MiNT: A Reconfigurable Mobile Muli-hop Wireless Network Testbed

A DISSERTATION PRESENTED

BY

PRADIPTA DE

TO

THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE

STONY BROOK UNIVERSITY

May 2007

Abstract of the Dissertation

MiNT: A Reconfigurable Mobile Muli-hop Wireless Network Testbed

by

Pradipta De

Doctor of Philosophy

in

Computer Science

Stony Brook University

2007

Advisor: Professor Tzi-cker Chiueh

Academic research in mobile wireless networks relies largely on simulation. However, fidelity of simulation results has always been a concern, especially when the protocols being studied are affected by the propagation and interference characteristics of the radio channels. Inherent difficulty in faithfully modeling the wireless channel behaviors has encouraged several researchers to build wireless testbeds. One key difficulty of setting up a multi-hop wireless testbed is that it must be spread over a large physical space to introduce non-overlapping collision domains. This makes the setup, management and configuration of these testbeds challenging. In this thesis we focus on alleviating the space problem by designing a miniaturized 802.11b based, multi-hop wireless network testbed, called MiNT. MiNT occupies significantly smaller space compared to existing testbeds in use for mobile wireless

experiments. Engineered completely from off-the-shelf components, each node in MiNT supports untethered mobility that is controlled remotely. We also present solutions for problems associated with node mobility, namely node tracking in the testbed arena, and collision free node movement.

In order to provide flexibility in terms of setup, management and reconfigurability, MiNT provides a graphical interface for getting complete view of the testbed. Mint cOntrol and Visualization InterfacE (MOVIE) provides a suite of necessary controls for configuring each node, the entire topology, as well as, collects and displays experiment related statistics in real-time as well as offline. In addition to supporting the regular features for remotely operating the testbed, MiNT also supports two novel features: (i) hybrid simulation capability, and (ii) a fault injection and analysis tool (FIAT). Hybrid simulation allows execution of simulation experiments on MiNT with the modeled link, MAC and physical layer replaced by real hardware. Our results show that hybrid simulation provides more accurate results compared to pure simulation. Our hybrid simulation technique implemented on top of ns-2 has rich functionalities, like execution rollback, pause and breakpointing execution on flagged events, that makes debugging easier. The fault injection and analysis tool helps in debugging implementations of network protocols by introducing user-specified network faults without requiring any code instrumentation.

In the thesis we demonstrate the fidelity of our miniaturization approach by comparing experimental results on it with similar experiments conducted on a non-miniaturized testbed. Evaluations of hybrid simulation results against pure simulation results show the efficacy of the new approach in evaluating wireless protocols and applications. Evaluations of MOVIE and FIAT is intended to highlight the correctness and usability of the tools. Finally, we also a present a case study to underline the usefulness of MiNT, and a critique on remote usability of MiNT.

To my mentor, Mr. Ashim Kumar Basu

Contents

List of Tables

List of Figures

Acknowledgments

First of all, I would like to thank my adviser, Professor Tzi-cker Chiueh for providing me the opportunity to work on very interesting projects, and in a stimulating environment at Experimental Computer Systems Lab (ECSL). With his unrelenting ways of tackling problems, he has spurred me to scale new challenges in each and every project. High standards he sets through example has always inspired me. He has been a source of ideas all through, and discussions with him always leaves one with new directions to ponder. I am indebted to him for supporting me through the course of my doctoral studies at Stony Brook.

My dissertation committee members, Professor Samir Das and Professor Alexander Mohr from Stony Brook University, and Dr. Arup Acharya from IBM T.J. Watson Research Center have helped me with useful insights starting from the preliminary proposal phase. Professor Das's comments on my dissertation has been truly useful in making it better. Professor Alex Mohr has been extremely supportive and helped me with his comments all through. Arup Acharya made himself available every time I needed him to serve in the committee. I sincerely thank all of them for the effort they have put in towards my thesis.

I would like to specially thank my collaborators in the dissertation project; Rupa Krishnan, Ashish Raniwala, Srikant Sharma, Jatan Modi, Krishna Tatavarthi, Nadeem Ahmed Syed who has contributed in one way or the other to enrich the

project. Experimental Computer Systems Lab provided one of the best environments in terms of friends and colleagues: Srikant Sharma, Ningning Zhu, Ashish Raniwala, Gang Peng, Rupa Krishnan, Fanglu Guo, Manish Prasad, Anindya Neogi, Kartik Gopalan, Prashant Pradhan, and others who had been part of the group. Each of them in their own way made it an enriching experience to spend the years in Stony Brook. They have acted as sounding board for ideas, helped in shaping problems and solutions, and have supported me during the different phases in the road to this dissertation.

There are several other people in Stony Brook and outside whose support and presence was invaluable to me during the course of this work. They include Ajay Gupta, Saikat Mukherjee, Bikram Sengupta, Prem Uppuluri, Arnab Ray, Saumyadipta Pyne, Pranav Nawani, Diptikalyan Saha, Rahul Agarwal, and VN Venkatakrishnan. A note of thanks to my friends at Delhi: Nilanjan Banerjee, Dipanjan Chakraborty, Koustuv Dasgupta, Gargi Dasgupta, Amit Purohit, Shourya Roy, who kept me motivated in the final phase of writing the dissertation.

Finally, I owe a great deal of this achievement to my parents who have been a never-ending source of support for me. They have been there for me in all the troubled times with their encouragement and advice. My sister, Parna Mookherjee and brother-in-law, Suvadip Mookherjee are two other people who stood by me when it was most needed.

Last, but not the least, I would like to sincerely thank Mr. Ashim Kumar Basu who has been a mentor to me. His wisdom has provided me with the guidance in every new path that I have embarked upon.

Chapter 1

Introduction

> "Today's scientists have substituted mathematics for experiments, and they wander off through equation after equation, and eventually build a structure which has no relation to reality."
>
> — Nikola Tesla, Modern Mechanics and Inventions, July, 1934

A large set of academic research in wireless networks uses simulation as the primary means of validating their results. Simulation has several advantages which is highlighted by its effective application in understanding wired network behaviors. This has prompted the use of simulation tools in wireless network research as well. However, choosing correct abstractions in wireless network simulation is more complex than wired network because of the nature of wireless media. Questions regarding the credibility of simulation results is not uncommon [PJL02]. This tension has motivated researchers to validate their protocols on more realistic setting than simulation by building testbeds. We begin by highlighting one of the most common pitfalls of wireless simulation, and subsequently broach the topic of using testbeds as an alternative. The next topic of discussion focuses on a key set of requirements for designing a successful testbed for wireless experimentation. Finally,

the thesis proposal is a step towards designing an efficient wireless testbed with a comprehensive suite of software tools that supports experimentation.

1.1 Fidelity of Wireless Simulation Experiments

A correct simulation experiment should match closely the results from a similar experiment in a real set-up. This requires that various characteristics of the real world that has any impact on the protocol be correctly modeled. Thus the fidelity of a simulation experiment is dependent largely on the accuracy of the models used in the simulation. A "good" simulation experiment is one in which an experimenter is able to choose appropriate models to faithfully capture the behavior of the environment in which she wants to validate the protocol.

For wireless networks, setting up a good simulation experiment faces two problems: firstly, accurate and detailed models of the wireless channels are difficult to design, and secondly, detailed models are usually compute intensive, thereby increasing the overall simulation time. For instance the physical layer in wireless network stack is inherently complex due to the interaction of multiple factors that affect the wireless physical layer. These factors include path loss (large-scale fading), multi-path effects (small-scale fading), interference and channel noise. Path loss and multi-path effects are two key factors that determine the signal propagation behavior of wireless channels. Hence path loss and multi-path effects should be accurately modeled in order to model signal propagation. The path loss model calculates the average signal power loss of a path on a terrain. The commonly used path loss models in most simulators, like GloMoSim [ZBG98] and *ns-2* [NS-], are the free-space and two-ray path loss models. The free-space model is a basic reference model and is an idealized propagation model. The two-ray path loss model takes into account both the direct path and a ground reflection path of propagation.

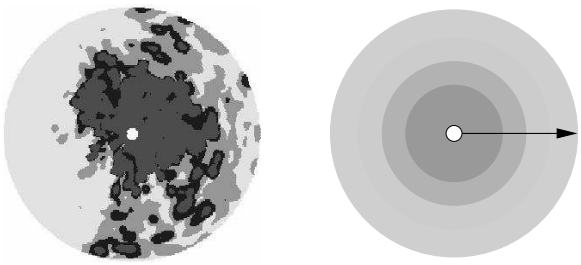

Figure 1.1: Comparing the commonly used signal propagation model to the typical real world signal propagation characteristics. Figure (a) represents a typical signal propagation characteristic from a point source. Different shades of gray denote different connectivity levels. Figure (b) shows the signal propagation as a result of the commonly used model in simulation.

On the other hand, the models for multi-path or small-scale fading effects calculate variation of signal powers at receivers due to varying path conditions from transmitter to receiver. Fading models with Rayleigh and Ricean distributions [Skl97] are commonly used to describe wireless environments.

Using the the commonly used path loss models, the received signal strength at a receiver is only a function of the distance between the transmitter and the receiver. In other words, the signal propagation region is a spherical zone around the transmitter such that points that are equidistant from the source always have the same received signal strength. It is common for most simulation experiments in the literature to use this signal propagation model. However, the reality is typically different, as shown in Figure 1.1[1]. It is seen in the figure that in real setting the signal strength varies non-uniformly with distance, whereas the modeled signal strength drops uniformly with increasing distance from the source.

[1]This figure is taken from http://www.comgate.com/ntdsign/wireless.html

It is true that more complex models can capture the behaviors accurately. However, as complex models are more compute intensive, a common practice in most reported simulation experiments has been to resort to simplified models, without much attention to the impact that the omission of details in the models might have on the final results. This has been pointed out by Heidemann et al. [HBE01] and Takai et al. [TMB01]. With more field experiments, inadequacies of the models are increasingly becoming evident. For instance, Zhou et al. observed that due to non-isotropic path losses radio irregularity is a non-negligible phenomenon in wireless communication [ZHKS04]. The difference in signal propagation effects in simulation and in reality have also been evaluated thoroughly [GEW+03, KNG+04]. Channel error is yet another phenomenon which is hard to model correctly [KWK03, KZJL03]. With a growing interest in cross-layer protocol optimizations [HJ04], it is important to get accurate values of the lower layer phenomena. For example an ad hoc routing protocol that uses the signal strength value to make path selection decisions will require the signal strength variations to be reflected correctly for proper evaluation. Other examples of cross-layer protocol optimizations include hop-by-hop error control in multi-hop wireless networks, channel state-dependent packet scheduling, and signal strength-aware packet routing in ad hoc networks. Unless simulation experiments are designed more carefully, simulation results will suffer from lack of fidelity for wireless protocol evaluations.

An alternate way to validate a protocol is to execute it in a real set-up. However, simulation has its own merits, namely, (i) the repeatable results of radio performance are hard to generate in an uncontrolled realistic setting, (ii) simulation tools are easily and freely available, in contrast to testbeds that are usually costly and time-consuming to set up. Although testbeds have been used occasionally to evaluate wireless protocol performance, its use has not been as widespread as simulation.

Most testbeds are custom built for specific projects, and unusable for other experiments. There is also no easy approach to build a standalone testbed within limited time frame. It is therefore necessary to provide a standard methodology for setting up standalone testbeds; if such testbeds could be shared across a wider research community it can become a useful and more accurate alternative to simulation.

1.2 Wireless Network Testbed Design Challenges

Having argued for the importance of using testbeds, it is imperative to take a closer look at the requirements for a wireless testbed. In fact, several research groups in academia and industry have shown keen interest in building their own testbeds for validating results. Therefore, it is worthwhile for the large community dealing with various aspects of wireless networks, comprising of researchers, application developers, and administrators to develop a better understanding of the challenges underlying the design of a versatile mobile wireless network testbed.

One of the foremost steps in setting up a wireless testbed is choosing the appropriate hardware that has favorable cost and performance tradeoff. Some of the key components required to build a mobile wireless testbed are: (i) the basic hardware platform which could be a desktop PC, a laptop, any other small form-factor PC, or some other device, such as a Linksys WRT54GS wireless router that runs a standard OS like Linux; (ii) wireless interface cards that give right degree of control in terms of adjusting different configurable parameters; (iii) external antennas (omni-directional or directional); (iv) other accessories, like RF signal attenuators, RF cables, steering device for directional antennas, battery packs; (v) platform for introducing mobility, which could be anything from cars, paid volunteers, to mobile robots capable of carrying payloads. Any of these components can either be

custom-made or commercial off-the-shelf products. Per node cost against the flexibility of operation is usually the guiding factor. For example, commercial wireless cards often provide limited configurability. It may not be possible to finely configure transmit power or reception sensitivity. It is therefore a challenging task to choose the right set of hardware for putting together a wireless testbed node.

