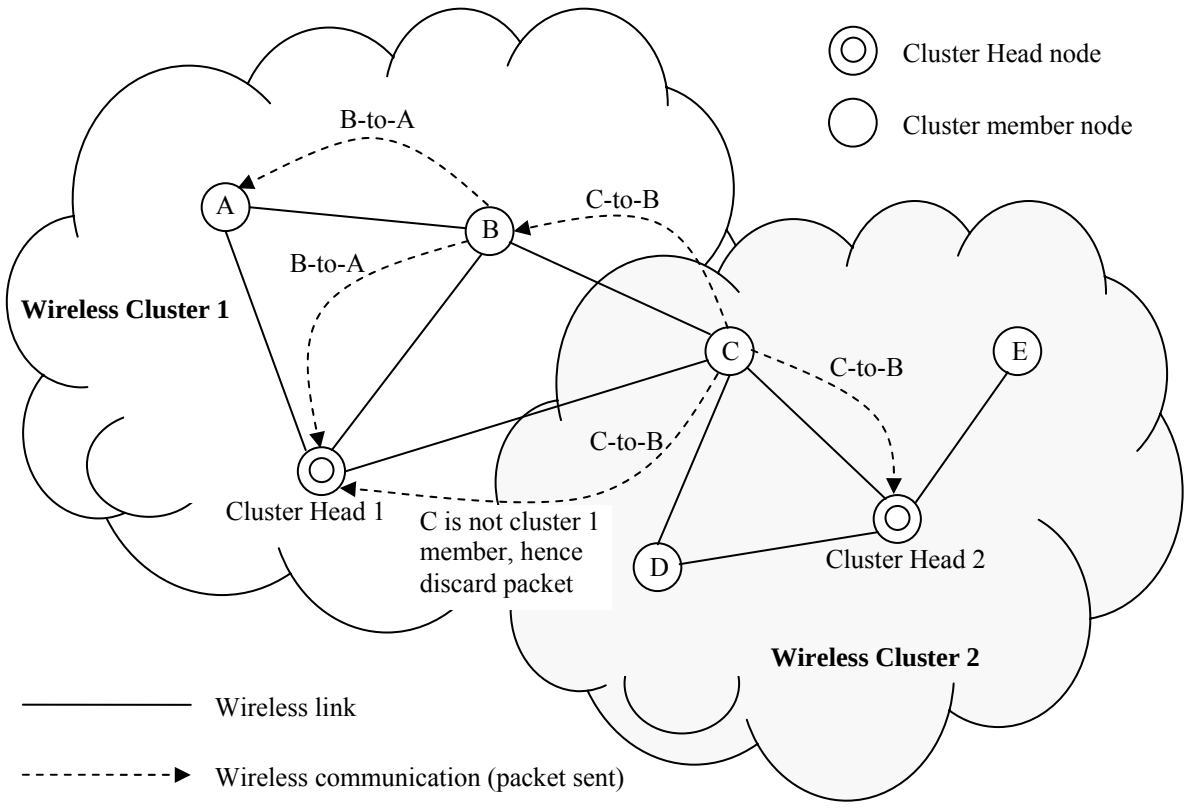


Figure 5.4. Wireless Ad Hoc Network Communications.

introducing redundant processing action. This is visually demonstrated in the figure 5.4. As the physical network arrangement changes, cluster membership is dynamically updated. Figure 5.5 shows a percentage of nodes engaged in network-monitoring activities vs. the total number of the nodes.

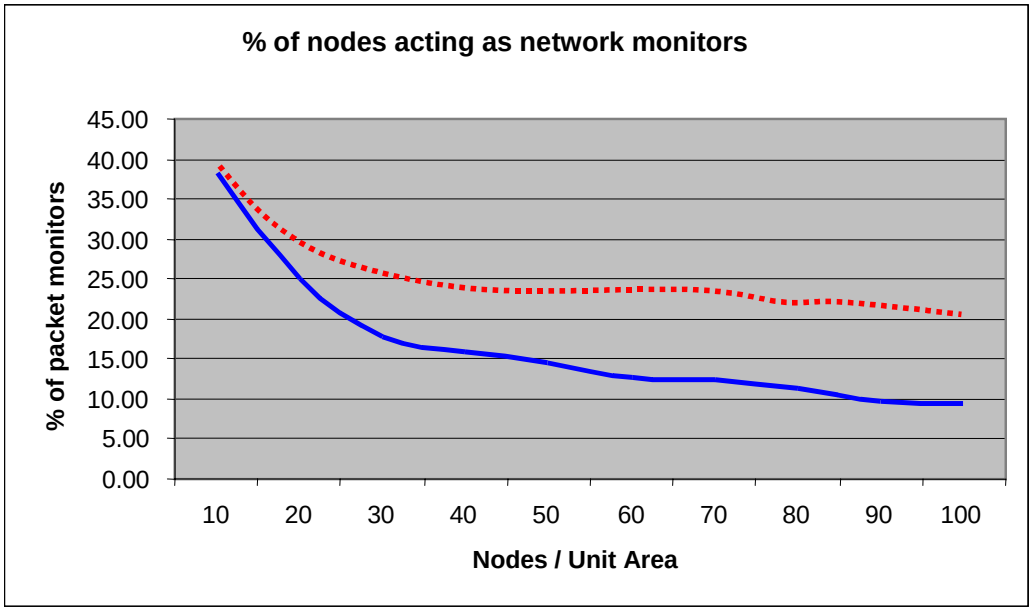


Figure 5.5. Percentage of nodes engaged in packet monitoring in a one-hop (dashed line) and two-hop (solid line) network.

5.2.3 IDS Activity Monitoring Process

As shown in Figure 5.2, monitoring agents are categorized into packet monitoring sensors, user activity sensors and system-level sensors. While packet monitoring is activated only when a node participates in the network (is a member of a cluster), local activity sensors are present on each node and are active all the time. Each sensor performs certain level of monitoring activity and reports anomalies to the decision agents.

Packet-monitoring agents reside on each selected node. In the Figure 5.3 above, we can see that for a case of one-hop cluster, 5 nodes out of a total of 11 nodes host network monitoring sensors, resulting in the entire network being monitored. For instance, a

packet sent from node *A* to node *B* will be received and analyzed by the monitoring node to the left of node *A*. In fact, for a case of one-hop cluster, every node has at least one neighboring node hosting a packet monitoring agent, and thus the entire network is always being monitored. If the system resources are scarce and security requirements can be relaxed, a two-hop system will be more appropriate, as indicated in Figure 5.3(b). Here, we have only 3 hosts dedicated to packet monitoring and decision-making process, saving overall system resources. However, in this scenario, 3 links are not being monitored, which may be acceptable for a highly-dynamic environment, where network configuration changes often. The rationale is that a node is located in close proximity (within two hops) to the packet-monitoring node, and rapid movement may position the node within a communication range of that packet-monitoring node.

Each cluster head monitors packets sent by every member of its cluster, and therefore, the agent subsystem has a low-level access to the underlying operating system's network layer to capture packets that are not intended for the cluster head node. We limit the collection of packets only to those that have as originator any node that belongs to the cluster. This is done to prevent processing of the same packet more than once by any packet-monitoring agent. When packets are captured, they are inserted in a queue (logically), and physically added to a buffer of fixed size (the size depends on the node's available memory and processing capabilities). The packets are then dequeued and processed by the agent's case-based reasoning engine (CBR) for intrusion detection. The mechanism of a CBR engine is described in section 5.1.1. If a packet queue of a cluster

head node becomes full, further packets are dropped until space is available in the queue (see Figure 5.6). By varying queue size, we limit processing done by a cluster head node, as its resources are also used for performing regular user tasks with minimal latency for the user. Agent subsystem is also configured to limit CPU usage by an agent to a certain level, acceptable by the user. To improve the quality of intrusion detection process, a distributed algorithm is applied to prevent large number of packets from being dropped even in traffic-intensive environments. The algorithm is discussed in section 5.4, and is shown to provide a scalable solution to packet-monitoring process with graceful degradation (increased number of dropped packets) in extreme conditions.

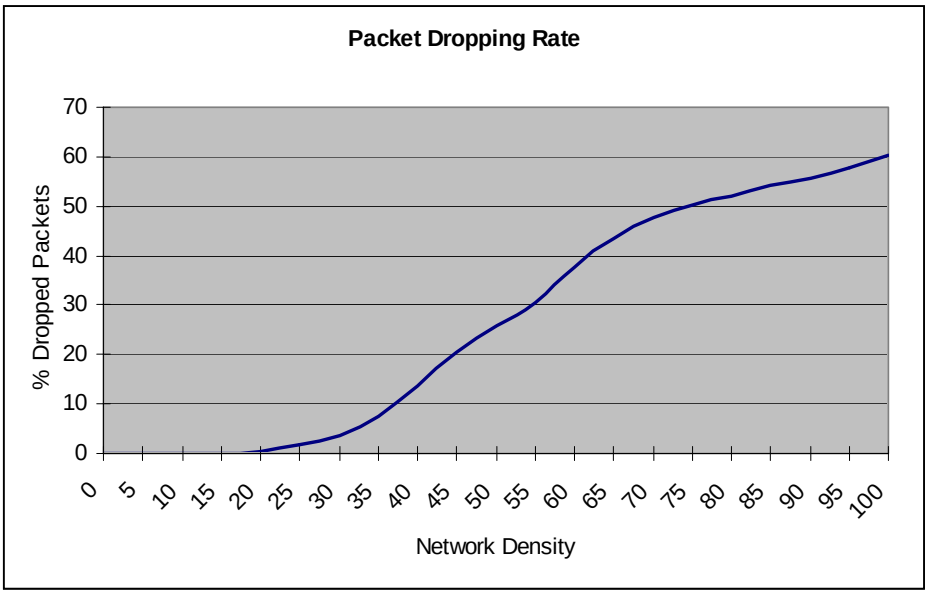


Figure 5.6. Increase in packet dropping rate as the network density increases.

Local detection agents are located on each node of an ad-hoc network, and act as user-level and system-level anomaly-based monitoring sensors. These agents look for

suspicious activities on the host node, such as unusual process memory allocations, CPU activity, I/O activity, user operations (invalid login attempts with a certain pattern, super-user actions, etc). If an anomaly is detected with strong evidence, a local detection agent will terminate suspicious process or lock out a user and initiate re-issue of security keys for the entire network. If some inconclusive anomalous activity is detected on a host node by a monitoring agent, the node is reported to the decision agent of the same cluster that the suspicious node is a member of. If more-conclusive evidence is gathered about this node from any source (including packet monitoring results from a network-monitoring agent), the action is undertaken by the agent on that node, as described above. This functionality of our IDS system is currently undergoing active research. A few known approaches have been evaluated for suitability of application to wireless ad hoc networks, and a profile-based anomaly-detection system is being considered for deployment within the host monitoring agents.

5.3 Simulation of a Wireless Ad Hoc Network Clustering Algorithm

This section describes implementation of the distributed simulation of a large-scale wireless ad hoc network in support of the wireless clustering algorithm and packet monitoring for intrusion detection study, described above.

5.3.1 Objectives

- To compare the efficiency of our hierarchical ad hoc wireless network clustering algorithm [29] with a non-hierarchical scheme, where every wireless node

receives and processes a packet sent by any of its neighbors through an intrusion detection system's packet-monitoring module.

- To utilize a Case-based reasoner [30, 69] agent in the IDS packet-monitoring module at each node taking part in the simulation, and process each packet being transmitted, resulting in a realistic IDS scenario.
- To utilize MIT Lincoln Lab intrusion datasets [32] as a source of packets.

5.3.2 Simulation Design

The simulation *pre-conditions* are as follows:

1. Node mobility is governed by Random Waypoint mobility pattern with zero waypoint wait time.
2. For each node taking part in packet-monitoring task, the buffer storing incoming packets has a fixed length.
3. Network traffic is generated from the pre-processed database of packets, originally obtained from MIT Lincoln Laboratory IDS studies [32] and containing known intrusions.
4. Simulation area is fixed, and the network density (number of nodes per unit area) is varied from 10 to 100, resulting in elevations of network traffic rates.

The simulation proceeds according to the following scenario:

1. Ad hoc wireless nodes are created and communication parameters initialized.
2. Each wireless node participates in a clustering scheme via broadcast of control packets.

3. Each wireless node sends and receives packets and is mobile throughout the simulation.
4. Network density (number of nodes per unit area) is varied from 10 to 100.
5. Simulation executes for 10 minutes (wall clock) at each setting of network density.
6. Simulation clock is set to 250ms (4x faster-than-real time execution).
7. Each node taking part in packet-monitoring task records the number of total packets and the number of IDS-processed packets for every change in network density.
8. Total number of received and IDS-processed packets is recorded for each setting of network density.
9. Simulation stops after the final run has been completed, network density=100.

The network traffic used for this simulation comes from a processed Lincoln Labs intrusion dataset. As our WINDS simulation framework uses internal flat address scheme, all internal subnet addresses are mapped to existing nodes (e.g., N1, N2, etc.), while all external traffic from the dataset is mapped as arriving from Ext address. Timestamp information is also pre-processed and modified to start at 0 to match simulation clock. The processed packet data file has the following structure:

```
Source_addr;Source_port;Dest_addr;Dest_port;Protocol;Payload
N1;1036;N5;25;ICMP;A765BC7F54D3AC59…
```

Internally-generated packets are forwarded to each source wireless node via GetPacket() method by the simulation event handler. The node then routes the packet to the

destination using a certain ad hoc wireless routing protocol (e.g., ADSR). For the purpose of this simulation, routing protocol is not used; the packet is forwarded by the simulation event handler module on each processor to an appropriate receiver node, and the method GetPacket() is deprecated. An external packet generator class may also be devised. Simulation objects and their properties for this simulation are given in the table 5.1.

Table 5.1. Simulation objects for ad hoc wireless network clustering simulation.

Object	**Properties**
1. Ad hoc wireless node (wnode)	1. Mobility: Random Waypoint algorithm 2. IsClusterHead: Boolean 3. CommRange: integer – node's communication range in simulation distance units (e.g., meters) 4. GetPacket(): get next packet method – invoked by the simulation framework to pass next source network packet to the wireless node from the packet distribution process. 5. Send(): send packet method 6. Receive(): receive packet method 7. ProcessPacket(): IDS packet-monitoring method 8. PacketQueue: fixed-size incoming-packet buffer (40 packets) 9. cntReceivedPackets: long integer – total number of received packets by that node 10. cntProcessedPackets: long integer – total number of packets processed by IDS packet-monitoring service

The following figure 5.7 shows a simplified activity diagram of Ad Hoc Wireless Node object activities throughout the simulation.

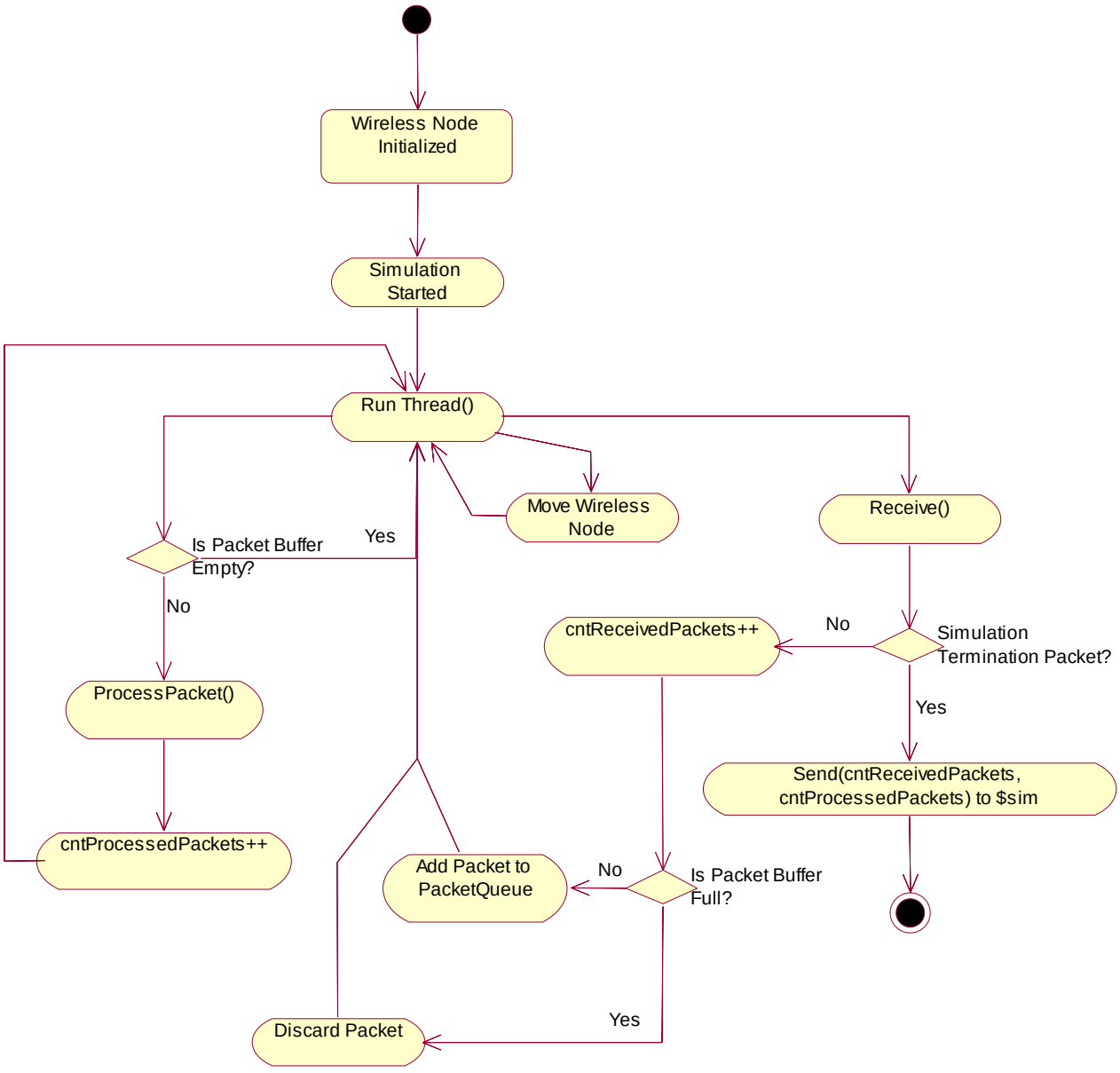


Figure 5.7. Ad hoc wireless node activity diagram.

The wireless node object is implemented as a Java class following the WINDS object design template, and has the structure (in pseudo code) depicted in the following listing.

```
// Wireless Node class
// Object parameters and default parameter values:

//#P str DefaultObjectNamePrefix, int initXPos, int initYPos, bool
isMobile, int commRange

//#D wNode, 0, 0, true, 800


public class wnode {

  int ID;

  String objName;

  simEvtHandler ParentClass;

// IDS & Clustering parameters

  int cntReceivedPackets = 0;

  int cntProcessedPackets = 0;

  queue PacketQueue[40];

// Other parameters

  boolean IsClusterHead = false;

  int CommRange;

// Methods

  wnode() // Wireless node object constructor - initialization

  void Tick() // Simulation clock pulse

  void Receive(Long simClock, String dataType, String strPayload)

  void Send(Long simClock, String dataType, String strPayload)

  void wMove() // Wireless node mobility

  boolean ProcessPacket() // IDS packet processing method

}
```

Listing 5.1. Ad hoc wireless node Class.

5.3.3 Simulation Execution and Results

The *wnode* class was placed in the *simulation root/objects* directory, then dynamically loaded by the simulation framework. The packet generation is performed by the simulation event handler on each processor (in case of a WINDS distributed simulation), or a separate packet processor class (as implemented in the single-processor simulation version). Each node then locally receives a packet through the *Receive()* method and processes the IDS lookup via *ProcessPacket()* method. The results are sent to the central *Simulation Statistics* window of the user interface, and are saved into a CSV file for analysis. Upon simulation execution, the following results were obtained.