The management of a testbed begins with the initial setup - hardware/software configuration and deployment of the nodes, and continues through the entire lifetime of the testbed in terms of monitoring the status of each node and link in the testbed, thereby keeping the testbed operational most of the time. The initial phase of management involves finding suitable locations for placing each node so that a true multi-hop network could be created. Thereafter monitoring the conditions of a testbed, often spread over a large geographical area, requires a visualization tool enabling remote monitoring through constant feedback on various node parameters, and changing wireless link conditions. This information could be useful in remotely restarting nodes that may have crashed during experiments, or for re-adjusting node positions for achieving desired link conditions. Often a difficult problem is to identify a set of parameters, like CPU activity, link Signal-to-Noise (SNR) ratio, etc. that should be monitored. The values of these parameters could be fed to the monitoring agent on a central controller, thus enabling remote administration and running of the testbed. For testbeds with node mobility, another key requirement for 24x7 autonomous operations of the testbed is the self-recharging capability.

Building testbeds that could be shared by multiple users is receiving significant attention in the network research community. The wired network research has been spurred by the development of the shared testbeds, like Emulab [EMU] and PlanetLab [PACR02]. Resource sharing in *wired* network testbed involves allocating a set of nodes from a pool to individual experiments (Emulab), or to multiplex several experiments on each node (PlanetLab). However, it is not possible to borrow

this paradigm unmodified for wireless testbed sharing among multiple users. Since wireless channel is a shared resource, it must be ensured that multiple experiments conducted on a testbed are isolated in the spatial, frequency, or temporal domains.

Having built a testbed, the next most important thing is the ability to use the testbed for experimentation. One can break up the process of executing an experiment successfully into two parts: experiment execution with flexibility for fine-grained control, and experiment analysis. Experiment control involves several steps, which we are going to describe in further detail. First, *configuring topology* involves placing the nodes in such a way that each node-pair satisfies some link property, like SNR or delivery rate. The difficult problem is that given a large number of nodes it is a tedious effort to determine the correct location of nodes that satisfies all the constraints. It is also not clear whether it is possible to come up with a completely declarative way of topology configuration, where a user specifies all the constraints, while locations of the nodes are automatically computed based on a priori measurements on the testbed. Second step is configuring the applications to run for an experiment. This involves setting up the traffic generators and traffic sinks, and can be done in two ways: a user can write her own applications, or a library of applications could be provided from which the user chooses appropriate applications. This could be useful in reducing the set-up overhead of an experiment. Third, a user should be able to configure node mobility by specifying the new locations of the nodes at specific times. Node movements must be triggered appropriately to reach the locations at fixed times. Fourth, the user must be given privileged access to a node to allow changing node/card configurations, as well as, to let her install kernel modifications implementing the protocol under test. Unprivileged access to users makes it necessary to be able to restore vanilla conditions once the experiment is completed. In addition, it is important to save the different

configurations the user has applied for an experiment because reloading the configurations could make re-running the experiment faster. The next step in experiment control is providing the user with ways to fine-tune an experiment by observing the results during execution. If one can start, stop and pause experiments, and modify parameters on the fly, it could potentially reduce experimentation time. Lastly, debugging any wireless protocol on a testbed involves all the known difficulties of distributed debugging. Hence having facilities for isolating bugs in protocol implementations could be an additional advantage in experimentation on a wireless testbed.

The experiment analysis part depends on accurate collection of the packet traces so that the packet dynamics during an experiment can be easily reconstructed, and studied offline. The first step is trace collection, which means recording the packets that are exchanged among the nodes. The standard technique for capturing packets is to use network sniffing tools, such as *tcpdump* or *ethereal*. However, these tools are not capable of capturing link-level packet dynamics, like link-layer retransmissions. In wireless networks, RF sniffing tools are used. Many commercial cards provide RF sniffing capability as an additional feature, called the *monitor mode*. In the monitor mode, a wireless NIC can capture all link-level transmissions including 802.11 headers, and 802.11 control frames. In a distributed environment, multiple monitor nodes are needed to completely cover the entire transmission domain of the nodes. There are two ways to set up the monitoring facility of a wireless testbed. One approach is to keep the monitor nodes separate from the experiment nodes. The advantage is that control and management of the monitor nodes is isolated from the testbed nodes; but placement of the monitor nodes must be carefully planned to ensure complete coverage of the transmission space. Alternatively, each testbed node can also perform the monitoring functionalities and sniff the packets in its neighborhood. In this case, each node must be equipped with at least two

wireless cards, assuming one card is being used for the experiment. Designing the monitoring facility is an important criteria for a useful wireless testbed. Next, it is required to aggregate the traces to build one consolidated trace that depicts the dynamics of the network. Traces from all nodes must be collected on a central node and merged based on timestamp. This entails that all nodes are time synchronized at the start of the experiment. Also, same packets could be collected by multiple nodes. These packets must be culled to get a clean trace of the packet dynamics. Finally, visualization of the collected trace constitutes an important part of analysis. Visualization must show the transition of packets from each node with respect to time. Visualization could be offline, or in real-time as the experiment is progressing. For real-time monitoring the collected traces must be transported to the controller node that must perform the parse, collate and display operations in an efficient manner taking into consideration the real-time constraints. Another element in packet trace collection and analysis is filtering of packets that are collected in order to reduce the amount of trace collected on each node. With user-defined filters, it could be possible to collect only the packets pertinent to an experiment. This is analogous to collection of application, routing or MAC layer traces in *ns-2* simulator. In real-time visualization of traces this could be very useful in reducing the amount of data that needs to be collected and processed at the controller.

An important question to ask is the applicability of a testbed. As we will see later in Chapter 2 there are several testbeds that have been designed with very specific goals in mind. It is worthwhile to invest some thought while designing a testbed such that it is applicable in as large a set of experiments as possible. While designing a shared testbed the goal is to incorporate facility for conducting diverse experiments, and protocol scenarios. Experiments could be live testing, emulation, or simulation. Apart from letting users run real implementations of protocols, another interesting mode of experimentation is *hybrid simulation*. Here, some layers

of the protocol stack in a simulator are replaced with real hardware. For example, the link, MAC and physical layer in simulator could be replaced by real hardware, while the routing, transport, and application layers are kept intact. In order to accommodate diverse protocols, requiring changes at different layers in the stack, it should be able to modify each layer. Usually the MAC layer implementation is part of the wireless card firmware, and no interface are exposed to modify it. However, depending on such requirements, testbeds could be designed using Software defined Radio (SDR) which lets you modify the MAC layer. Similarly, it could also be useful to be able to support different physical layer technologies, like GPRS, 3G, UWB, 802.11, or a mix of them in the testbed.

Last, but not the least, accurate repeatability on a testbed in an uncontrolled environment is hard to achieve because the external factors, like fading, attenuation, presence of other interfering source are always changing. In order to achieve repeatability, without sacrificing reality completely, it is possible to create controlled environments for the testbed by placing it in anechoic chambers and introduce interference in a regulated manner, or use RF cables shielding to prevent external interference. There is always the tradeoff between testbed realism and repeatability. An advantage of repeatability is that it could provide a reference platform for similar experiments.

1.3 Thesis Proposal: A Reconfigurable Multi-hop Mobile Wireless Testbed

In this proposal, we address some of the key inadequacies of existing simulation tools and wireless network research testbeds by developing *a miniaturized mobile multi-hop wireless network testbed* called *MiNT*. MiNT serves as a platform for

evaluating mobile wireless network protocols and their implementations. Like a generic wireless network testbed, MiNT consists of a set of wireless network nodes that communicate over one or multiple hops with one another using wireless network interfaces. A key feature of MiNT is that it *dramatically reduces the physical space requirement* for a wireless testbed while providing the fidelity of experimenting on a large-scale testbed. For example, using MiNT it is possible to set up an IEEE 802.11b-based 3-hop wireless network with up to 12 nodes in a 11 ft by 14 ft space. This space reduction is achieved by attenuating the radio signals on the transmitter and the receiver. Through this miniaturization it is possible to substantially reduce set-up, fine-tuning, and management efforts required for a wireless network testbed. Additionally, attenuation on the transmitters reduces the interference of the testbed with the production wireless networks operating in its vicinity.

MiNT is also a *hybrid* testbed platform that enables one to run *ns-2* simulations with its link, MAC and physical layers replaced by real hardware and driver implementations. The large number of wireless network protocols and traffic models already coded for *ns-2* can thus be directly used on MiNT. MiNT allows unmodified *ns-2* scripts to be executed on a set of physical nodes. Since the effects of radio signal propagation, like multi-path fading and interference, are better captured while executing simulations in the hybrid mode, it produces much more realistic results for simulation experiments.

The main contributions out of this research are,

- We present the architecture and implementation of a miniaturized wireless network testbed, that features mobile multi-hop ad hoc networking within a space that is order of magnitude less compared to full-scale testbeds. The node mobility infrastructure requires no manual configuration in (a) node movement control, (b) node position tracking and (c) node recharging, thereby significantly reducing testbed setup and administration cost. We also

verify the fidelity of the miniaturization approach and point out its limitations through extensive experimentation on an operational MiNT prototype.

- We have developed one of the first hybrid simulation platforms that can run unmodified *ns-2* simulations with its link, MAC and physical layers replaced by real components.

- The testbed offers one of the most advanced management interfaces among all existing wireless protocol simulation systems. This interface, called MOVIE, is able to provide a detailed real-time view of the system configuration, network traffic load distribution, node/link liveliness, and evolution of protocol-specific states. In addition, MOVIE provides users the flexibility to dynamically steer the direction of a simulation run (including reversing the execution) by inspecting protocol states and modifying protocol parameters, network configurations, and traffic loads accordingly.

1.4 Dissertation Outline

The rest of the report is organized as follows. First, in chapter 2 we survey several testbeds that are built so far with a focus on their effectiveness as a comprehensive wireless network testbed platform. Then in chapter 3 we present detailed system architecture of MiNT with an emphasis on the hardware components and techniques applied. In chapter 4 we present the design and implementation of all the software tools that are used in MiNT. In chapter 5, we have evaluated different aspects related to MiNT, like the fidelity of the miniaturization approach, the scalability of the testbed, effectiveness of hybrid simulation, as well as, presented the results of a case study on a wireless protocol, called ATP [Kar03] that demonstrates the applicability of MiNT. Finally, we summarize and conclude in chapter 6.

Chapter 2

Related Work

The background for this thesis has spanned several areas, starting from earlier efforts in building wireless testbeds to object tracking techniques in a dynamic environment; from existing tools for network management to tools for conducting experiments on a wireless testbed. In this chapter we present a survey of some of the work that is most relevant to the different components used in the design of MiNT.

2.1 Wireless Network Testbeds

A number of wireless testbeds have been built so far. Some of them have been full-scale outdoor or indoor deployments that are additionally used for experimental purposes. Design goal for some of the other testbeds have been to reduce the space required for setting up a testbed without sacrificing the fundamental features of an actual wireless testbed. In this section we present a critical view on these testbeds, that we group in two major categories: Full-scale Testbeds and Miniaturized Testbeds. In addition to these, there are some testbeds for which prototypes

are not yet in place (circa Dec, 2006), but has a novel approach in its design. We highlight the strengths and weaknesses of these testbeds, showing the techniques used to address core testbed design issues, and also point to some of the limitations.

2.1.1 Full-scale Testbeds

We first look at some of the wide area wireless deployments. Some of these testbeds were designed for specific projects. So the testbeds were tailored accordingly to satisfy the requirements of those projects. Recently there have been a growing interest in designing shared testbeds that could be used by a larger research community, and not confined to a project. In this subsection we will look at these two types of testbeds that are designed with a different set of users in mind.

CMU MONARCH group pioneered in building one of the first wireless testbeds for academic research when they set up a full-scale testbed for implementing and evaluating the Dynamic Source Routing (DSR) protocol developed by the same group. This testbed at CMU[1] was primarily designed for testing and evaluating DSR on a realistic setting apart from the simulations [MBJ99]. Since the primary goal of this testbed was to evaluate DSR protocol performance, the testbed is designed with 5 mobile nodes and 2 static nodes spread over an area of 700 meter by 300 meter. The mobile nodes are implemented with rented cars carrying laptops acting as mobile ad hoc nodes. For management of these nodes spread over a wide area this testbed had implemented a visualization daemon, that provides SNR information of each link. This testbed did not try to resolve the questions raised in relation to resource sharing among multiple users, efficient experiment control, or wide applicability. The purpose was to execute a specific application and study its behavior. It took around 7 months to set up this testbed, which points to the

[1] henceforth we refer to this testbed as CMU-DSR

difficulties of setting up a similar full-scale wireless testbed.

Ad Hoc Protocol Evaluation Testbed (APE) designed at Uppsala University is used for comparative study of different ad hoc routing protocols [LLN+02]. Similar to any other multi-hop wireless testbed, APE also uses a large space for placing the nodes. Node placements must be done manually, but management is aided by APE-view that is a log driven animation tool showing node positions and connectivity. This is the method for topology generation for experiments. Mobility in APE is introduced by providing explicit movement directions to volunteers carrying the laptops. This is a choreographed mode of mobility management. Since this is not a shared testbed, APE does not focus on usage across diverse experiments.

In order to facilitate mesh networking research, the *Roofnet project at MIT* came up with this testbed with nodes spread across volunteers' rooftops in Cambridge city [Cha02]. The testbed is also used to provide broadband Internet access to the users. Roofnet is deployed for conducting 802.11 measurement experiments to understand the nature of large-scale wireless networks. Since these nodes are static nodes, there is no flexibility of creating new topologies (changing transmit power may generate new topology by breaking some links), or introducing mobility. Roofnet is a network more useful for studying network characteristics rather than for experimenting with diverse application scenarios.

At Rice University researchers have built a testbed using *Transit Access Points (TAPs)* in order to explore the design of a high-speed wireless backbone [KSK03]. The core building block in the testbed, Transit Access Points, is a node equipped with multiple radios and antennas, which can be used in unison for spectrally efficient links at very high data rates. Lack of commercially available hardware with the specific requirements has prompted custom design of TAP hardware. This makes the TAP hardware expensive. Although the testbed is useful for specific types of experiments tuned to the project goals, it is not suitable for use across wide

variety of experiments.

2.1.2 Shared Testbeds

In tune with the growing needs for scale and realism by a wider community of wireless researchers, the following are some of the large-scale open platforms that are designed for shared usage.

2.1.2.1 Mobile Emulab

Mobile Emulab developed at Utah University is arguably the first mobile wireless testbed for shared usage. It is open for public use remotely, and is a good example of shared wireless testbed. Mobile Emulab is developed as an extension to Netbed [WLG02], which itself is a shared network emulation platform mainly for wired networks. The current version of the testbed, as per the webpage, consists of 6 motes and 6 Stargates mounted separately on robots (Acroname Garcia), besides 25 static Motes that are placed on the walls and ceilings in the testbed arena. The mobile wireless node component of the Mobile Emulab testbed comprises of a Intel Stargate, which is a single board computer, that hosts a 900 MHz Mica2 mote, and the combination is mounted on the Acroname Garcia robot for mobility. Six overhead cameras in the testbed arena provides vision-based tracking. The 25 static motes are also provided for giving additional flexibility with configuration and management. Besides these key components, there are additional features like live feed of the robot motion using webcams, integration with the prior Emulab environment, that provides a well-tested infrastructure for experimentation and data gathering from the testbed. The management of the testbed is still a pain-point because the charging of the batteries must be done manually. Hence the testbed is operated only on fixed working hours, and not on a 24x7 basis.

There are two challenging problems that were addressed in order to get this testbed operational. First, each node had to be tracked in the testbed arena. Since the Mobile Emulab uses a vision based tracking system consisting of ceiling mounted cameras, they faced a problem of inaccurate dewarping after the image is captured. They have devised a clever scheme of dewarping which gives them a locationing accuracy within an error of $1\ cm$. Secondly, since the testbed physical movement of the robots, it was necessary to devise a technique to allow collision free movement of the robots in the testbed arena. They use the proximity sensor on the Garcia robots to detect presence of any obstacles, and triggers corrective steps to avoid a collision. If there is a static obstacle the wireless node computes a path around the obstacle to reach its destination.

Though quite attractive in its design, and arguably the first mobile wireless testbed, the Mobile Emulab suffers from some drawbacks. Besides the fact that it is operational only for a fixed duration of time on weekdays, it also involves regular management tasks, like recharging the batteries. At one time only a single experiment can be run on this testbed since there is no space sharing. It is admittedly a difficult problem to make a wireless testbed truly sharable simultaneously among multiple users. Despite its few limitations, Mobile Emulab is an operational testbed for use by external users.

2.1.2.2 ORBIT

Open Access Research Testbed for Next-Generation Wireless Networks (ORBIT) is developed at Rutgers University [RSO+05]. ORBIT has built an indoor radio grid of 400 nodes in a 20 ft by 20 ft space (Figure 2.1). It is planned to be extended over an outdoor field trial network consisting of both high-speed cellular and 802.11x

Figure 2.1: *The large indoor grid at the ORBIT testbed comprising 400 nodes, as shown in the ORBIT webpage.*

wireless access. Each testbed node in ORBIT, shown as the ceiling-mounted yellow box in Figure 2.1, is custom-designed in order to provide several useful features, like remote monitoring of the nodes. Each node is static, and thereby it is easy to connect them to a central node using a wired interface making management functionalities easier and reliable. Mobility of each node in ORBIT is emulated through a separate mobility server, that transfers the state of a mobile node from one node in the grid to another. The topology generation is another problem due to the static nature of the grid. This has been circumvented by selectively switching some nodes on and off, as well, as through injection of noise to create links with different capacities [KGS06].

ORBIT provides an extensive infrastructure for running experiments, and subsequently storing the results of an experiment. Users must login remotely to the ORBIT testbed, and execute experiments. Some of its salient features are: (i) A host of traffic generators that can be used by a user to configure her experiment, (ii) Measurement Library, which minimizes coding effort of an user who can use predefined functions, (iii) well designed storage mechanism of all the results from an

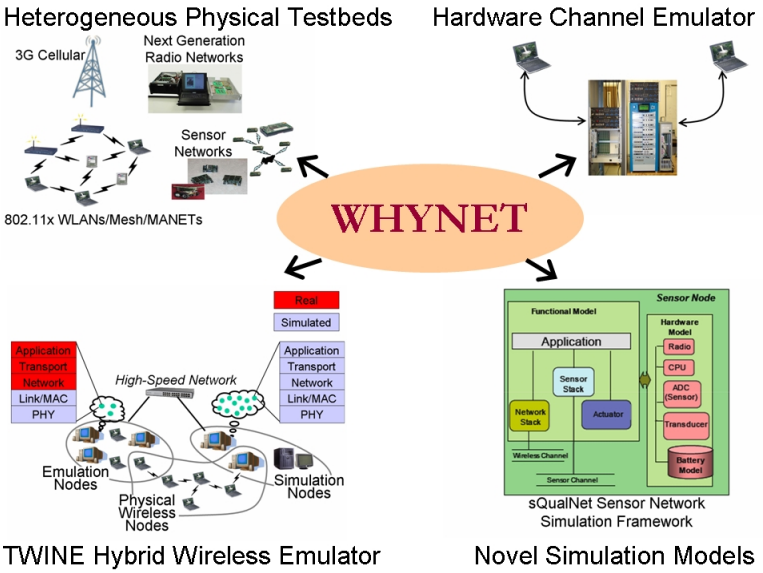

Figure 2.2: *A birds eye view of the WHYNET testbed and its components.*

experiment in a database that can be accessed through standard SQL queries. However, ORBIT does not appear to provide unprivileged access to a node, which may make it difficult to test kernel level protocol modifications on ORBIT. Nevertheless, ORBIT is a useful platform for testing protocols involving several technologies, like Wi-fi, high-speed cellular, Zigbee, Bluetooth.

2.1.2.3 WHYNET

WHYNET is a collaborative project among several university campuses which aims to come up with a shared testbed spread over geographically separated resources, and encompassing several wireless technologies, as depicted in the figure 2.2 taken from the WHYNET webpage. It is planned to be a scalable testbed for next-generation mobile wireless networking technologies [TBG+05]. Till Dec 2006, the WHYNET project has not opened up its resources for external use. However,

it is planned to be usable remotely by external users. There are several interesting features that WHYNET has incorporated. One of them is the hybrid emulation framework with two distinct capabilities: (i) seamlessly integrate emulation, simulation and physical testbeds for greater scalability or realism [ZJB06]; (ii) high-fidelity radio and channel emulation in real time. A number of simulators, like *ns-2* [NS-], GloMoSim [ZBG98] are also part of the testbed, besides several other tools, like sQualNet, SCTP, UWB, that have been developed for evaluating other wireless technologies.

2.1.3 Miniaturized Testbeds

Although large testbeds are more easily acceptable in the research community due to its similarity to full-scale deployments, such testbeds can often fall short in terms of manageability and flexibility of experimentation. In this subsection, we will look at some of the testbeds that tried to overcome the large space requirement, at the same time capturing the realism of full-scale testbeds as best as possible.

The *testbed at Sarnoff Research center* (which we refer to as Sarnoff-tbd) was the first attempt at experimenting with the idea of attenuating radio signals for setting up a wireless environment in a limited space. Kaba and Raichle at Sarnoff designed this testbed on a desktop by restricting and controlling radio ranges and propagation effects of the PC cards [KR01]. Through this testbed the authors demonstrate the idea of using fixed radio signal attenuators to reduce the radio range of wireless cards. In order to control the external effects on signal propagation, the testbed uses coaxial cables that can pass the radio signals from one card to another unaffected by external interference. The PC cards are shielded with custom-made copper shields to prevent leakage power from the internal antennas.

The use of programmable attenuators found its way in another project, called

RAMON at Florida State University. In this setup the experimenters aimed at evaluating the performance of mobile network protocols when the hosts are moving at high speed. RAMON is a rapid mobility network emulator [HH02] designed using mostly off-the-shelf devices. The novelty in the design of this emulator is in its use of programmable attenuators to vary the signal strength observed by a wireless station, which acts as the mobile host. Since lower signal strength indicates greater distance from the source (base station), therefore RAMON varies attenuation level on each node to emulate different mobility patterns.

EWANT is an *Emulated Ad Hoc Network Testbed* [SBBD03] built by Sanghani et al. with a goal of providing a low cost environment for wireless research. Similar to the Sarnoff testbed they also used attenuators and shielding to shrink the radio ranges. The mobility of the nodes is emulated without physically moving the nodes. It is emulated by connecting 1 PC card to 4 external antennas through 1:4 RF multiplexer, and switching the transmission through these antennas.

A novel approach in creating an environment for wireless experimentation is tried by Glenn Judd and Peter Steenkiste in their *physical emulation platform* approach [JP03]. They use digital emulation of signal propagation using Field Programmable Gate Arrays (FPGA). The aim is to have repeatable experiments, while preserving the realism of the MAC and physical layer. To achieve this, they use coaxial cables to feed the signal from a RF device to the emulator. The emulator controls the emulation of signal propagation by taking into account the impact of external factors, like multi-path interference, through use of signal propagation models. The main drawback of this approach is that external factors are still modeled, and is not truly real.

2.1.4 Notable Work-in-Progress

In this section, we will touch upon some of the testbeds for wireless experimentation that are still under development (circa Dec 2006). However, the uniqueness in their design, and the potential impact that these testbeds might have on the wireless research community as a whole makes each of them interesting to be included.

The *Kansei testbed* is a sensor network testbed developed at Ohio State University [EARN06]. It also reduces the transmit power through software control to scale down the size of the testbed. The Kansei testbed has been used to demonstrate an important result by Naik et al. [NEwZA06] that comments on the fidelity of scaling up/down from a full scale testbed. The claim is that the relationship between a link set and its scaled version is a probabilistic one. This implies that at different scales through repeated experimentation the behavior of the protocol will be statistically equivalent. However, an important empirical observation is that for a link set with only high and low received signal strength links the variation in path losses on different instances of the network have minimal impact.

The *Illinois Wireless Wind Tunnel (iWWT)* is a proposed testbed at UIUC [VBV+05]. The aim of this testbed is also to build a scaled version of a wireless testbed. But an important objective that iWWT targets is repeatability of wireless experiments. Repeatability for almost all known testbeds is difficult because of uncontrolled external factors. To alleviate the problem of external noise, iWWT proposes to build an anechoic chamber that will house the wireless nodes. Thus the entire radio environment will be under the control of the experimenters.

The *Heterogeneous Wireless Access Network Research Testbed (HWANRT)* project is focusing on exploiting heterogeneous wireless networks [HWA]. It explores the use of software defined radios (SDRs) for tiding over the differences in the use of radio spectrum by multitude of wireless devices. It is supported by Vanu Inc., one of the leaders in SDRs in this effort. In addition to it, the project also looks

at different network, transport and application layer protocols for bringing together heterogeneous wireless components to interact seamlessly in an integrated ad hoc network.