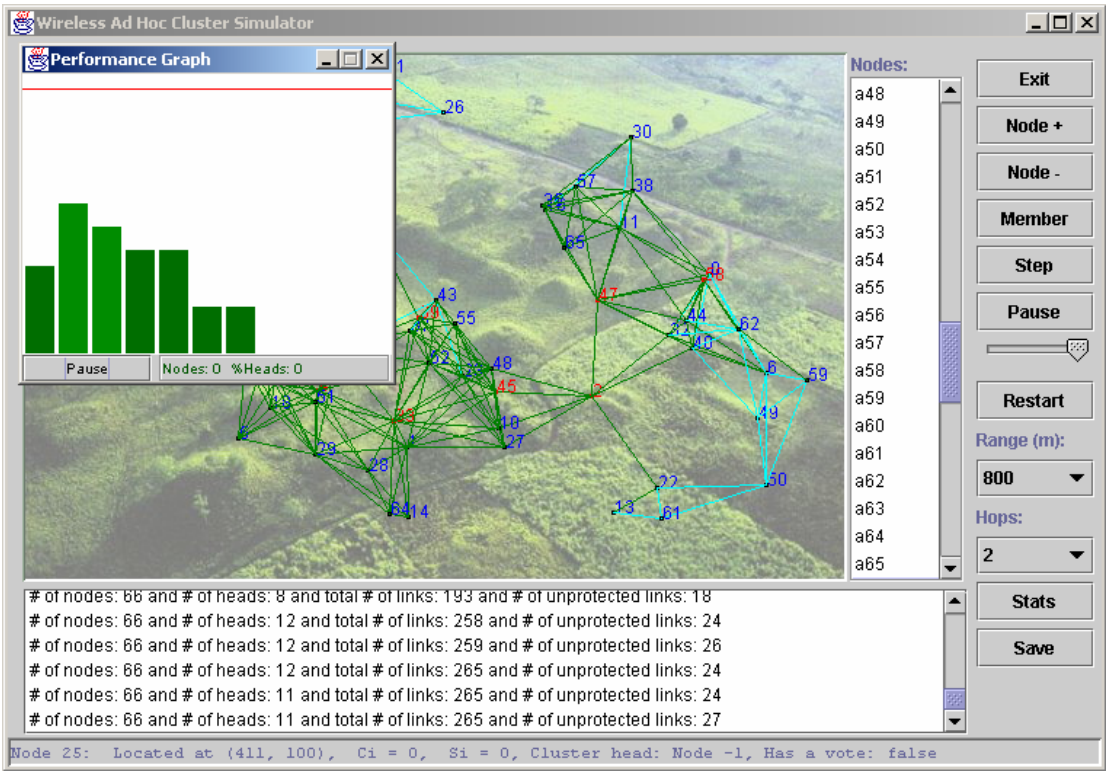


Figure 5.8. Packet-monitoring agent allocation simulation in progress.

The simulation results show that as the network density increases, the percentage of wireless nodes running packet-monitoring IDS functionality decreases, showing a very good scalability of the clustering algorithm (figure 5.5). At the same time, with the increase of wireless network density, these packet-monitoring nodes get overloaded with the amount of network traffic that needs to be processed, resulting in a large amount of packets being excluded from the IDS monitoring process (figure 5.6). The detailed explanation of these results has been given in the previous section, and the solution to the non-scalability of the packet-monitoring mechanism is presented further in this chapter.

5.4 Intrusion Detection Mechanisms

5.4.1 Collaborative vs. Independent Decision Making

Experience with intrusion detection systems designed for wired networks helps us to classify decision-making mechanisms for such IDS systems into two categories – collaborative and independent. The first type of decision-making mechanism is employed in systems where each node can take active part in intrusion detection process. An example of such a system is given in [86], where a simple majority voting scheme is used, in which any node that detects an intrusion with high enough confidence can initiate a response. More sophisticated cooperative decision-making schemes use fuzzy logic and rules to determine the threat level more accurately and initiate intrusion response. Such

mechanisms are discussed in [71] and [20]. However, such systems are prone to denial of service and spoofed intrusion attacks, where any (malicious) node can trigger full-forced intrusion response, affecting entire network.

In an independent decision-making system, certain nodes are designated to perform decision-making functionality. Their task is to obtain intrusion alert information from other nodes and to decide with a good accuracy whether or not a node in question presents a threat to network security. Other nodes don't have any influence on the decision-making process that concerns a certain node. This category of decision-making mechanisms is far less prone to spoofing attacks; however, the amount of information obtained by a decision-making node about each node participating in the network is limited. If a node in question had failed in local intrusion detection and all reporting mechanisms were somehow disabled, it will be difficult to detect such kinds of passive intrusion, where, for instance, a node could be intruded into and used as a passive listener on the network.

5.4.2 Intrusion Detection Process

Our intrusion detection system utilizes a customized independent decision-making mechanism. Decision agents are located on the same nodes as packet-monitoring agents. Detection and classification of security violations works as follows. Decision agent contains a state machine for all the nodes within the cluster it resides in. As intrusion or anomalous activity evidence is gathered for each node, the agent can decide with a

certain confidence that a node has been compromised by looking at reports from the node's own local monitoring agents, and the packet-monitoring information pertaining to that node. There is no need for other neighboring nodes to detect an intrusion or anomalies from the node in question, as this will be subject to denial of service (DOS) attacks on such a decision scheme. When a certain level of threat is reached for a node in question, decision agent dispatches a command that an action must be undertaken by the local agents on that node, as described in section 5.2. In time, the threat level decreases for each node in the decision agent's database. Decision-making agent maintains a "sliding-window" view on the intrusion data for each node within its cluster. Repetitive alerts of the same type within that window will cause an action to be undertaken by a decision agent to secure the breach in a network caused by a certain node or a group of nodes. The sliding window technique is necessary to account for certain uses of the network node that do not conform to accepted range of normal behavior, yet do not represent a threat to the wireless network as such.

Local anomaly detection models have been developed [86, 71, 20] that can detect an intrusion with a great degree of accuracy. According to the surveyed research, two types of profiling are made. Some IDS systems maintain a database of possible intrusion activity patterns and trigger alarm when such activity is detected. These systems result in fewer false alarms due to a variation in node usage patterns; however, intrusion activities with new patterns are likely to be underreported. The other category of IDS systems maintain a normal operational profile formed by a learning process. Anything that falls

outside such a profile of activities is classified as a possible intrusion. These systems have a higher false alarm rate, but are more likely to discover unknown intrusion, making such a model a choice for our IDS.

5.4.3 CBR Implementation of a Packet-Monitoring Agent

Packet-monitoring agents were described in the previous sections. Their functionality is to screen incoming network packets for known intrusions. This section describes the architecture of a case-based reasoner that is built into our packet-monitoring agents.

The generic Case-Based Reasoner (CBR) component assumes no knowledge about the application domain regarding case features and their comparison. The system can be tailored into a domain-specific case-based reasoner by defining the data type definition of the XML representation of a domain specific case, together with a metadata dictionary where the data about different case features, such as the required comparator and its value type, are stored. This use of a metadata dictionary and the separation of domain specific knowledge from generic components is an example of "adaptive" or "reflective" architecture. Hence the title for our packet-monitoring engine architecture. The advantage of this software engineering approach is that the same generic CBR source code can used for any application domain – no, or very minimal, additional programming is required.

The case-based reasoning process of an IDS is as follows. A packet is received from the network and fed into the CBR module. The packet is converted into an XML representation as specified in a corresponding DTD file. The search engine in the CBR module searches for similar cases in case archive. In the search process, cases are compared to the received packet's XML representation. For each feature in the packet XML data, the CBR module looks in the metadata dictionary for the type of the required comparator, and the comparator is created by reflection (i.e., during run time). Each comparator determines whether the packet feature matches the corresponding case feature. Once all the features are compared, the CBR module assigns a similarity value for the compared case. Lastly, the CBR module retrieves the matching case or cases and performs the prescribed action.

In order to implement a packet-monitoring agent using CBR, we converted the rules of the well-known Snort intrusion-detection system into a case archive. All the elements in a rule header, as well as the rule options, that are used in the rule matching process by Snort are treated as case features, and the rule action and its corresponding rule options (such as message element) are treated as the case action. An example of a Snort rule is shown below:

```
alert tcp $EXTERNAL_NET any -> $SQL_SERVERS 139 (msg: "MS-SQL
xp_peekqueue possible buffer overflow"; content:
"x|00|p|00|_|00|p|00|e|00|e| 00|k|00|q|00|u|00|e|00|u|00|e|00|"; nocase;
flags: A+; offset: 32; reference: bugtraq,2040; reference: cve,CAN-2000-
1085; classtype: attempted-user; sid: 697; rev: 3;)
```

The attributes in this case are protocol, source and destination IP addresses and ports, and packet's payload that contains a given hex string.

In the Snort IDS, the corresponding rule action is taken only if all of the elements that make up a rule match with the network packet. This means that in the corresponding domain-specific CBR system, the similarity metric must be bivalent; in other words, the matches must be exact. Hence there is no need for similarity ranking. Thus, from a CBR standpoint, the corresponding system is quite simple, and accounts for a sizeable number of false-positive and false-negative alarms. The entire packet monitoring CBR system, including the case archive created from Snort rules, together with required comparators, is quite small in size. Moreover, due to its modular design and implementation, both the reasoner and case archive components are portable. This is very important for network agent implementation. This CBR system serves as the core of the packet-monitoring agents in the intrusion detection process.

5.5 Load-Balancing for Packet-Monitoring Agents

This section describes distributed load-balancing approach to packet monitoring process that reduces the number of unscreened packets and improves the quality of intrusion detection process.

5.5.1 Load-Balancing Strategies for Network Packet Monitoring

This section presents a linear-time solution to the problem of packet-monitoring agent overload in the form of distributed online load-balancing algorithm. Our idea of load balancing the ad hoc wireless IDS resembles to some extent the techniques applied in computing clusters [45, 6]. Generally, a computing cluster consists of nodes connected by a communication network. Tasks can be split into multiple processes that can be assigned to execute efficiently on several members of the cluster in parallel. An optimal assignment of jobs to machines was shown to be NP-hard [24] even when jobs require single resource and the demands of this resource are known in advance.

In an ad hoc network, we divide wireless nodes into clusters of several connected machines. These clusters provide multitasking time-sharing environment for executing sequential jobs on multiple nodes at a time. The situation can become more unpredictable than in the case of a cluster computer, as wireless cluster membership changes rapidly and traffic patterns cannot be modeled precisely. Formally, load-balancing problem is defined as follows: Given n machines and a sequence of independent jobs arriving at random, minimize the maximal resource utilization among all the machines. A job j is defined by its demand vector:

$$p(j) = (p_1(j), p_2(j), \ldots, p_n(j))$$

where $p_i(j)$ is the resource demand of job j in a machine i, for a problem of one-resource allocation in a cluster of heterogeneous machines. For our intrusion detection system we

consider several resources on each node – CPU utilization, available memory and bandwidth. The above-mentioned load-balancing problem is then modified to minimize the maximal utilization of all the resources on all members of the cluster. As the intrusion detection tasks are CPU and memory-intensive processes, each resource carries different level of importance. We have adopted the weight ratio of CPU : Memory : Bandwidth as 5 : 3 : 1. Each node in a cluster then computes its weighted resource index that is then compared to the demand vector of a given job, using the formula:

$$R_i = 5a^{CPU\ (i)} + 3a^{Mem\ (i)} + a^{Bandwidth\ \ (i)} \qquad (5.1)$$

where for each node i, CPU, Mem and $Bandwidth$ are CPU utilization, memory utilization and bandwidth utilization, respectively, and a is a constant so that $1 < a < 2$ [45].

The sequence of intrusion-processing job assignment to other members of the cluster by a CH node is as follows:

- When cluster head (CH) node's resource index reaches a predefined threshold, it prepares for job delegation
- All nodes within the cluster compute their current snapshot R using equation 5.1
- CH node partitions the buffer of incoming packets scheduled for intrusion detection processing into batches

- Each batch job is selected to best fit the resource usage index of an optimally-selected node
- Each batch of packets is then forwarded to respective node for intrusion detection processing
- Results are returned back to the CH node's decision-making agent

The optimal node selection for the computing cluster job allocation has also been studied. One possible strategy is to use a Greedy algorithm. It is an online load-balancing algorithm for assignment of new jobs to a machine in order to minimize resource utilization. This algorithm has been shown to have a complexity of O(n) [6] for unrelated machines. Another online algorithm that was shown to improve the competitive ratio of Greedy for unrelated machines [6] is called ASSIGN-U. This algorithm defines a nonlinear cost function to assign jobs to machines in such a way as to minimize its marginal added cost. The complexity of this algorithm has been shown to be O(log n) for heterogeneous machines. We introduce several optimizations in the implementation section below, resulting in decreased processing due to a load-balancing procedure.

5.5.2 Optimal Job Assignment and Algorithm Implementation

As discussed, we consider three resources for the purpose of packet-monitoring task. When packet queue reaches certain threshold in a cluster head node, a message is sent to all current members of the cluster, asking for each node's resource index. Every node then forwards its resource index to the CH node. CH node then makes a decision on how

to optimally partition the buffer into batches of packets to be dispatched to selected nodes. The assignment is done in such a way as to minimize the overall resource usage RU, i.e.:

$$RU = \min(\sum_{i,j}(5a^{CPU(i)+CPU(j)} + 3a^{Mem(i)+Mem(j)} + a^{Bw(i)+Bw(j)})) \quad (5.2)$$

for each node *i* and task *j*. Overall resource usage is comprised from a sum of marginal costs of assigning each job to a machine within the cluster. In general case, when there are multiple heterogeneous nodes to choose from, each having multiple resources of different importance, the formula to compute the marginal cost is:

$$H_i(j) = \sum_{k \in M}(b_k a^{l_k(j)+p_k(j)} - b_k a^{l_k(j)}) \quad (5.3)$$

where M is a set of available resources, p_k is a demand vector of task *j* on resource *k*, and l_k is resource utilization of resource *k* on node *i*, and b_k is the weight of resource *k*.

When implementing the load-balancing algorithm in our simulation test-bed, we have adapted it to a low-power wireless network scenario and optimized several computation-intensive operations, as follows. Each node periodically computes its resource index and keeps a running average of recent resource usage, maintaining a sliding history window to the node's overall resource usage. This helps us decide on the node's current resource availability. This resource usage average is then periodically sent to the cluster head node (e.g., once every second). When a cluster head node receives

considerably more packets than it can handle (as in the case of a traffic burst), it will partition the queue of packets into batches and forward these to the most suitable node, as selected by the load-balancing policy.

One assumption we have maintained is that all the nodes within the cluster are homogeneous, and that each node has a 10-packet buffer allocated for IDS packet processing. In the simulation framework model, packets that were extracted from a network trace file averaged 1kb in size (this was due to MTU settings on the host system where packets were collected). Due to nodes being homogeneous, IDS packet-processing time is considered the same for all nodes, being an exponential distribution with the mean of 100ms. Network link speeds were taken to be 2Mbps (typical numbers for an 802.11b network), yielding a transmission time of 10ms, including overhead. This allows us to pre-compute each task's demand vector to be (CPU, Memory, Bandwidth) = (1000ms, 10240b, 10ms). Therefore, these computations won't have to be performed every time. As shown in [29], the average number of nodes per cluster is 5, for a network of up to 100 mobile nodes. For the optimal task allocation, cluster head (CH) node maintains a square matrix of resources for each cluster member, with each row being a vector of resource indices of all cluster members. When need arises, CH computes the H_i values, replacing the diagonal of the matrix, then adds each row of the matrix and picks the smallest number, with the index of that row indicating the node to which the next batch of packets is to be sent to. This operation is not computationally expensive, as average number of nodes per cluster is small. We have reduced the processing cost due to load-

balancing policy decisions, which is negligible compared to the processing requirements of the intrusion detection system. Therefore, there is a real benefit of using the load-balancing algorithm compared to a one-server approach.

5.5.3 Simulation vs. Analytical Results

To model the packet-monitoring situation, we view our model as a queuing system. First, we consider original one-server (cluster head) IDS packet-monitoring approach. The model in this case is an M/M/1 queuing system with fixed queue length of 10 packets, a Poisson packet arrival process, and exponentially-distributed packet-monitoring service times with a mean of 100ms per packet. The mean arrival packet rate is varied to accommodate several network scenarios, where number of cluster members and respective transmission rates are varied. Simulated versus analytical packet drop rates are shown in figure 5.9. Next, we consider the load-balancing algorithm applied to our intrusion detection system. The model is an M/M/n queuing system with n nodes per cluster that can act as packet monitors for the IDS. To further reduce the load due to intrusion detection processing on each wireless node, the incoming packet queue size is also reduced. From our earlier experiments with simulation test-bed executing ad hoc network clustering algorithm [29], the number of cluster members varies mostly in the range from 2 to 8, with an average being 5 nodes per wireless cluster, for an ad hoc wireless network with up to 100 mobile nodes scattered over a 4 square-mile area. This is depicted in figure 5.10, where results for load balancing approach with varied number of cluster members are compared to a one-server per wireless cluster model.

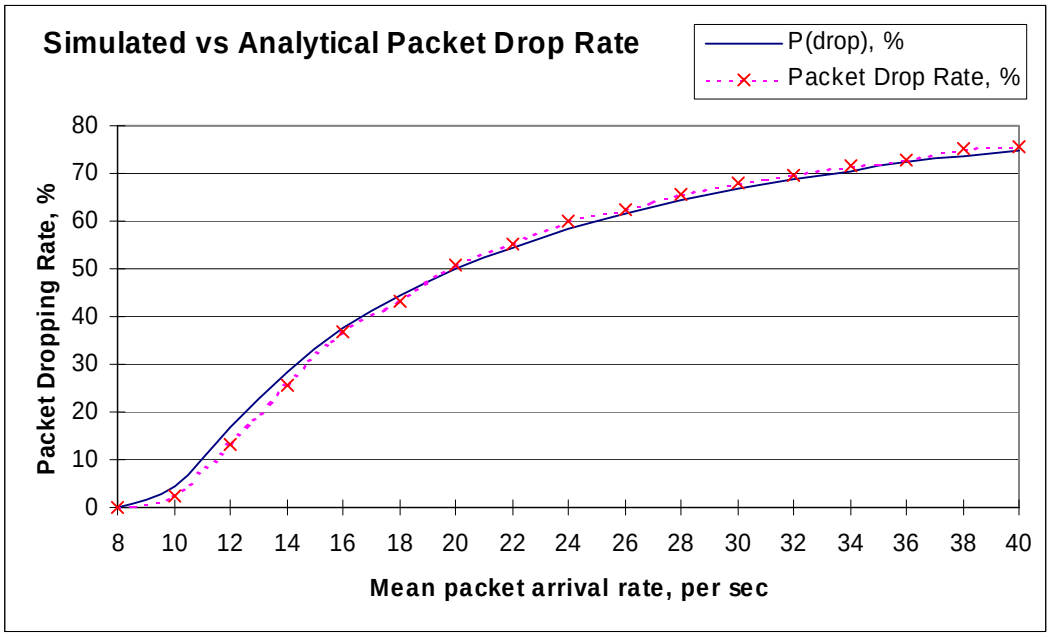


Figure 5.9. Simulated vs. analytical packet drop rate for one server per cluster IDS processing.

Upon implementing the load-balancing strategy, simulation results from our simulation framework show that there is a considerable reduction in the number of packets dropped from the cluster head node's intrusion processing buffer, as shown in figure 5.11. The graph shows considerably-reduced packet dropping rate as the network density increases, indicating good scalability of the DID (distributed intrusion detection) strategy, therefore improving the quality of intrusion detection. Using the distributed load-balancing algorithm, tasks of intrusion detection are optimally allocated to nodes within the cluster so that overall resource usage is minimized.