The *National Radio Network Research Testbed (NRNRT)* is an initiative to support research and development of new radio devices, services, and architectures and to provide a facility for researchers to test and evaluate their systems [NRN]. The NRNRT consists of (1) a field deployed measurement and evaluation system for long-term radio frequency data collection, and experimental facility for testing and evaluating new radios, (2) an accurate emulation/simulation system incorporating long-term field measurement for evaluating new wireless network architectures, policies, and network protocols, and (3) experiments with innovative wireless networks that integrate analysis, emulation/simulation, and field measurements.

2.1.5 Discussion

So far we have highlighted the salient features of most of the current wireless network testbeds. In Table 2.1, we present a characterization of these testbeds with respect to how efficient each of these testbeds are in terms of addressing the challenges in each of the key testbed features. The characterization is based on the available literature of the testbeds. In our view, large physical space requirement not only makes management of a testbed difficult, but also adds to its operational cost. This is true for most of the full-scale testbeds, like CMU-DSR, APE, RoofNet, TAP. We judged some testbeds, like Netbed, WHYNET and ORBIT to be high in cost, but the expenditure is justified keeping in mind the wide usage those are aiming. In evaluating a testbed on experimental control aspect, we emphasize the flexibility of new topology generation and introduction of mobility. Testbeds, like Roofnet, based on fixed nodes are less flexible compared to others on this count. A shared

Testbed	*Cost*	*Mgmt*	*Share*	*Expt Control*	*Expt Analysis*	*Repeat-ability*	*Applic-ations*
CMU-DSR	X	X	X	√	√	X	X
APE	X	X	X	√	√	X	X
ORBIT	X	X	√	√	√	X	√
WHYNET	X	X	√	√	√	X	√
Mobile Emulab	X	X	√	√	X	X	√
Roofnet	√	X	X	X	√	X	X
TAP	X	X	X	X	-	X	X
Sarnoff-tbd	√	√	X	X	-	√	X
EWANT	√	√	X	X	-	X	X
CMU-emu	X	√	X	X	-	√	X

Table 2.1: *Comparison of various wireless network testbeds in terms of the desirable features. An "X" denotes that the testbed has inadequate support for the feature and a "√" denotes that it addresses that feature. In all other cases, the presence/absence of the feature is not known.*

testbed must be rich in its applicability to diverse scenarios, like live experimentation, emulation or simulation. WHYNET, ORBIT and Netbed propose to address a truly diverse set of scenarios compared to others.

2.2 Support for Physical Node Mobility

Enabling physical node mobility in a testbed leads to several features that must be addressed, like introduction of a mobility platform, tracking the mobile nodes, allowing for their collision free movement, as well as keeping these battery powered nodes operational for the longest possible time. Different testbeds that allow physical mobility has attempted different techniques to address this challenge. This section reviews these methods for the different issues.

2.2.1 Mobility Platform

One of the earliest wireless testbeds to introduce node mobility is the mobile wireless testbed built at CMU for testing DSR protocol [MBJ99]. Each mobile node in CMU-DSR is a car carrying a laptop. Similar to this, the APE testbed at Uppsala University [LLN+02] choreographs the movements of volunteers who carry the mobile devices around the campus. These techniques of introducing mobility are closely dependent on human interaction. Ideally, the node movements should be remotely controlled and their mobility fully automated. In this paper, we take the approach of introducing mobility using remotely controlled robots.

The ORBIT testbed at Rutgers [RSO+05] introduces *virtual mobility* using fixed wireless nodes placed in an 8x8 grid. To simulate mobility of a virtual node, its state is transferred from one physical node to another thus simulating the effect of node movement. This technique, however, leads to discretized mobility over the grid

nodes. In contrast, our testbed is completely based on mobile robots and physical mobility.

Mobile Emulab [EMU] uses 4 Acroname Garcia robots for mobility. These robotic platforms cost over $1000 a piece, as opposed to our improvised robotic platform based on Roomba robotic vacuum cleaners ($249 a piece) [Rooa]. Moreover, Netbed's robots must be manually taken to their charging bases every 2-3 hours for recharge. Roomba comes with an auto-charging feature, that makes our mobile testbed truly autonomous. We also leverage on the design technique of using attenuators to miniaturize the testbed [DRScC05]. This makes the arena of operation smaller thus requiring smaller number of overhead cameras to track the nodes.

The mobility of the nodes in the first prototype of MiNT [DRScC05] was captured by mounting the external antennas (connected to the radio interface) on top of remotely controlled LEGO Mindstorm robots. This design restricts the mobility region of a node to a fixed space around its corresponding desktop PC, acting as the wireless device. The antenna cable length becomes the limiting factor in determining the space of operation for a node. In contrast, currently MiNT uses a design of mobile nodes that moves around the whole testbed arena without any limitation [DRK+06].

A key aspect in node mobility is to allow collision-free movement of the robots, which requires path and motion planning of the robots. Existing literature [CLH+05] has explored robot motion planning for various complex scenarios. Since our testbed offers a much controlled environment, we explore a heuristic that is lightweight, and computationally efficient. In contrast to the motion planning algorithm used in Mobile Emulab, since the Roombas do not have object sensor, we have built the intelligence to detect obstacles on the path into the trajectory planning procedure by using the tracking subsystem.

Robotic platforms for introducing mobility of nodes is not uncommon even in other areas of research. One of them is Caltech Multi-Vehicle Wireless Testbed [CDvG+02] used for studying decentralized control methodologies for multiple vehicle coordination and formation stabilization. They have custom designed their robotic platform, and pattern based tracking mechanism. In another similar project at UIUC, Scott and Kumar have built a wireless testbed based on toy cars. Tracking the cars inside the arena is based on matching colored patches on the cars that are grabbed periodically using an overhead camera. We use similar color based mapping techniques as these work. But we have simplified and scaled the overall tracking system using off-the-shelf webcams, and multiple colors. We are avoiding any form of pattern recognition techniques which makes detection of the nodes fast and reliable.

2.2.2 Node Tracking

Associated with mobility feature is the use of a tracking system for accurately determining the position and orientation of each node. There are various systems that use vision-based tracking system to track mobile nodes in different environments. Here, we briefly discuss three tracking systems that are most similar to ours. Graham and Kumar [GK03] use ceiling-mounted cameras and colored patterns on toy cars to track them. They use 8 colors and an error-correcting 3x2 colored pattern to track the cars. Their system is designed to track up to 22 mobile nodes and is able to uniquely identify nodes as well as provide their position and orientation. Cremean et al. [CDvG+02] use ceiling-mounted monochromatic camera and binary (black/white) patterns to compute the position and orientation of the nodes. Concurrent to this work, Johnson et al. [WLG02] have also implemented a centralized tracking system that uses ceiling mounted camera and color patterns to determine the position and orientation of the mobile nodes in their wireless testbed. Their

tracking system does not uniquely identify each tracked node, instead locality and motion pattern information is used to determine the identity of nodes.

Simplicity and cost of construction are the main differentiators of our tracking system. Use of off-the-shelf consumer webcams and standard APIs makes our tracking system inexpensive and easily portable. Additionally, miniaturization makes our tracking system more scalable. We also do not need any frame-grabber cards. Thus no specialized equipment/API needs to be procured and set up to install our tracking system.

2.3 Testbed Control and Management

Several software tools are designed for use in MiNT. These tools span different areas, starting from visualization interfaces to tools for better management of an experiment. This section focuses on these tools categorized according to their applications.

2.3.1 Real-time Monitor and Control Interfaces

Any testbed requires a visualization tool to study the dynamics of the experiments. CMU testbed [MBJ99] uses a graphical interface that displays the position of the nodes, the link characteristics, route changes for DSR protocol, and throughput information. Similarly, the MIT Roofnet [Cha02] also has an online map of the link characteristics among all the nodes in the testbed. It refreshes the link characteristics periodically. Similarly, Kurkowski et al. [KCC05] extended NAM to develop a tool called iNSpect, which adds features needed to study the mobility of nodes. Our extension of NAM (MOVIE) goes much beyond iNSpect in making NAM display

real time as well as showing important events related to wireless protocol evaluation, like route changes, link characteristics, and protocol-specific attributes. Additionally, MOVIE supports advanced control features such as experiment breakpoint based on specified events, and rollback of experiments.

Among commercially available tools for management of experiments Labview from national Instruments [Lab] is well-known. Besides this, there are several tools providing interfaces for wireless network management for enterprises. Any of them could definitely be used for managing a wireless testbed, but would be an expensive choice.

2.3.2 Emulation Platforms

The usefulness of seamlessly migrating from simulation environment to field testing using actual implementation has been noted earlier. *ns-2* itself provides a network emulation facility where real applications can interact with simulated ones [Fal99]. The nsclick project [NJG02] attempts to bridge the gap between simulation and deployment by presenting a set-up where the code written for *ns-2* simulation can be used with minimal change in real implementation. Saha et al. has also built a system, called PRAN which allows reuse simulation models directly for physical implementations of routing protocols [STP+05]. We do not set reuse of code as the main goal. Instead, we want to complement simulation tools by building validation platform that is based on these tools, but adds to the accuracy of the results of experiments performed using them. We achieve this by enabling unmodified simulation scripts to be executed on the testbed nodes with MAC and physical layer functionalities from physical world instead of using simulation models.

2.3.3 Application Debugging

There is a large body of literature covering different methodologies aimed at uncovering faults in systems. The idea of injecting faults into a system to validate its capabilities under exceptional conditions has been a well-known technique in hardware testing. This idea is used to test software systems, where we inject errors into the system in a program-driven manner and observe the response of the system to such exceptions. This technique is called Software Implemented Fault Injection (SWIFI). One of the early works in SWIFI has lead to the tool, called FIAT [SL88]. It is a tool that added functions to test trigger conditions and inject faults at compile time. Thus FIAT could trigger a fault on a condition such as the arrival of a message from a particular node. In another tool, called *Xception* [CMS95], Carreira et al. used the advanced debugging and performance monitoring features existing in most of the modern processors to inject faults by software, and to monitor the activation of the faults and their impact on the target system behavior. An environment for testing distributed real time systems was created in the DOCTOR project [HRS95]. It supported injection of memory, CPU, and communication faults. Stott et al. developed an integrated fault injection environment, called NFTAPE [SFKI00] which provides mechanism to inject different fault models into the test system. All of these systems aim at testing a wide range of faulty behavior of the system. Since we are mainly interested in detecting network-related faults in MiNT, therefore, it makes our fault injection and analysis tool compact, flexible and easy-to-learn.

In our fault injection and analysis tool we use "active probing" of the network traffic. The idea of active probing, as introduced by Comer and Lin [CL94] has been used with several extensions. Their idea was extended with the flexibility for manipulating messages in Orchestra [DJM96]. Orchestra uses a fault injection layer between the layer under test and the layer below on the network stack. The user has to program the fault injection experiment using Tcl and C. When generating

test cases from the protocol specifications, it is more intuitive to use a declarative language for generating the fault injection experiments because just like the design specifications, the fault scenario description can be kept separate from the specifics of implementation. Therefore, our approach is to present a domain specific language with a set of primitives that should be able to represent all network faults, as well as, perform analysis at run time. An approach similar to active probing has also been in used in the Piranha project [Gri99], where packet headers are corrupted systematically and protocol's response is noted. A work by Tsai and Singh [TS00] aimed at fault testing for Windows NT platform also uses the approach of interception of library calls and corruption of library call parameters. Both the works seem limited to only packet corruption/modification. We intend to address all possible faults that can arise in the network environment, hence packet modification is one of the different primitives we support in our fault specification language.

Emulating network interactions is also a widely used technique to observe the protocol behavior. Delayline [DG94] provides a configurable environment for emulating wide-area network over local-area network by specifying topologies with different link delays. Dummynet [Riz97] simulates different features of the network, like finite queue size, limited bandwidth and delays. It is used to test real protocol implementations. However, these tools lack the ability for packet manipulation.

Chapter 3

System Architecture of MiNT

Setting up a multi-hop wireless network is a grueling exercise mainly because determining suitable positions for placing a node that satisfies the multi-hopping requirement is a non-trivial task. Nodes must be placed such that there are pairs of nodes which are (a) *in communication range* of each other, (b) *in interference range* of each other, and (c) *outside interference range* of each other. This constitutes a typical multi-hop wireless network setup. Usually the communication distance between a pair of wireless nodes operating in 2.4 GHz frequency range and using off-the-shelf IEEE 802.11b wireless NICs is around 200 feet; the interference range for the same would be about 500 feet depending on the sensitivity thresholds set on different types of adapters. Even these distances vary dramatically due to several external factors, like multi-path interference, noise in the channel. Hence larger the physical distance among the nodes, more tedious and time-consuming is the task of setting up a multi-hop topology. Several research projects, like CMU-DSR [MBJ99] and MIT-Roofnet [Cha02], has reported experiences of setting up a multi-hop wireless testbed that has taken significant amount of time and labor. We assess that the *large geographical distance* to be one of the key pain-points in setting up a typical

wireless testbed. We propose a technique to alleviate the difficulty of setting up a multi-hop wireless testbed by reducing the *space of operation* of multiple nodes at the same time maintaining the properties of multi-hopping.

While configuring a topology, it is common to introduce incremental change to node positions. Also, mobility of nodes is required for an experiment. In order to study the aspects of node mobility in a full-fledged testbed, the nodes must often traverse large distances. This has prompted use of cars to drive a laptop around campus, as in CMU-DSR, or employ volunteers to carry mobile devices in an orchestrated manner, as in APE at Uppsala University [LLN+02]. Scaling down the space of operation gives us the opportunity to apply mobile robots to introduce remotely controlled mobility for the nodes. Mobility of the nodes with minimal direct human/administrator involvement is a desirable feature in a wireless testbed with large number of nodes. In our design of MiNT nodes we have kept in mind the need to take care of mobility in a manner that requires least intervention.

Our design of MiNT centers around two key ideas discussed. This chapter presents the design of a MiNT node and the overall reconfigurable multi-hop wireless testbed. We present the design of individual components in the testbed, with specific details on building a mobile wireless node, and other associated aspects, like tracking of a node in the testbed and issues related to keeping the testbed operational over long periods of time.

3.1 Building Blocks of MiNT

MiNT comprises of a collection of *core testbed nodes* managed remotely by a *controller node*. There is a tracking server setup aiding in the process of determining exact location of a node in the testbed. The overall interaction of the hardware and software components in MiNT is explained in this section. The components and

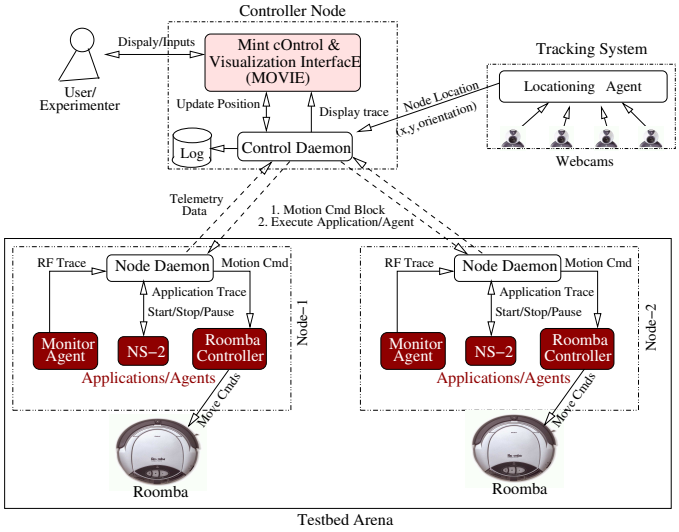

Figure 3.1: *Overall MiNT architecture. The* control daemon *running on the control server collects inputs from the tracking server and the user, and controls the movement of mobile robots. It also includes the MOVIE interface for monitoring and control. Each testbed node corresponds to a Roomba robot and has a* node daemon *running on it, which communicates directly with the control daemon over a dedicated wireless control channel that is non-interfering with the channels used for the experiments. The core testbed nodes communicate with the peers using wireless NICs that are connected to low-gain antennas through radio signal attenuators. The* vision-based tracking server *periodically captures images of testbed nodes, and processes them to derive the location of each testbed node.*

their relationship is shown in the schematic in Figure 3.1.

3.1.1 Hardware Components in MiNT

A *mobile node* comprises of a wireless computing device and a mobile robot for physical movement. In MiNT, the wireless device is a RouterBoard (RB-230), that is a low-power battery-operated small form-factor computing board. Each RouterBoard allows attaching 4 mini-PCI IEEE 802.11 a/b/g cards, that makes it possible to support multi-radio experiments [RC05]. For the wireless cards used in the experiments, a radio signal attenuator is inserted between the wireless interface and its antenna to shrink the signal coverage area and thereby the physical space requirement. The mobile robot is an inexpensive off-the-shelf robotic vacuum cleaner from iRobot, called Roomba. Necessary modifications to the Roomba are made to allow (i) the Roomba movements to be controlled from the wireless computing board mounted on it, and (ii) automatic recharging of a mobile node when the batteries drain out with suitable customization of the vendor supplied docking station, that has a self-homing feature to bring a Roomba back for recharging when in low-charge state. The mobile node design is discussed in detail in Section 3.2.

The *control server* is a PC equipped with multiple wireless network interfaces. In our current prototype the control server uses 3 wireless NICs. All control traffic is transported over an IEEE 802.11g channel and thus does not interfere with IEEE 802.11a channels, which are used in actual experiments. Multiple NICs allow the flexibility to scale the testbed to an increasing number of testbed nodes as the control traffic grows with additional nodes in the testbed.

The *tracking server* is used for providing accurate position and orientation information of a node in the testbed as it changes its location in the testbed. The *tracking server* can be a cluster of PCs depending on the number of cameras used.

Figure 3.2: *MiNT prototype with 12 mobile nodes and charging stations (top left corner of the image).*

The tracking system periodically sends snapshots of the entire testbed captured using a (3x2) grid of commodity web cameras, and uses them for node identification and locationing. Smaller physical space requirement plays to our advantage in reducing the number of cameras required to cover the entire space.

Figure 3.2 shows a prototype of MiNT consisting of 12 nodes, with charging stations visible at the top left corner of the image.

3.1.2 Software Components in MiNT

The key software components in MiNT are: (a) the *control daemon* running on the central control server, (b) the *node daemon* residing on each testbed node, and (c) the network monitor and control interface called MOVIE (Mint cOntrol and Visualization InterfacE).

The *control daemon* runs on the control server. It collects position updates of the core testbed nodes from the tracking server, as well as, event traces from the nodes used in an experiment, which it uses to update the visualization interface. It

also communicates user-issued control commands, regarding node position or configuration changes, to individual node daemons. Based on the position commands issued by the control daemon the node daemons trigger the movement of mobile robots. Because all event messages from the testbed nodes are routed through the control server, the control daemon maintains a complete log of the activities in the testbed.

The *node daemons* on the testbed nodes communicate with the central control daemon over an IEEE 802.11g channel that is fixed at start-up time. The messages being communicated are either movement commands from the central control daemon, or simulation events reported by testbed nodes back to the central control server. Other programs running on testbed nodes, for example, an ns-2 simulator, a TCP sender, or an RF monitoring agent, rely on the node daemon for any communication they may require with the central control server. For example, critical events in the event trace that an ns-2 simulation run generates are passed in real time through the node daemon to the controller node for display.

MOVIE provides a comprehensive monitor and control interface that offers real-time visibility into the testbed activity and supports full interactive control over testbed configuration and hybrid simulation runs. MOVIE is derived from Network Animator (NAM), a well-known off-line visualization tool for ns-2 traces. Several enhancements in the form of powerful features for real-time monitoring and controlling simulation runs and for interactive debugging of simulation results, such as protocol-specific breakpoints and simulation state rollback, has been introduced.

3.2 Core Node Design

MiNT is a reconfigurable multi-hop wireless network testbed that is possible to set up in a very small space. The key principles that guided the design of a core testbed

node in MiNT are (a) miniaturization of the area of installation of a multi-node multi-hop wireless testbed, (b) remotely controlled mobility of the nodes to allow easy reconfigurability of topology in the testbed, (c) and last, but not the least, the emphasis on building a low-cost testbed from commodity off-the-shelf equipments as much as possible.

3.2.1 Miniaturization

A typical wireless testbed spans a large geographical area because radio signals from transmitters used in commodity cards can be usually received over a large radius of the order of few hundred meters. This is contrary to our goal of setting up a testbed in a limited space because every node within the area will be in single hop communication range of each other. To achieve our goal of miniaturization, it is imperative to restrict the radio signals from a sender before it reaches the receiver within a small area. This enables us to set up a pair of nodes in a manner such that even though the nodes are separated by a small distance the signal from the sender can be prevented from reaching the receiver. This is the key to establishing multiple collision domains in a much smaller area compared to an unmodified wireless set up.

The simplest technique for limiting radio signal propagation distance is to *reduce the transmit power* on the wireless interface card. One can use a laptop or a PDA with a commercially available PC card that allows setting the transmit power to different values, like 100 mW, 50 mW, 10 mW, 5 mW, 1 mW. Since we are aiming to minimize the space as much as possible, we tried using a Cisco Aironet 350 series card that allows us to reduce the transmit power of the card to the smallest value possible in a commercially available card (1mW). However, experiments revealed that this transmit power setting is still too high to carry the radio signal across

two mid-sized rooms. This defeats our goal of miniaturizing the testbed to the desired scale of a couple of feet of communication distance between two neighboring nodes.

The alternative choice is the use of *radio signal attenuators*. Radio signal attenuators are available in two different types, *viz.* fixed signal attenuators and programmable attenuators. However, there is a stark price difference between the two: the fixed signal attenuators are priced in tens of dollars, as opposed to the programmable attenuators which are usually close to $1000 a piece. Therefore, we choose fixed signal attenuators to design a low-cost core node[1]. We determine the extent of attenuation (dB rating) based on the desired range of signal propagation. Use of attenuator, which is external to a wireless card, requires the use of an external antenna. The attenuator is connected between the PC card and the external antenna using RF cables with suitable connectors. The problem with this approach is that most commercially available PC cards come equipped with an internal antenna. The internal antenna is not fully disabled upon attaching an external antenna, and radiates significant RF energy, thus defeating the goal of miniaturization.

There are two ways to overcome this problem: one is to desolder the internal antenna[2]. The other option is to use a card that does not have an internal antenna. It is hard to find a PC card without internal antenna; however, miniPCI and PCI cards are available which do not have an internal antenna. Since the computing platform of our choice, RB-230, provides only 1 PCI slot, we opt for miniPCI cards. Multiple miniPCI wireless NICs can be attached on RB-230 making it possible to use a node for experiments requiring multiple interfaces [RC05].

Another step aiding in attenuating the signal propagation is the use of low-gain antennas. Since we are using external antenna, we opt for antennas with a gain of 2

[1]For a detailed specification of equipments used in a MiNT prototype refer to Appendix A.1

[2]This method is reported to be applied by the wireless Emulab testbed [JSF+06]

dBi.

3.2.2 Node Mobility

The unrestricted node mobility in MiNT must be implemented using mobile robots. Key determinants for our choice of robots are: (a) low-price, (b) minimal assembly, and (c) remote controllability. Hobby robots provide an inexpensive option, but require extensive assembly. Robots typically used in many commercial and robotics applications, like AmigoBots [Ami], PatrolBots or Acroname Garcia [Acr] robots, are very high priced, often going into thousands of dollar for each piece. Hence, instead of choosing one of these standard robotic platforms, for building the MiNT nodes we choose a *Roomba robotic vacuum cleaner* from IRobot as the mobility platform for a wireless node. Roomba is a consumer grade product with a retail price of $249 at the time of our purchase. Use of Roomba greatly reduces the cost per-node in MiNT.

Designed primarily to be a vacuum cleaner, Roomba does not have an open API for controlling its movements. We overcome this limitation through a clever use of its IR-based remote control facility. More specifically, we achieve arbitrary Roomba movement using two primitives: (1) move the mobile robot forward, and (2) turn the robot clockwise or anticlockwise. A Roomba can be instructed to perform these primitives through a out-of-the-box remote controller that comes with the Roomba. We learn Roomba's remote control codes using a programmable remote controller called Spitfire [Spi]. The central control server moves a testbed node by sending a movement command to the testbed node's RouterBoard, which relays a corresponding command to Spitfire over serial port. Eventually Spitfire issues the associated infrared code to instruct the testbed node's Roomba to move accordingly.

Roomba does not expose any API for controlling its movement. The only option

for controlling Roomba's movement remotely is its remote controller. Therefore, to allow a user to control Roomba movement programmatically and accurately, we improvise on the use of the remote controller. We use Spitfire, a universal learning remote controller, to learn the IR codes associated with each button press on the Roomba remote controller. Spitfire provides a GUI to perform this step manually with clear instructions on how to learn the codes. Once this step is completed, the Spitfire acts as a remote controller for the Roomba, and can issue the same movement commands as the factory-made remote controller. The code for each button is stored as part of a library in an EEPROM in the Spitfire. Although it is possible to program each Spitfire manually using the GUI, this process introduces slight deviation in the codes that are stored in the EEPROMs, resulting in deviations in the movement of the Roomba for the same command issued from different Spitfires. In order to eliminate this inconsistency, we program *one Spitfire manually*, and copy the contents of this Spitfire's EEPROM on to the EEPROMs of other Spitfires using a hardware chip programmer[3].