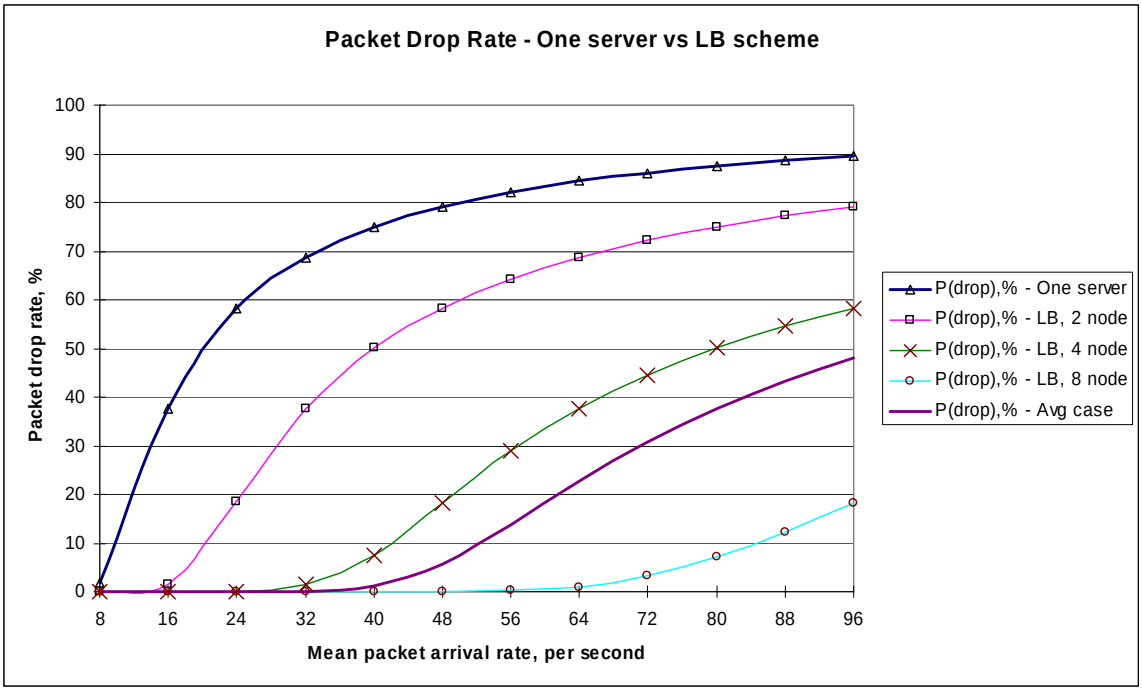


Figure 5.10. Load-balancing approach packet drop rates for various network configurations.

The worst-case scenario happens when there is a large amount of traffic on the network (see Figure 5.10). When this happens, cluster head node would follow the algorithm and pick nodes with least resource usage in sequence. However, as more traffic arrives, we may find ourselves in a situation that every node in a cluster performs packet-monitoring tasks. What would be the advantage of using our load-balancing algorithm compared to a monolithic intrusion detection system? There are several feasible advantages. First, each packet is reviewed only once by a single packet-monitoring node, as opposed to a monolithic system, where each wireless broadcast may result in multiple nodes checking the same packet, and thus wasting computing resources. Secondly, we

can regulate the load due to intrusion processing task by varying the quality of intrusion detection. That is, the amount of computing required for intrusion detection task on each node can be fixed, resulting in availability of processing power of each node for regular tasks. Therefore, excessive packets would be dropped from the system, when the network traffic increases considerably, thus aiding in a greater intrusion detection system scalability.

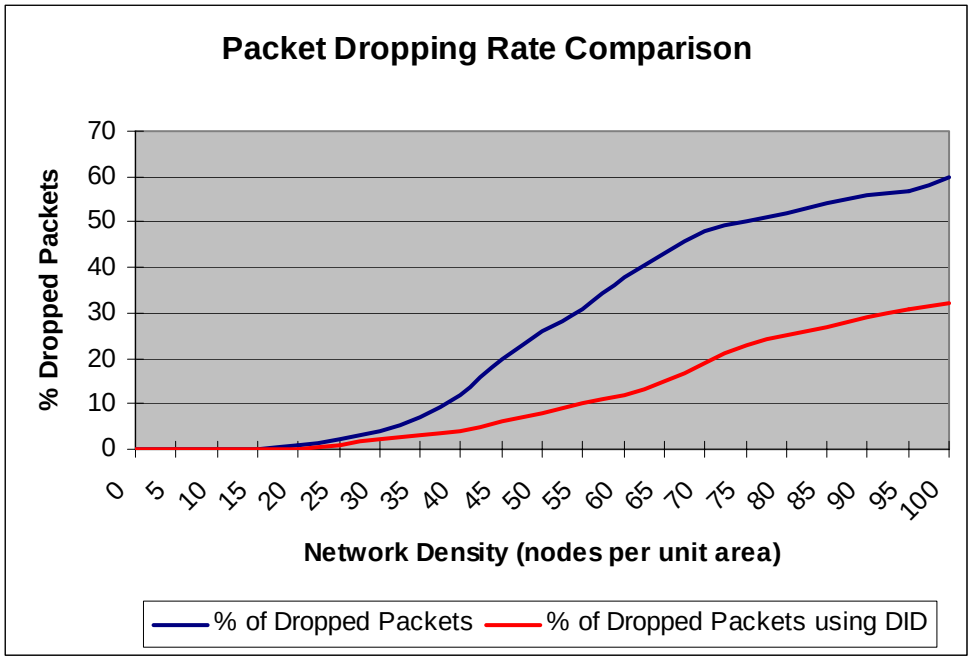


Figure 5.11. Decrease in packet dropping rate when using distributed load-balancing algorithm.

5.6 Summary

With emergence of a wide range of wireless devices, protecting ad-hoc wireless networks became an increasingly important but also a more difficult task. Scarce computational and power resources of mobile nodes impose heavy limitations on functionality of an effective intrusion detection system. Given these limitations, we have proposed a distributed modular IDS system designed for ad hoc wireless networks. This architecture is aimed to minimize the costs of network monitoring and maintaining a monolithic IDS system, also providing a high degree of protection against the intruder. New agents with added functionality can be plugged in when an expansion is necessary. Moreover, based on individual security requirements, the level of monitoring can be decreased resulting in greater availability of computational resources for the entire network.

Case-Based Reasoner [30, 69] has been implemented based on Snort rules that is incorporated into packet-monitoring agents and efficiently monitors incoming packets for known intrusion, checking them against the known database of attacks. The problem of packet-monitoring agent overload has been solved using an online distributed load-balancing algorithm, resulting in a much more accurate intrusion detection system.

CHAPTER 6: PERFORMANCE COMPARISON OF PINS, HLA AND TSPACES

This chapter describes a comparison study between two commonly-used distributed technologies – High Level Architecture, TSpaces, and our PINS simulation framework. A computationally-intensive algorithm for back-propagation neural network classifier training is implemented and simulated using HLA, TSpaces and PINS. First, we discuss these distributed technologies that are commonly used to create parallel and distributed systems. We then briefly describe parallel back-propagation neural network training algorithm for packet-based intrusion detection systems, which is being simulated in this study. Finally, we discuss implementation and performance differences in each of the technologies used and in the underlying hardware architectures – uni-processor and symmetric multiprocessor cluster computers.

6.1 Study Objectives

- To compare simulation performance of PINS simulation framework with High Level Architecture and IBM's TSpaces Java implementations.
- To utilize computationally-intensive back-propagation neural network algorithm for implementation using the above distributed technologies.

6.2 Distributed System Architectures – HLA and TSpaces

With increased complexities of computing problems we deal with today, the processing power of a single computer system became inadequate for certain problems of global scale; and use of super-computer wasn't always an option for many researchers. With the advent of computer networks and later on - computing clusters, the field of parallel and distributed computing has received a widespread recognition. Beowulf cluster implementation has raised parallel and distributed computing to a new level. Cluster computers were first introduced in 1994 as a result of the Beowulf Project at CESDIS. A Beowulf system is a collection of personal computers constructed from commodity-off-the-shelf hardware components interconnected with a local-area network and configured to operate as a single unit, a parallel computing platform, using an open-source network operating system (e.g., Linux) [8]. The driving design philosophy of a Beowulf system is to achieve the best possible price/performance ratio for a given computing problem. For many problems it's possible to achieve an order of magnitude improvement in price/performance compared with "conventional" parallel supercomputer designs. Cluster computing is used in this section as a hardware platform for comparing distributed computing technologies – Defense Modeling and Simulation Office (DMSO) High Level Architecture/Run-Time Infrastructure, IBM's TSpaces, and our PINS simulation framework – as implementation platforms for the back-propagation neural network training algorithm.

One of the most recent developments in distributed simulation technologies is the High Level Architecture (HLA/RTI), driven by DARPA, which has become a standardized framework (IEEE 1516) for modeling and simulation. It has been introduced in this dissertation in section 2.1.3. HLA is a component-based software architecture that incorporates subsystems for communication and data sharing, synchronization, and time management [59]. This framework facilitates distributed and multi-platform computing by providing standard integration architecture for separate and remote applications, thus facilitating reuse of components. HLA provides specifications for seamless integration of various simulation components; however, the efficiency of the entire simulation heavily depends on implementation and operational performance of each individual simulator comprising the simulation. At the network level, RTI utilizes reliable multicast protocols to implement inter-federate data exchange.

IBM's TupleSpaces (TSpaces) is a set of network communication buffers called tuple spaces and a set of APIs for accessing those buffers [85]. TSpaces allows heterogeneous, Java-enabled devices to exchange data with little programming effort. The package includes server software that implements the buffers and client software for accessing the buffers. The TSpaces server is composed of two main layers. The bottom layer comprises the basic tuple management. This is where tuple sets are stored, updated, indexed, and scanned. The interface to this layer is the Tuple Management API. The top layer comprises the operator component, which is responsible for operator registration and handling, implementation, and management. TSpaces provides group communication

services, database services, URL-based file transfer services, and event notification services. With its small footprint, it is ideal for bringing network services to small and embedded systems. TSpaces emulates a shared-memory multiprocessor architecture and reduces the complexity of writing parallel programs accessing shared data. This comes at the expense of message-passing performance, especially in the cluster computer environment, where the data is not physically shared among the processors.

6.3 Knowledge Discovery Problem for Intrusion Detection

Neural networks offer alternative means of maintaining a model of expected normal user behavior. They offer a more efficient, less complex, and better performing model than statistical models of system and user behavior [49]. Neural network techniques may be found to be more efficient and less computationally intensive than conventional rule-based systems. A lengthy, careful training phase is required with skilled monitoring. After the training period, the network tries to match actual commands with the actual user profile already present in the net. Any incorrectly predicted events actually measure the deviation of the user from the established profile.

The problem of data mining within large datasets places high demand on computational resources. This makes it a viable problem to measure the performance of a distributed system. Specific application of it is considered in our comparison study – a back-propagation neural network training algorithm for an intrusion detection system

(BPNN). Given a large initial training data file, the intrusion detector learning task is to build a predictive model (i.e. a classifier) capable of distinguishing between ”bad” connections, called intrusions or attacks, and “good” normal connections. A standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment, was provided by Lincoln Labs [32] in the form of a network trace file incorporating known labeled attacks. A connection is a sequence of TCP packets starting and ending at some well defined times, between which data flows to and from a source IP address to a target IP address under some well defined protocol. Each connection is labeled as either normal, or as an attack, with exactly one specific attack type [42].