Spitfire is fitted with a serial interface using which it can connect to an external controller, in our case the RouterBoard. It also exposes APIs for issuing commands over the serial interface, accessing the codewords stored in the code library stored in the EEPROM. The Roomba controller agent on each node uses these Spitfire APIs to initialize the serial interface, and issues the IR signals for steering the Roomba. The distance traversed or the rotation by a Roomba is in multiples of the unit distance/angle traversed in response to 1 move signal.

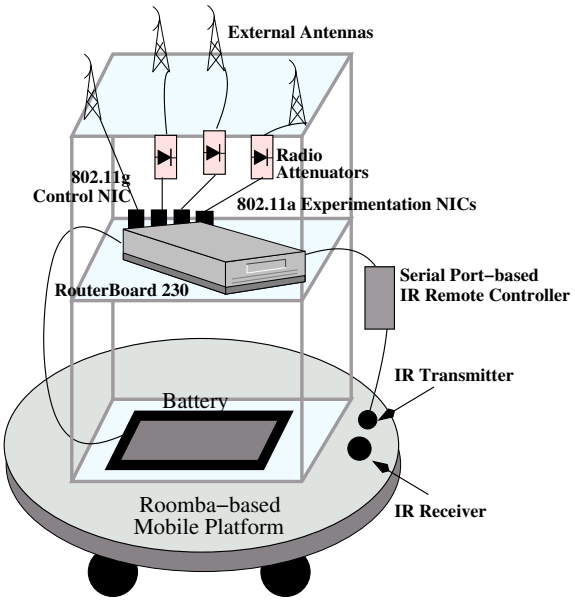

Figure 3.3: *A MiNT node comprises of a RouterBoard (RB-230) powered by an external laptop battery, and a Roomba robotic vacuum cleaner whose movement is controlled by a Spitfire Universal Remote Controller. The RouterBoard is equipped with 4 wireless NICs each connected to a separate omni-directional antenna via a radio signal attenuator.*

3.2.3 A Complete MiNT Node

The complete design of a MiNT node involves setting up a full-fledged computing platform that can act as the wireless node. Since the entire setup must be mobile, and be mountable on a Roomba, therefore it is imperative to choose a platform that has a small form-factor, is as light weight as possible, and must be energy efficient to run on battery power for a considerably long duration. Additionally, custom made platforms are avoided keeping in mind the cost constraint involved. After considering several options, we choose RouterBoard's RB-230 board as the hardware platform for the wireless networking device. RB-230 is a small-form-factor PC with a 266 MHz processor and runs on an external laptop battery. It also comes with a PCI extension board (RB-14), which allows us to attach 4 Atheors-based 802.11 a/b/g mini-PCI cards. As discussed earlier, each of these cards is connected to a 2 dBi external antenna through a 22 dB attenuator. This adds a total of 44 dB attenuation on the signal path from transmitter to receiver and thus makes it possible to deploy a 12-node MiNT prototype within a space of 132.75 inches X 168.75 inches (Figure 3.2). In addition to the fixed attenuation, the transmit power on the mini-PCI cards can also be altered to provide additional flexibility in tuning inter-node signal-to-noise ratio.

Figure 3.3 shows the current MiNT node prototype. There are two shelves mounted on the Roomba. The laptop battery and the Spitfire universal remote controller sit on the lower tier, while the RouterBoard based wireless networking device is on the top tier. The four external antennas are mounted on poles located at four corners. Detailed instructions for assembling a MiNT node is provided in Appendix A.2.

[3] this requires opening the Spitfire box, identifying the EEPROM chip storing the data, and using the chip programmer to program that chip.

3.3 Tracking System

The tracking system in MiNT is necessary to maintain the location and orientation information of node in real time, and display it through the GUI. To enable autonomous robot movement, the central control daemon must keep track of the current position and orientation of each testbed node. One of the simple ways to track position of a node is by using the odometry data that gives a feedback about the distance traversed by a node from a starting point, and the angle rotated with respect to a fixed direction. However, Roomba provides no API for collecting these data. Since we calibrate each forward and rotate step in the beginning, it can also be used to compute the odometry data using the move commands that are issued. But mechanical inaccuracies, difference in floor friction can lead to errors. Hence a more accurate locationing system for the nodes is required. One option is to use RF/ultrasound-based indoor local positioning systems such as Cricket [PCB00]. However, this option increases the per node cost, and introduces additional RF interference. Therefore, we choose an optical or vision-based position/orientation tracking system that only requires off-the-shelf webcams and color patches mounted on testbed nodes. The resulting tracking system is able to uniquely identify each testbed node, and pinpoint its (X, Y) position and orientation (θ).

Compared to general object tracking, the object tracking problem in MiNT is less complicated because of several simplifications that is possible specifically for MiNT. First, the lighting condition in the room housing the MiNT testbed does not change much. Consequently, it is not necessary to dynamically account for fluctuation in lighting condition once the color profiles have been calibrated for the initial lighting condition. Secondly, color patterns used to identify individual testbed nodes can be chosen such that they are different from the background color, in this case the floor's color. By including multiple colors in the patterns used

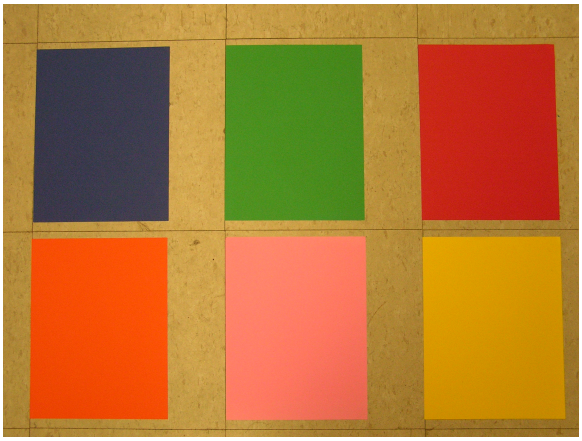

Figure 3.4: *The 8 different colors used in MiNT.*

to identify individual testbed nodes, this scheme can easily support hundreds of testbed nodes. Thirdly, placement of webcams that periodically take snapshots of the testbed nodes does not change once it is mounted.

Identification of colors uniquely is at the heart of the tracking mechanism in MiNT. We use the HSV (Hue, Saturation, Value) space to represent colors because the distribution of colors is more uniform, and the Hue and Saturation components are orthogonal to the Value (or brightness) component. Given images captured in the testbed arena, we compute the HSV profile of a number of colors, and eventually arrive at 8 colors that can be clearly distinguished based on at least one of the H, S, or V components. Figure 3.4 shows the 8 colors used in MiNT. Because the HSV profile for the same color may change substantially from one camera to another (due to difference in CCDs of each camera), we profile each camera separately.

In the choice of the camera used in the tracking system, we are driven by the cost effectiveness. It is possible to use high resolution cameras with wide angle lens that has a large viewport, but comes at a cost in order of thousand of dollars. Similarly, for high frame rates, it is possible to use hardware decoders that can provide frames of the order of 30 frames per second. Since we have an application that can work

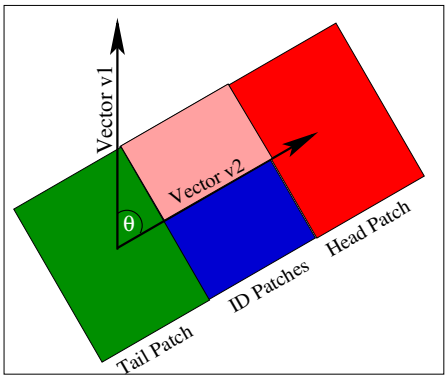

Figure 3.5: *The color patch on a node has a unique head and tail patch on all nodes. The vector from the centroid of the tail patch to the centroid of the head patch is used to determine the Roomba's direction, thereby computing a node's orientation in the testbed arena. The node location and identification are done using the center ID patches.*

with a lower resolution image and lower frame rate, we identified cheaper options that cater to the requirement. Commercial off-the-shelf webcams has a low resolution of 320x240 but comes at a price of $112 (retail price for Logitech Quickcam Pro 4000).

3.3.1 Tracking Mechanism

MiNT employs a vision-based tracking system where it must identify combinations of colors to uniquely identify a node and determine its location and orientation. MiNT associates a four-color pattern with each testbed node, as shown in Figure 3.5. The head and tail color patches are the same for all testbed nodes. So only the center patch, which consists of two colors, are used in node identification. In particular, the *location* of a testbed node is the centroid of the ID patch. The *orientation* of a testbed node is determined based on its direction, which in turn corresponds to the vector connecting the centroid of the tail patch to that of the head patch. Using

the same colors for the head and tail patches introduces redundancies that can guard against noises and simplifies the determination of robot orientation. There are two main steps in identification and locationing of a node: (a) finding the presence of a color patch in the snapshot image, (b) reducing parsing of each image by using inter-frame coherence in the images captured.

The color recognition algorithm used in MiNT is extremely simple and thus light-weight, and can do away with several image processing techniques, like edge detection. Once an image is grabbed, the pixels are scanned one by one in the scan line order. If a pixel of a known color is detected, then the pixels in its immediate 1-pixel neighborhood are checked for similarity and the color block is grown. It is possible to detect multiple blobs of the same color in an image captured by a webcam. This can happen if multiple nodes are captured by 1 webcam, in which case the head and tail patches will lead to blobs of same color. It can also be due to noise which can result because the color patch might fade in portions and lead to disconnected blobs for a single patch. The noisy case is handled by using a merging technique. For all the blobs of same color, it is checked if the centroids are within a distance less than a pre-defined *merge threshold*. The merge threshold is chosen such that it will guard against merging same color patches from different nodes, but will be able to compose all the blobs from the same patch into a single blob. The blob merging technique is aimed at making the system robust against noises that is quite common in such a system.

An optimization to improve the frame rate is to reduce the scanning of entire image for every cycle. Since we are only interested in the pixels corresponding to a node, the aim is to start the search in a captured image for a node where we most expect it in the next frame. In other words, with a knowledge of the state of the node between two frames, it is possible to exploit the location information to start the search closer to the node's current location. A bounding box is computed

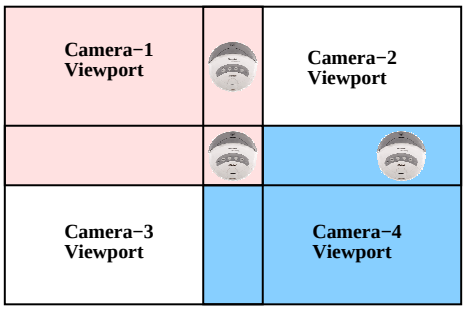

Figure 3.6: *The schematic shows the placement of multiple cameras such that viewport overlaps. The overlapped areas are such that at any point in time a node must be fully covered by a camera; none of the camera captures a partial image of the node.*

beyond which the node will not have moved, given the inter frame latency, and the speed of the node. This bounding box directs the search for the node in subsequent frames, thus reducing detection time.

It must be noted that a single webcam cannot cover the entire testbed arena. Multiple webcams are used to fully cover the entire arena for the testbed. A node naturally moves from the viewport of one camera to another camera. We noted that the complexity of the tracking algorithm increases unnecessarily if a node is partially covered by 2 different webcams. A simple design choice in the layout of the cameras takes care of this problem. The cameras are mounted such that the view ports of adjacent cameras overlap, and the overlapped area is large enough to cover a node completely. Hence a node is always fully covered by a camera; none of the camera captures partial view of the node, as shown in Figure 3.6.

3.3.2 Implementation and Tracking Setup

The implementation of the tracking system in MiNT splits the entire process in two stages, first each webcam captures an image of the space it is covering, and next, the

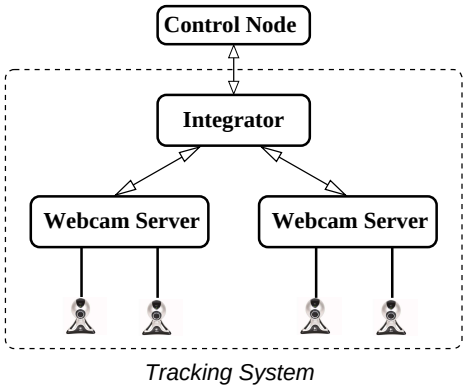

Figure 3.7: *The schematic shows the components of the tracking system. The webcam servers collect images captured by each webcam, and feeds to the integrator. The integrator translates the node positions to global coordinates and sends it to the control node for display in GUI.*

partial images of the testbed from each camera is merged so that the coordinates of a node is presented to the control server in global scale. There is a webcam server for capturing image from each webcam, and the integrator server sitting in between the webcam servers and the control server merges the data collected by the webcam server to generate the coordinates in the unified 2-dimensional space representing the testbed arena. Figure 3.7 shows the components in the tracking system. Multiple instances of the webcam server can be executed on a single machine along with the integrator. With increasing number of webcam servers running on the same node, the frames per second that is reported reduces due to the increasing processing overhead. In order to scale the system it is possible to split the entire system into a cluster of PCs.

In order to get the image that each webcam captures we need image grabbing APIs. Usually for most of these webcams image capture APIs are available in Windows platform. However, APIs on Windows platform suffers from poor scaling, and gives a very low frame rate even when just 2 webcams are plugged in.

VideoForWindows API is unable to instantiate more than one webcam on a single machine, whereas DirectShow API gives a low frame rate of 1.5 fps with only 2 webcams. After exploring several choices as above, we have chosen Camstream application [Cam] on Linux. Camstream uses Video4Linux [Vid] as the image grabbing API. Incorporating our tracking algorithm into Camstream allows us to capture frames at 15 fps with 6 webcams connected to a single Intel based machine with a 2.8 GHz processor. In our implementation, Video4Linux or v4l library calls are used by a user level application for grabbing the image from a webcam. The specific driver used for the Logitech Quickcam Pro 4000 webcams is Philips USB Webcam (PWC) driver.

The integrator application simply listens for a connection request from the control node. Once it connects to the control node, it establishes connection to the webcam servers, and starts collecting data periodically. It merges the data to compute the global coordinates and sends it back to the control node. When a node is transitioning from one webcams coverage area to another, the integrator communicates the change to both the webcams. This allows the webcam which is seeing the node for the first time to use the bounding box information provided by the integrator.

In our current prototype setup 6 webcams are mounted at a height of about 9.1 ft with each covering a floor space of 87 inches X 66 inches, with the total testbed arena of 132.75 inches X 168.75 inches. Each webcam has a resolution of 320 x 240, thereby each pixel corresponds to 0.075 square inch area. Factors that affect the accuracy of MiNT's color-based position/orientation tracking algorithm are the size of each color patch, the number of distinct colors used, the stability of lighting condition, optical noise in the patch boundaries, which might distort centroid computation. Given the patch size and the camera resolution used in the current MiNT prototype, a 1-pixel recognition error could potentially translate to

0.27 inch in location error, and 2.2 degrees in orientation.

3.4 Trajectory Determination

Unrestricted mobility of the nodes introduces the challenge of careful planning of node movement such that obstacles in the form of other nodes in a node's path is avoided during its displacement. MiNT's trajectory computation is based on a static trajectory planning algorithm, which computes a robot's path assuming the world is static, and a dynamic collision avoidance algorithm, which detects and resolves collision by fine-tuning pre-computed trajectories.

Given the current position and the target destination of a testbed node (TN), the control server takes a snapshot of the positions of other testbed nodes and treats them as obstacles in the calculation of the TN's trajectory. The static trajectory planning algorithm first checks if there is a direct path between the TN's current position and its destination. If such path does not exist, the algorithm identifies the obstacle closest to the source position, and finds a set of intermediate points that lie on the line which passes through the obstacle and is perpendicular to the line adjoining the source and destination and have a direct path to both the source and destination. If no such intermediate points exist, the algorithm finds a random intermediate point that is δ steps away from the obstacle closest to the source and is directly connected to the source, and repeats the algorithm from this new intermediate point as if it is a new source. The algorithm is shown in Algorithm 1.

In Figure 3.8, node N_1 is set to move from $A_{initial}$ to A_{final}. However, N_2, N_3 and N_4 block the direct path between $A_{initial}$ and A_{final}. The trajectory planning algorithm first figures out that N_3 is the obstacle closest to $A_{initial}$, and then computes the intermediate points $P_1, P_2, ..., P_6$ to search for 2-hop paths to A_{final}. Because the paths L_1 and L_2 are partially blocked, the algorithm eventually chooses path L_3,

Algorithm 1 Node Trajectory Determination

```
for (∀ nodes marked for mobility) do
  obstacle ← Nearest obstacle on direct path between A_initial and A_final
  if (obstacle == 0) then
    // There is a direct path to the destination
    Generate Roomba moves
  else
    Determine intermediate points (P_1, P_2, ..., P_n) on lines ⊥ or at angle θ to
    direct path passing through the nearest obstacle;
    Check for direct path between A_initial and any of (P_1, P_2, ..., P_n) ;
    Check for direct path between any of (P_1, P_2, ..., P_n) and A_final ;
    if (∃ 2-hop direct path via P_i) then
      Generate Roomba moves from A_initial to P_i ;
      Generate Roomba moves from P_i to A_final ;
    else
      if (∃ direct path from A_initial to some P_i) then
        Generate Roomba moves from A_initial to P_i ;
      else
        Move δ steps in a random direction away from nearest obstacle;
      end if
    end if
  end if
end for
```

which passes through the intermediate point P_3.

In addition to static trajectory planning, MiNT also requires a dynamic collision

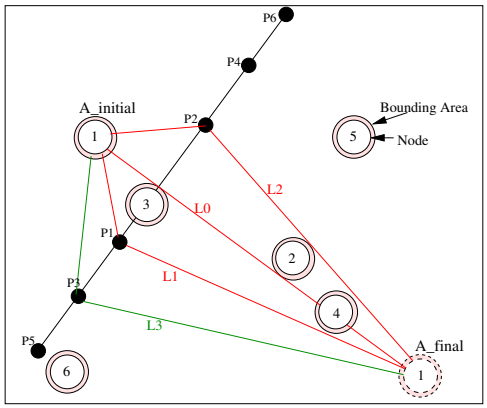

Figure 3.8: *Finding the trajectory from N1's current position $A_{initial}$ to A_{final}. As nodes N2, N3 and N4 block the direct path, the algorithm tries to identify an alternate 2-hop path to move N1 from its current source to its destination.*

avoidance algorithm because testbed nodes could be moving and the robot movement is not perfect. Given a snapshot of the testbed at 15 fps in the current prototype, MiNT performs a proximity check for each testbed node. If any two nodes are closer than a threshold distance, both of them stop, a new path is re-computed for each of them, and the algorithm moves them on their new trajectory one by one. In the event that two nodes indeed collide with each other, the algorithm again detects it through a proximity check and stops the nodes immediately. In this case, the algorithm also recomputes a new path for each of the two nodes, and moves them one by one.

One problem with Roomba is that its movement is not very accurate, which could also lead to dynamic collision. Its forward movement is 5 inches per step, and the pivot movement varies from 4 to 5 degrees per step. Furthermore, a Roomba only allows three types of movement: forward, clockwise turn and counter-clockwise turn. Hence backward movement is implemented by a turn of 180 degree followed by forward motion. The inaccuracies in Roomba movement

necessitate constant correction. Additionally, MiNT's object tracking errors also contribute to position inaccuracy. Hence, after applying every sequence of 5 move commands on a testbed node, its trajectory is re-computed till it reaches the destination point.

3.5 24x7 Autonomous Operation

A key challenge in the design of MiNT is how to render the testbed self-manageable and providing uninterrupted 24x7 operation. This ideally means that a node should be self-administered with respect to routine operations, like recharging the batteries for the nodes; and it should be self-recovering in the face of faults. Since each testbed node is battery powered, the batteries must be recharged periodically. Usually battery charging is a manual process that requires the administrator to take discharged nodes to charging stations [JSF+06]. In contrast, MiNT supports automatic recharging of node batteries and imposes zero manual charging overhead. In a setup involving large number of nodes, and executing any user code and unchecked kernel modules, node crashes are not unusual. A node must respond to this by detecting the node failure and be able to recover to a clean state. In this section, we present the auto-recharging mechanism, the residual battery capacity estimation mechanism, and the re-charge scheduling policy. Finally, we discuss how we recover from node crashes resulting from bugs in protocols under test.

3.5.1 Auto-recharge Mechanism

Roomba provides a docking station to charge its batteries. The Roomba docking station emits an IR beacon that is received by a Roomba over a distance of around 5 ft. When a Roomba's battery power drops below a threshold, it starts looking for a beacon emitted by the docking station and uses the signal to home into the

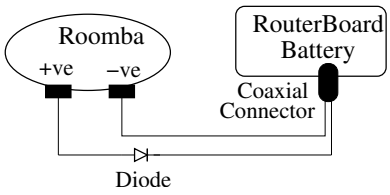

Figure 3.9: *The auto-charging circuit for charging the wireless node battery when the mobile node docks itself into the docking station for recharging.*

docking station for recharge. Unfortunately, Roomba's built-in battery cannot be used to directly power the RouterBoard. Hence, we use a separate universal laptop battery to power the RouterBoard. To recharge the RouterBoard battery along with the Roomba battery, we connect the RouterBoard battery to the charging tip of the Roomba battery as shown in Figure 3.9. This allows both the batteries to be charged simultaneously from the same docking station. A diode connected between the Roomba battery and the RouterBoard battery ensures that the batteries are not drained by each other.

In order To keep the testbed running on a 24x7 basis, nodes with low residual charge must be scheduled for recharge on every charging cycle. Unfortunately, neither the Roomba nor the RouterBoard battery provide any API for probing the residual battery capacity. Hence the residual charge on a testbed node is estimated based on profiling of the batteries and the node's usage [4].

For each node, the residual charge on the Roomba's internal battery and the RouterBoard's battery are estimated separately. Specifically, the residual charge on Roomba's internal battery is estimated using the equation:

$$R_{roomba} = I_{roomba} - N_{roomba} * U_{roomba} \tag{3.1}$$

[4]With the introduction of Roomba Serial Command Interface since October 2005, it is now possible to retrieve the status of the Roomba battery and have a more accurate measure of residual charge (discussed later in section 3.7

where R_{roomba} and I_{roomba} are the residual and the initial charge on Roomba's battery. N_{roomba} is the number of movements performed by Roomba, and U_{roomba} is the energy consumed per movement. Although, a Roomba battery drains even when Roomba is not moving, the drainage is negligible.

Similarly, the residual charge on RouterBoard's battery is estimated as:

$$R_{board} = I_{board} - T_{board} * U_{board} - N_{disk} * U_{disk} - N_{packet} * U_{packet} \quad (3.2)$$

Here, R_{board} and I_{board} are the residual and the initial charge on RouterBoard's battery. T_{board} is the amount of time RouterBoard has been on, and U_{board} is its idle power consumption per unit time. N_{disk} and U_{disk} are the number of hard disk operations performed by the RouterBoard and the energy consumed per disk operation respectively. Finally, N_{packet} and U_{packet} are the number of packets sent/received by the RouterBoard and the energy consumed per network operation respectively. The energy consumed by each IR transmission is negligible.

A node cannot be used for experimentation while it is being charged. However for 24x7 operation a set of nodes must be operational at all times. Hence, a set of spare nodes are provisioned in the testbed. Specifically, the testbed uses $n + m$ nodes, where n nodes are used in the experiments at any time while m nodes are being recharged. If c is the average charging time for a node and d is its average discharging time, then by maintaining $(m > n * c/d)$ spare nodes, the testbed can be run without any downtime. Neither the Roomba's nor the RouterBoard's battery suffer from any memory effect due to incomplete charge/discharge cycles. Therefore, the actual re-charge scheduling algorithm is straightforward: upon every re-charge cycle, just dock the m least charged nodes out of $n + m$ testbed nodes for re-charging.

3.5.2 Node Crash Recovery

The other aspect of 24x7 operation is how to deal with node crashes due to bugs in the kernel modules. We utilize the 2 hardware watchdog controllers that are equipped on each RouterBoard. A software daemon periodically writes certain bytes to an I/O port indicating to the watchdog controller that the node is still alive. In the event of a node crash, the control daemon stops writing to the I/O port, and the control server detects it after a timeout. The controller then automatically reboots the node. To ensure that a reboot restores the original kernel image, no user-specified module is loaded at boot-time. This way a crashed node can always automatically recover within a fixed time.

3.6 Limitations of MiNT

The key feature of a MiNT testbed is its ability to limit the signal propagation range between two nodes to within a few feet through use of attenuators. However, the approach of miniaturization through attenuation has certain limitations. This section presents a word of caution on the pitfalls of MiNT and explains their potential impact on the final outcome of experiments.