The training process proceeds as follows. A finite dataset [50] of 2,000,000 records is partitioned equally among N processors. Each processor is an independent self-contained computer that is part of a computing cluster (further referred to as a Node). One node is designated as a Master and is responsible for task allocation and result unification; while other nodes are designated Workers for the back-propagation neural network. BPNN [83] is one of the widely used neural network training algorithms and has shown robust performance in many applications [22]. Several parallelization techniques have been introduced for BPNN [49, 47, 76]. Our choice for a cluster computer is a training method based upon set partitioning and epoch-based weights update schemes. This allows us to reduce the amount of inter-processor communication needed to distribute data, which is beneficial for a cluster computer environment with a

relatively slow inter-communication links. BPNN is trained iteratively, until an acceptable mean square error rate is achieved. At the beginning of each iteration (called epoch), master process creates a series of task requests and forwards them to the workers via a communication technology of choice. Each request contains information on a fraction of the entire dataset to be processed. Worker processes work on the corresponding fractions of the dataset and communicate results back to the master. Master process then aggregates all partial results. Once data mining step is complete, the quality of training result is evaluated. Algorithm follows certain steps to determine its error and propagate it back to modify the connection patterns of its hidden layers. The error value of the output unit j is:

$$e_j = a_j(1-a_j)(t_j - a_j) \tag{6.1}$$

The adjustment of the connection weights between unit i and unit j occurs using the equation:

$$\Delta weight_{i-j} = r * e_j * a_i \tag{6.2}$$

The activation function is:

$$a_j = \frac{1}{1+e^{-Ii-j*gain}} \tag{6.3}$$

where r is the learning rate of the network, e_j is the calculated error value, t_j represents the target output, and a_i represents the activation level of sigmoid function at the unit. The new value of a connection is an addition of the old weight value, and the change in weight.

Quality is measured by the classification rate with regard to the training dataset and the new testing dataset, instances of which were not part of the training dataset. Usually, good classification rate on the training dataset is regarded important; but good results on a testing dataset are also significant, as they imply that the data mining process was able to extract knowledge that represents not only the patterns in the training dataset, but also those of unseen patterns. For the purpose of this study, the testing dataset was not used.

The BPNN parallel neural network classifier problem for intrusion detection was implemented using several distributed communication technologies – High Level Architecture/RTI, PINS framework and TSpaces. Hardware platform used was a cluster computer with 16 individual nodes. Two clusters were involved in this experiment to detect the effect of using a symmetric-multiprocessor cluster configuration for processor-intensive tasks with operating system-directed task allocation. These clusters had the following configurations. Cluster 1 (Scerola) has 128 nodes with single 900 MHz AMD Athlon processors, running Linux Red Hat, and interconnected via Fast Ethernet network. Cluster 2 (Ariel) has dual Pentium-4 2.6GHz processor configuration on each of its 32 nodes, managed by SUN Solaris OS (only 16 were used for performance comparison), and interconnected via Gigabit Ethernet network links.

6.4 Simulation Design

HLA/RTI implementation involved defining user interaction classes and devising communication strategy among federates. First, the Worker federates were started, that would join RTI federation execution, publish and subscribe to interaction classes and partition the training file. Then each Worker waits for a work order from a Master process, and upon receipt, computes BPNN weights. After sending results back to the Master, the Worker awaits further work orders in an infinite loop. The Master process retrieves initial weights, allocates portions of work order to each Worker process, then sends out work orders and awaits results. When all the individual results have been received and combined, an error is computed. If within the threshold (0.01 percentile), the result is accepted and Master displays total run-time. Otherwise, a new set of work orders is distributed among the Workers. The total work order is divided in a way that each worker receives equal part of the training file for processing, since the program is executed in a homogeneous parallel environment with all nodes having approximately equivalent system resources.

The TSpaces implementation differs from the RTI implementation in that the Master doesn't communicate with a specific Worker process, but rather places a work order in a virtual shared memory. Since task processing time of Worker process is much larger than network communication time, and having the number of Workers equal to the number of work orders, the problem is evenly distributed among available processors. The algorithms for Master and Worker classes are given below:

```
Class Worker {

Connect to RTI;
Create / Join Federation Execution;
Publish / Subscribe to Interaction Classes;
Retrieve Training Data;

  Loop {
    Receive Work Order;
    De-serialize Parameters;
    Build / Verify Work Order Set;
    Compute BPNN Weights;
    Serialize Parameters;
    Send Result to Master;
  }
}
```

Listing 6.1. BPNN Worker Class Algorithm.

```
Class Master {

Connect to RTI;
Create / Join Federation Execution;
Publish / Subscribe to Interaction Classes;
Read Initial Weights;
Get Start Time;

  Loop {
    Compute New BPNN Weights {
        Send Out Work Orders to Workers;
        Receive All Results;
    }
    Test Result {
        If Result Successful, Display Running Time and Quit;
        Else Continue;
    }
  }
}
```

Listing 6.2. BPNN Master Class Algorithm.

The difference in implementation using RTI and TSpaces libraries is in Send and Receive methods. While RTI publishes interaction and transmits message via multicast to all federates, TSpaces implementation puts a data tuple into virtual shared memory and notifies all clients that data is available for pickup (via signaling or polling). Each client then picks up a tuple from tuple space, effectively removing it from shared memory. Since there is no addressing scheme and no way to specify a particular processor to send data to, the tuple space clients continuously scan for data that matches specific template.

The PINS implementation differs from HLA/RTI implementation only in communication parameters of Send() and Receive() methods between the *worker* and the *master* processes. RTI provides template for data communications by declaring parameter types at design time, and adding those parameter values in the parameter list prior to each communication. Then, the communication interaction is performed by broadcasting parameter list. Each *worker* process receives the interaction and parses it to find out if this interaction is addressed to that particular process. PINS implementation uses *master* and *worker* classes that communicate via built-in communication broker mechanism. Every time a message is sent to a particular *worker* class, only the intended recipient actually receives the communication, and no additional processing is performed.

```
// BPNN Worker class
// Object parameters and default parameter values:

//#P int numNodes, int myID
//#D 4, 1

public class Worker {

  int ID;

  String objName;

  simEvtHandler ParentClass;

// BPNN algorithm parameters

  int p_orderID;

  Vector p_currentSet;

  String p_hxILines[];

  String p_oxHLines[];

// Methods

  Worker() // BPNN Worker object constructor – initialize, load
           // data file portion

  void Tick() // Simulation clock pulse – not used

  void Receive(Long simClock, String dataType, String strPayload)

  void Send(Long simClock, String dataType, String strPayload)

  void getNextJob() // Receive work order from Master

  void buildSet() // Build BPNN weights set based on work order

  void verifySet() // Verify BPNN weights set based on work order

  void writeDoneTuple () // Send results back to the Master


}
```

Listing 6.3. BPNN Worker Class.

```
// BPNN Master class
// Object parameters and default parameter values:

//#P int numNodes, int myID
//#D 4, 0

public class Worker {

  int ID;

  String objName;

  simEvtHandler ParentClass;

// BPNN algorithm parameters

  int p_orderID;

  Double p_orderSum;

  String p_hxILines[];

  String p_oxHLines[];

// Methods

  Worker() // BPNN Worker object constructor – initialize, load
           // data file portion

  void Tick() // Simulation clock pulse – not used

  void Receive(Long simClock, String dataType, String strPayload)

  void Send(Long simClock, String dataType, String strPayload)

  void train() // Start training session for current epoch

  void distributeOrder() // Send out work orders to workers

  void buildDistributionSet() // Partition training file among workers

  void testBPNN() // Verify current training error


}
```

Listing 6.4. BPNN Master Class.

While it was possible to create PINS implementation that multiplexes BPNN parameters into one communication, separate consecutive messages were used to send different parameters. The code listings 6.3 and 6.4 outline the class structure of *worker* and *master* classes used by PINS implementation of the BPNN training algorithm. Listing 6.1 above shows the sequence of events for the *worker* object.

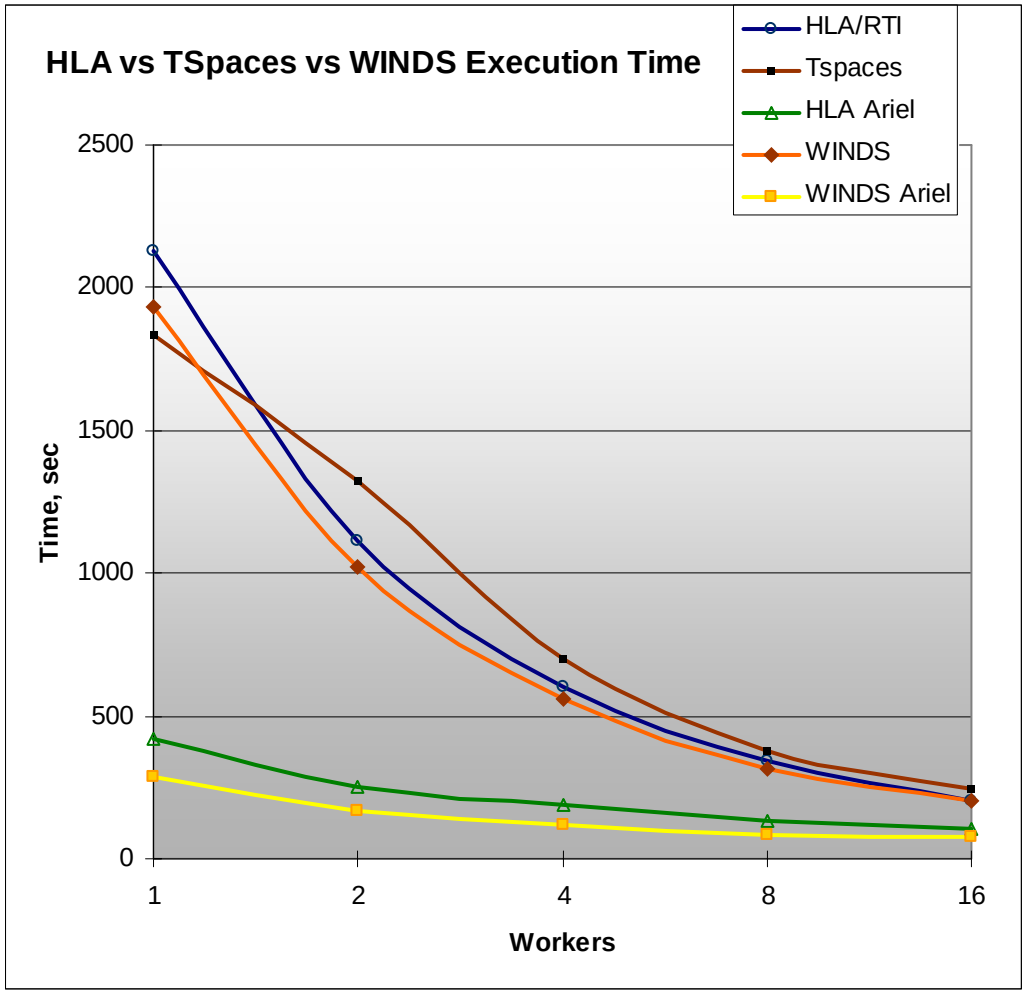


Figure 6.1. Parallel BPNN program execution times for HLA, TSpaces and PINS (WINDS).

6.5 Simulation Execution and Results

Simulation was executed for each implementation on an identical training data set. The results of running these implementations on a varied number of cluster nodes are plotted in figure 6.1.

Even though the HLA implementation required considerably more effort from software engineering perspective, the results show a considerable overall decrease in the total execution time compared to TSpaces, attributed to faster inter-node communications. The performance of PINS implementation is slightly better than that of a High Level Architecture – this is the case for both uni-processor and symmetric multiprocessor underlying hardware architectures. Implementation of PINS using WINDS framework involved less effort than HLA, and comparable to TSpaces, making it a viable choice for solving parallel and distributed systems problems. The performance of TSpaces is better than any other distributed technology in the case of only one worker. This may be attributed to the internal optimizations of TSpaces API implementation, detecting local communication and using direct method access. As far as the overall expense of the distributed system solution goes, the total processor time (on all nodes) plus inter-node communication demonstrates an added overhead of a distributed system. For instance, using RTI with one worker yields 2129 processor-seconds execution time (total time for problem solution by all processors, including idle time during wait, since we used a dedicated cluster with no foreign processes). Using 16 processors increases the total time to 3267 processor-seconds per problem. In situations where the processor time is an

expensive resource, a careful cost analysis and planning has to be performed prior to implementing the distributed system.

Examining performances of both HLA/RTI and PINS Java implementations on a single-processor and symmetric multi-processor cluster computers, we infer that Java Virtual Machine efficiently utilizes dual-processor configuration and dramatically speeds up computationally-intensive tasks without further modeling by the researcher. This demonstrates cost-effectiveness of using a multi-processor cluster configuration, since processing time is considerably reduced for the same-size task, compared to single-processor cluster configuration. In our tests of BPNN algorithm on all 32 nodes of Ariel cluster, the execution would occasionally deadlock when running HLA implementation. This is due to very small processing times on the client processors, resulting in increased utilization of communication channel. As the number of processors increases in a parallel system, problem implementation has to be planned and evaluated carefully to avoid deadlock problems, as network communication becomes the bottleneck.

6.6 Summary

In this chapter we have presented comparison of implementations of an intrusion detection-class data mining problem using various distributed technologies and concluded that WINDS offers better performance for certain distributed problems than comparable HLA and TSpaces implementations.

CHAPTER 7: CONCLUSION

The result of our research and development effort in the area of distributed simulation systems for wireless networks is PINS – an interactive, scalable, cross-platform parallel/distributed simulation system for computer networks. The PINS architecture is a scalable multi-processor simulation framework that can be used in tightly-coupled symmetric multiprocessor cluster computer environments, as well as in distributed networked systems (e.g., LANs). The object-oriented nature of the PINS architecture makes it possible to design distributed simulations and obtain results in a short time. A simulation clock synchronization mechanism is incorporated, that enables the framework to execute simulation events and commands simultaneously on all processors. Several parallel-system-oriented optimizations have been incorporated into the design of the PINS architecture. For instance, nodes with large inter-node communication requirements can be grouped on corresponding processors to reduce the amount of network traffic required for the simulation execution. This is achieved by serializing simulation objects and transporting them to the optimal destination processor. While this introduces an overhead in simulation control, for certain simulated problems it has proven to be a worthwhile strategy. A large network traffic dataset from the Lincoln Labs studies has been used to test the scalability of the distributed simulation framework running the simulation of intrusion detection system for wireless networks. The WINDS

simulation framework, based on PINS architecture, was developed to assist us in research on wireless network technologies and applications.

One such application is the intrusion detection system targeting ad hoc wireless networks. This IDS system was designed with the low-resource, high-mobility nature of mobile ad hoc networks (MANETs) in mind, featuring lightweight IDS components in the form of intelligent mobile agents. One aspect of our IDS system was thoroughly researched and implemented: the network packet-monitoring module, which captures network packets and analyzes them for known attempted intrusions. This component is based on a Case-Based Reasoner engine, and relies on a SNORT library of known intrusions. Due to low resource availability and low power of nodes comprising wireless ad hoc networks, our intrusion detection system was optimized to reduce the processing load due to intrusion monitoring on each node in a wireless network. A load balancing strategy is applied to distribute the amount of processing required by the IDS among several neighboring nodes, thus improving the quality of intrusion detection and reducing the load on individual nodes. These techniques were simulated using the WINDS framework. Since WINDS targets distributed computers of various configurations and network communication latencies, an intelligent algorithm is necessary to automatically adjust the distributed simulation clock to eliminate out-of-order delivery and processing of simulation events. We have incorporated a resource-monitoring module into our distributed simulation network to monitor network conditions and the performance of each node participating in a distributed simulation, and automatically adjust the

simulation flow to reflect dynamic changes in the simulation hardware environment. Simulation results from a variety of parallel-simulated algorithms show that PINS is a considerable improvement over a single-processor simulation execution in terms of scalability and speed of simulation execution for large network simulations. We have used a resource-demanding computational BPNN algorithm to compare the performance of our distributed simulation framework with current standards in distributed simulation systems, such as High Level Architecture and Tuple-Spaces.

Currently, the PINS architecture is used to study routing protocols for sensor wireless networks. Simulation objects relating to sensor networks will be added to the framework library. In the future, many simulation objects may be added, such as a MAC access control object, radio propagation and signal path loss models for wireless networks, etc. The PINS framework will also be used in studies of security protocols for wireless networks. Future work in intrusion detection for wireless networks may involve research into more robust and intelligent cooperative detection algorithms, as well as a choice of an anomaly detection model most appropriate for our IDS system. Although we have achieved a high level of detection of known intrusions, SNORT rules still give rise to a fairly high number of false-positive alerts. Data-mining techniques may be employed to reduce the amount of false-positive alerts and to further improve the accuracy of our intrusion detection system. Whereas an IDS system may detect attacks on mobile hosts, another possibility is to attack the IDS system itself. As our system employs mobile agents for intrusion detection, these mobile agents may be the primary target of an attack.

Possible attacks on our IDS system infrastructure and on individual mobile agents in particular may be investigated, and research conducted into effective means of defense against these attacks.

REFERENCES

[1] Amdahl, G.M, "Validity of the single-processor approach to achieving large scale computing capabilities", *AFIPS Conference Proceedings*, AFIPS Press, Reston, Va., vol. 30, pp. 483-485, April 1967.

[2] Anderson, J.P., "Computer Security Threat Monitoring and Surveillance", *Technical Report, James P Anderson Co.*, Fort Washington, Pennsylvania, April 1980.

[3] Anderson, R., Khattak, A., "The Use of Information Retrieval Techniques for Intrusion Detection". *Proceedings of RAID '98*, Louvain-la-Neuve, Belgium, September 1998.

[4] Ariel Cluster Computer, University of Central Florida, 2005, http://www.cs.ucf.edu/~jlee/Lab/facility.html

[5] Aspinall, D., "The ATLAS Compuer – The Technology", University of Manchester Institute of Science & Technology, May 2001, http://www.ukuug.org/events/linux2001/papers/html/DAspinall.html

[6] Aspnes, J., Azar, Y., et al, "On-Line Routing of Virtual Circuits with Applications to Load Balancing and Machine Scheduling", *Journal of the ACM*, Volume: 44, No 3, pp. 486-504, 1997.

[7] Bassiouni, M., Cui, W., Zhou, B., "Fast Routing and Recovery Protocols in Hybrid Ad-hoc Cellular Networks", *Book Chapter in Wireless Communications Systems and Networks*, edited by M. Guizani, Kluwer Publishing, 2004.

[8] Beowulf Project, http://www.beowulf.org/index.html

[9] Bernardes, M.C. and Santos Moreira, E., "Implementation of an Intrusion Detection System based on Mobile Agents", *Proceedings of International Symposium on Software Engineering for Parallel and Distributed Systems*, 2000, pp. 158-164.

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

[11] Bryant, R. E., "Simulation of Packet Communications Architecture Computer Systems", MIT-LCS-TR-188, 1977.

[12] Chandy, K. M. and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs", *IEEE Transactions on Software Engineering*, SE-5(5), pp. 440-452, 1978.

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

[14] Chiang, C.-C., et. al., “Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel”, *Proceedings of IEEE SICON*, pp. 197-211, April 1997.

[15] Chun Man, M., Wei, V. K., “A Taxonomy for Attacks on Mobile Agent”, *Proceedings of International Conference on Trends in Communications*, Volume: 2, pp. 385-388, 2001.

[16] CRAY T3D, http://www.cray-cyber.org/systems/t3d.php

[17] CRAY-1 Supercomputer, http://www.thocp.net/hardware/cray_1.htm

[18] Crosbie, M. and Spafford, E., “Defending a Computer System Using Autonomous Agents“, *Technical Report CSD-TR-95-022*, Department of Computer Sciences, Purdue University, 1995.

[19] Cui, W., Bassiouni, M., “Analysis of Hierarchical Cellular Networks with Mobile Base Stations”, *Journal of Wireless Communications and Mobile Computing*, John Wiley & Sons Publishing, Volume 2, pp. 131-149, March 2002.

[20] Dasgupta, D. and Brian, H., “Mobile Security Agents for Network Traffic Analysis”, *Proceedings of DARPA Information Survivability Conference & Exposition II*, 2001. DISCEX '01, Volume: 2, 2001, pp. 332–340.

[21] Denning, D. E., “An Intrusion Detection Model”, *IEEE Transactions on Software Engineering*, Vol. SE-13, pp 222-232, February 1987.

[22] Frasconi, P. et al, "Successes and Failures of Back-propagation: A Theoretical Investigation", *Progress in Neural Networks*, Vol. 5, pp. 205-242, 1993.

[23] Fujimoto, R., "Parallel and Distributed Simulation Systems", *Proceedings of the 2001 Winter Simulation Conference*, pp. 147-157, December 2001.

[24] Garey, M. R., Johnson, D. S., "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H. Freeman and Co., 1979.

[25] GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/

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

[27] Guan, X., Yang, Y., You, J., "POM-A Mobile Agent Security Model against Malicious Hosts", *Proceedings of the 4th International Conference on High Performance Computing in the Asia-Pacific Region*, Volume: 2, pp. 1165-1166, 2000.

[28] Guha, R., Kachirski, O., "An Architecture for Wireless Network Distributed Simulation (WINDS)", Fall 2003 SMART Publication, Modeling and Simulation Environment for Critical Infrastructure Protection project, https://quickplace.berbee.com/cip, 2003.

[29] Guha, R., Kachirski, O., "Intrusion Detection Using Mobile Agents in Wireless Ad Hoc Networks", *Proceedings of the IEEE Workshop on Knowledge Media Networking*, KMN'02, pp. 153-160, July 2002.

[30] Guha, R., Kachirski, O., Schwartz, D. G., Stoecklin, S., Yilmaz, E., "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

[31] Haines, J., Lippmann, R. et al, "1999 DARPA Intrusion Detection Evaluation: Design and Procedures", *Lincoln Laboratory Technical Report 1062*, Massachusetts Institute of Technology, 2001.

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

[33] Heberlein, L. T. et al, "A Network Security Monitor", *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Oakland, CA., May 1990.

[34] Helmer, G., Wong, J., Honavar, V., Miller, L., "Lightweight Agents for Intrusion Detection", *Technical Report*, Dept. of Computer Science, Iowa State University, 2000.

[35] Heybey, A., "MIT Network Simulator", *MIT Laboratory for Computer Science*, 1988.

[36] IBM Archives, www-03.ibm.com/ibm/history/exhibits/701/701_intro.html

[37] ILLIAC IV Computer, http://ed-thelen.org/comp-hist/vs-illiac-iv.html

[38] Intel Paragon, http://ed-thelen.org/comp-hist/intel-paragon.html

[39] Jansen, W., Karygiannis, T., "Mobile Agent Security", *Special Publication 800-19*, National Institute of Standards and Technology, August 1999.

[40] Jefferson, D., "Virtual Time", *ACM Transactions on Programming Languages and Systems*, 7(3), pp. 404-425, 1985.

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

[42] KDD Cup Data, *http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html*

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

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

[45] Keren, A., Barak, A., "Opportunity Cost Algorithms for Reduction of I/O and Interprocess Communication Overhead in a Computing Cluster", *IEEE Transactions on Parallel and Distributed Systems*, Volume: 14, No 1, pp. 39-50, January 2003.

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

[47] Klauer, B. et al, “Pipelining and Parallel Training of Neural Networks on Distributed-Memory Multiprocessors”, *Proceedings of IEEE World Congress on Computational Intelligence*, 4(27), pp. 2052-2057, 1994.

[48] Kotz, D., Gray, R. S., “Mobile Code: The Future of the Internet”, *Proceedings of the Workshop “Mobile Agents in the Context of Competition and Cooperation (MAC3)” at Autonomous Agents '99*, pp. 6-12, May 1999.

[49] Lee, J. and Siddiqui, M., “High Performance Data Mining for Network Intrusion Detection”, *IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004)*, MIT Cambridge, November 2004.

[50] Lippmann, R. 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.

[51] Liu, W. W. et. al., “Parallel simulation environment for mobile wireless networks”, *Proceedings of the 28th conference on Winter simulation*, pp. 605-612, 1996.

[52] Lunt, T. et al, “IDES: The Enhanced Prototype”, *Technical Report, SRI International*, Computer Science Lab, October 1988.

[53] Mah, B., “INSANE Users Manual”, *UC Berkeley*, 1998, http://www.employees.org/~bmah/Software/Insane

[54] Meadows, C., "A Formal Framework and Evaluation Method for Network Denial of Service", *IEEE Computer Security Foundations Workshop*, Mordano, Italy, 1999, pp. 4-13.

[55] Miller, D., Thorpe, J., "SIMNET: The Advent of Simulator Networkng", *Proceedings of the IEEE*, vol. 83, pp. 1114 - 1123, August 1995.

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

[57] Pawlikowski, K., Kreutzer, W., "Integrating Modeling And Data Analysis In Teaching Discrete Event Simulation", *Proceedings of the 2000 Winter Simulation Conference*, pp. 1645-1650, 2000.

[58] Ramanujan, R., Ahamad, A., Bonney, J., Hagelstrom, R., Thurber, K., "Techniques for Intrusion-Resistant Ad Hoc Routing Algorithms (TIARA)", *Proceedings of 21st Century Military Communications Conference*, Volume: 2, pp. 660-664, 2000.

[59] Reid, M., "An Evaluation of the High Level Architecture (HLA) as a Framework for NASA Modeling and Simulation", *Proceedings of the 25th NASA Software Engineering Workshop*, Goddard Space Flight Center, Maryland, November 2000.

[60] Riley, G. and Ammar, M., "Simulating large networks: how big is big enough?", *Proceedings of Conference on Grand Challenges for Modeling and Simulation*, January 2002.

[61] Royer, E., Toh, C.-K., "A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks", *IEEE Personal Communications*, Volume: 6 Issue: 2, April 1999, pp. 46-55.

[62] Russe, S., "Wireless Network Security for Users", *International Conference on Information Technology: Coding and Computing (ITCC '01)*, Las Vegas, NV, 2001, pp. 172-177.

[63] SANS Institute, "AINT Misbehaving: A Taxonomy of Anti-Intrusion Techniques", http://www.sans.org/resources/idfaq/aint.php?printer=Y

[64] SCD Supercomputer Library, "The CDC 6600", http://www.scd.ucar.edu/computers/gallery/cdc/6600.html

[65] SCD Supercomputer Library, "The CDC 7600", http://www.scd.ucar.edu/computers/gallery/cdc/7600.html

[66] SCD Supercomputer Library, "The IBM SP1", http://www.scd.ucar.edu/computers/gallery/ibm/sp1/wildhorse.html

[67] Scerola Cluster Computer, University of Central Florida, 2001, http://dart.ist.ucf.edu/dart/beowulf/scerola.html

[68] Schönauer, Willi, "Scientific Supercomputing: Architecture and Use of Shared and Distributed Memory Parallel Computers", *Rechenzentrum Universität Karlsruhe*, Germany, August 1999.

[69] Schwartz, D.G., Stoecklin, S., and Yilmaz, E., "A Case-Based Approach to Network Intrusion Detection", *Fifth International Conference on Information Fusion, IF'02*, Annapolis, MD, July 7-11, 2002, pp. 1084 - 1089.

[70] Sebring, M. et al, "Expert Systems in Intrusion Detection: A Case Study", *Proceedings of the 11th National Computer Security Conference*, Baltimore, MD., October 1988.

[71] Siraj, A., Bridges, S., Vaughn, R., "Fuzzy Intrusion Detection", *Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, Volume: 4, pp. 2165-2170, 2001.

[72] Smaha, S. E., "Haystack: An Intrusion Detection System", *Fourth Aerospace Computer Security Applications Conference*, pp 37-44, Tracor Applied Science Inc., Austin, Texas, December 1988.

[73] Snapp, S. R. et al, "A System For Distributed Intrusion Detection", *Proceedings of the IEEE COMPCON 91*, San Francisco, CA., February 1991.

[74] Spafford, E. H., "The Internet Worm Program: An Analysis", *ACM Computer Communication Review*, 19(1), pp 17-57, January 1989.

[75] Staniford-Chen, S., Cheunget, S. et al , "GrIDS - A Graph-Based Intrusion Detection System for Large Networks", *Proceedings of the 19th National Information Systems Security Conference*, Baltimore, MD., October 1996.

[76] Svensson, B. et al, "Using and Designing Massively Parallel Computers for Artificial Neural Networks", *Journal of Parallel and Distributed Computing*, 14(3), pp. 260-285, 1992.

[77] Takai, M., "Wireless Network Simulation in GloMoSim/PARSEC", *PARSEC Workshop*, UCLA, 1999.

[78] Tao, J., Ji-ren, L., Yang, Q., "The Research on Dynamic Self-Adaptive Network Security Model Based on Mobile Agent", *Proceedings of 36th International Conference on Technology of Object-Oriented Languages and Systems*, pp. 134-139, 2000.

[79] The Computer Museum, www.computermuseum.li/Testpage/IBM-709.htm

[80] Venkatraman, L., Agrawal, D., "A Novel Authentication Scheme for Ad Hoc Networks", *Proceedings of Wireless Communications and Networking Conference*, Volume: 3, pp. 1268-1273, 2000.

[81] Vigna, G., ed., "Mobile Agents and Security", Vol. 1419 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[82] Wikipedia, IBM STRETCH Computer, en.wikipedia.org/wiki/IBM_7030

[83] Williams, R.J. et al, "Learning Representations by Back-Propagating Errors", *Nature*, Vol. 323, pp. 533-536, 1986.

[84] Wilson, G., "The History of the Development of Parallel Computing", University of Toronto, http://ei.cs.vt.edu/~history/Parallel.html

[85] Wyckoff, P., "T Spaces". *IBM System Journal*, 37:3, August 1998.

[86] Zhang, Y. and Lee, W., "Intrusion Detection in Wireless Ad-Hoc Networks", *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom'2000, pp. 275-283.

[87] Zimmerman, P., "Status of IEEE 1516", Simulation Interoperability Standards Organization Newsletter, http://www.sisostds.org/webletter/siso/iss_75/art_362.htm