The most prominent change in MiNT from a typical full-scale testbed is that in MiNT the radio signals are attenuated at the transmitter and the receiver ends. As we are not placing the core nodes in a noise-free environment, the nodes operate in presence of external noise sources, like microwave oven, cordless phones, and other interfering channels. The RF signals from these noise sources are attenuated only at the receivers, leading to a phenomenon of *selective attenuation*. The noise sources are less attenuated than the real transmissions. Additionally thermal noise at the receiver is unattenuated because it does not go through the receiver antenna. Since the attenuation of signal is more than that of the noise, one might suspect that the

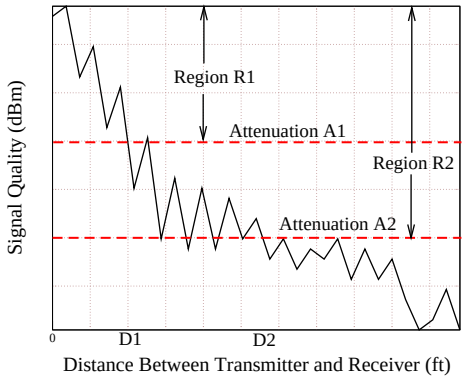

Figure 3.10: *Representation of relationship between signal quality and distance. Adding radio signal attenuators pushes the X-axis of the graph up, effectively reducing the extent of signal quality variation. At attenuation A1, the graph is confined to region R1. In region R1, the signal becomes 0 when the distance between transmitter and receiver is greater than D1.*

signal-to-noise ratio (SNR) for a link in MiNT is lower than that of an unattenuated testbed. However, this effect can be overcome by reducing either the attenuation level or the distance between the nodes.

In MiNT, since the nodes (and hence the antennas) are placed in close proximity of each other, *the receiver is in the near-field zone of the sender*. This is unlike a full-scale testbed, where the nodes are typically placed far from each other, hence the receiver is usually in the far-field zone of the sender. This difference is inherent to MiNT approach due to shrinking of the space.

Multi-path effects in signal propagation lead to small-scale variation in the signal strength. A qualitative representation of this variation of signal quality with distance is shown in Fig 3.10. Between two points, say 0 and D2 there are multiple crests and troughs in the signal quality. By adding the attenuator, we are effectively pushing up the X-axis in this graph by the dBm value of attenuation. As a result

of this, the number of crests and troughs between the same two points, 0 and D2, is *smaller* than that of the non-attenuated case. Constructive and destructive interference resulting from the multi-path effects are dependent only on the frequency of the signals. Hence a solution to this problem is to scale down the frequency of the signals which would make the number of crests and troughs same. However, changing the frequency would change the properties of the wireless medium under test, and hence is not a viable solution. This limitation impacts the mobility-related experiments where the extent of signal quality variation encountered by a mobile node in MiNT will differ from that of full-scale testbed.

Finally like any other testbed, lack of control over physical parameters leads to *non-repeatability* of experiments on MiNT. The inability to control external factors affecting signal propagation is the source of this problem.

3.7 Roomba Serial Console Interface (SCI)

In a recent development (circa October 2005) Roomba manufacturer iRobot has announced roll out of a new version of Roomba that contains an electronic and software interface for controlling and modifying Roomba's behavior, and remotely monitor its sensors. This interface is known as the Roomba Serial Command Interface or Roomba SCI [Roob]. The SCI provides commands to control all of Roomba's actuators, like the motors, as well as request sensor data. Roomba SCI simplifies several aspects in the design of MiNT, with an associated reduction in the overall cost. In this section, we present this new programming interface for Roomba, and how it affects the design of the individual components discussed in the earlier sections. One needs to connect the RouterBoard serial interface to the serial interface on the Roomba (mini-DIN connector).

SCI simplifies the design of mobility setup that uses the universal learning remote controller, Spitfire. SCI provides the API to issue the move commands directly to the Roomba. Specifically, the API that can be used is *Drive*. The command takes 4 data bytes, where the first two data bytes specify the average velocity of the drive wheels in millimeters per second (mm/s), and the next two bytes specify the radius in millimeters at which the Roomba should turn. This command gives the complete flexibility to steer a Roomba forward, turn it at any arbitrary angle, even do a forward motion while turning, as well as do a reverse motion. Overall, use of this API achieves two purposes: it eliminates the need for a hardware device leading to a cost reduction of $120 per node; secondly, it opens the possibility of exploring different velocities for node movement.

SCI could also be used in increasing the robustness of our tracking system. There are two APIs, *viz. Distance and Angle* in SCI, which provides the distance the Roomba has traversed in millimeters and the angle it has turned through since the last call to these APIs. If rapid movements are being requested, then it is required to make frequent calls to these APIs to get accurate values as the values are capped at its minimum or maximum. It is also worth noting that the reported values may be inaccurate due to floor friction variation and mechanical differences. Hence, this can only be used to verify the results from the vision-based tracking, but not eliminate the tracking system currently used in MiNT.

For enabling 24x7 autonomous operation we had to engineer the charging circuit on the Roomba so that the battery powering the RouterBoard is simultaneously charged along with the Roomba battery. With the introduction of SCI, this has been greatly simplified because two pins in the mini-DIN connector provides unregulated power from the Roomba battery. This can now be used to power the RouterBoard, eliminating the need for an external laptop battery. However, it should be noted that since this is unregulated power, it is may be necessary to introduce a voltage stepper

before it can be used for powering the RouterBoard. Since in this setup we can use a single battery for powering the complete node, therefore it becomes simpler to enable 24x7 operation. SCI exposes an API, named *Charge* which gives the current charge of Roomba's battery in milliamp-hours (mAh). In our case, the Roomba battery is the node battery. Once we can get a feedback on the residual charge it is possible to use another API, called *Force-Seeking-Dock* to send the node for charging. The Force-Seeking-Dock command causes the Roomba to immediately attempt to dock if it encounters the docking beams from a Home Base.

In summary, the opening up of the SCI APIs leads to a total savings of around $300 per node in building a MiNT node. It also simplifies some of the steps in building a node, like programming a Spitfire device, engineering the charging circuitry.

Chapter 4

Management and Control of MiNT: The Tool Suite

A hardware infrastructure requires a comprehensive software tool suite to be truly useful. For using a testbed effectively, there are several software tools that are essential. For example, remote access to the testbed necessitates a user interface that provides visibility of all the resources in the testbed, as well as flexibility to control those for individual experiments. Similarly, capability to allow execution of different experiments requires a framework to enable experiments with diverse requirements. We have designed a set of tools to allow such functionalities in order to make MiNT an easily usable testbed for wireless experimentation. In this chapter, first we revisit the components of MiNT, emphasizing the software building blocks, and how all of them fit together to form a tool suite for a wireless testbed. Detailed description of three main software components, *viz* the user interface for MiNT, the experiment execution framework used in MiNT, and a tool for efficient debugging of protocol implementations in a distributed environment, follows later.

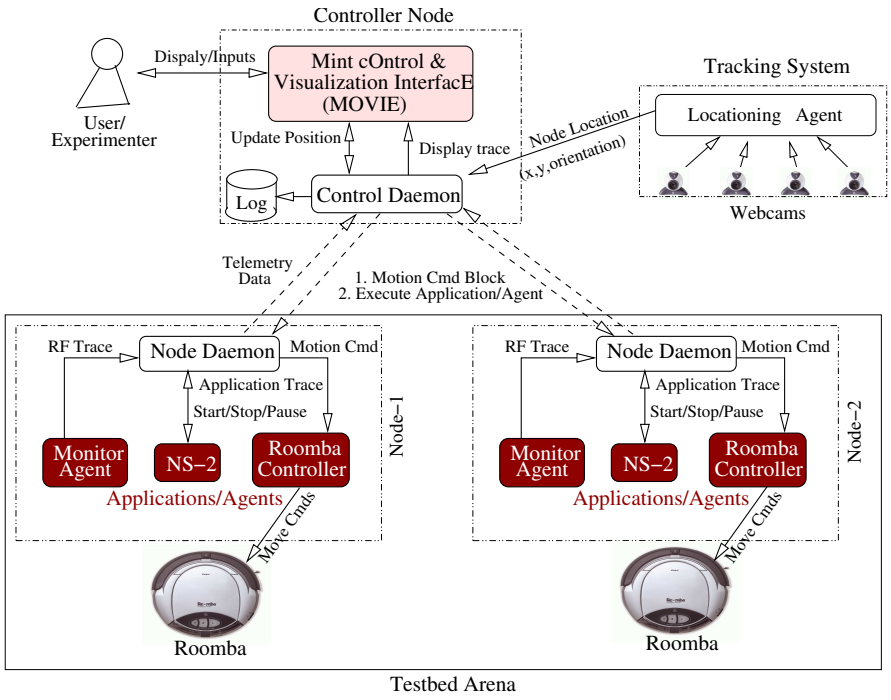

Figure 4.1: *Overall MiNT architecture. The* control daemon *running on the control server collects inputs from the tracking server and the user, and controls the movement of mobile robots. It also includes the MOVIE interface for monitoring and control. Each testbed node corresponds to a Roomba robot and has a* node daemon *running on it, which communicates directly with the control daemon over a dedicated wireless control channel that is non-interfering with the channels used for the experiments. The core testbed nodes communicate with the peers using wireless NICs that are connected to low-gain antennas through radio signal attenuators. The* vision-based tracking server *periodically captures images of testbed nodes, and processes them to derive the location of each testbed node.*

4.1 Software Building Blocks for MiNT

There are three main building blocks for the MiNT infrastructure, *viz.* the control node, the testbed nodes, and the tracking subsystem. Each of them runs several

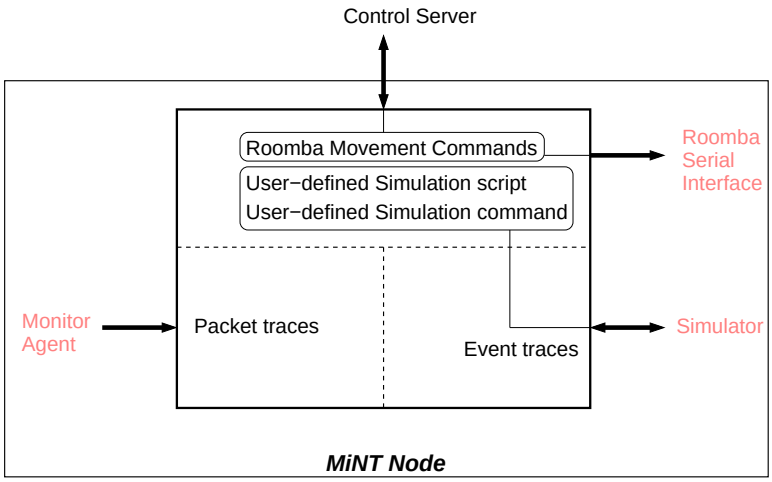

Figure 4.2: *The software architecture of the node daemon.*

software that come together to form the software setup for MiNT. Figure 4.1 highlights some of the important software components that are running on the control node, each testbed node, and the tracking subsystem. In this section, we will discuss the role of each of these components, and present how they communicate with each other.

The tracking subsystem is a relatively independent unit in the entire MiNT setup. The main purpose of this unit is to feed the current position and orientation information of each node moving around in the testbed, as described in detail earlier in Section 3.3. The software component running on the tracking server is a locationing agent that interprets the image data collected by the grid of webcams spread over the testbed arena to compute the coordinates of each MiNT node. The details of how the tracking server works is previously explained in Section 3.3, and is mentioned here for completeness.

Each MiNT node runs several software modules that help in running and controlling experiments on the nodes, as well as collecting statistics for further analysis. A *node daemon* runs on each node and acts as the single point of interface for each

node. The node daemon comes up when a node is booted. Each MiNT node is configured to acquire a DHCP address from a DHCP server in the testbed arena using one of its wireless interfaces which has non-attenuated connectivity. Once there is connectivity, the node daemon uses the preconfigured IP address of the control node to establish connection with the control node. The node daemon stays connected to the control server throughout, and waits for commands generated as a result of user activities. The node daemon also opens socket connections to the simulator running on each node, and the local packet monitoring agent. The complete software architecture for node daemon is shown in Figure 4.2.

The different messages received by the node daemon from the control server are, *Move* commands for issuing the user activated move instructions coming from MOVIE. Each move command is communicated through a serial interface to the Roomba controller module. The *Roomba controller* is a serial interface to a Spitfire device that controls the physical movement of the Roomba. The other commands from the control node are related to simulation: it sends the simulation file name, and the commands to control the simulation, like play, pause, stop commands for simulation. The node daemon uses signals to control the simulation based on the specific simulation command received from the control node.

The control node is the single point of control and information gathering for the testbed. It collects data from, and distributes data to, three entities, the tracking subsystem, the testbed nodes, and the user. The control node runs a *control daemon* that acts as a single point of communication for all the other software modules running on the control node, and for communicating with other modules outside. The software design of the control daemon is shown Figure 4.3. As soon as the controller node comes up, the control daemon establishes connection with the nodes present in the testbed by sending a broadcast advertisement packet. It keeps sending this packet periodically so that testbed nodes can be added any time. The control

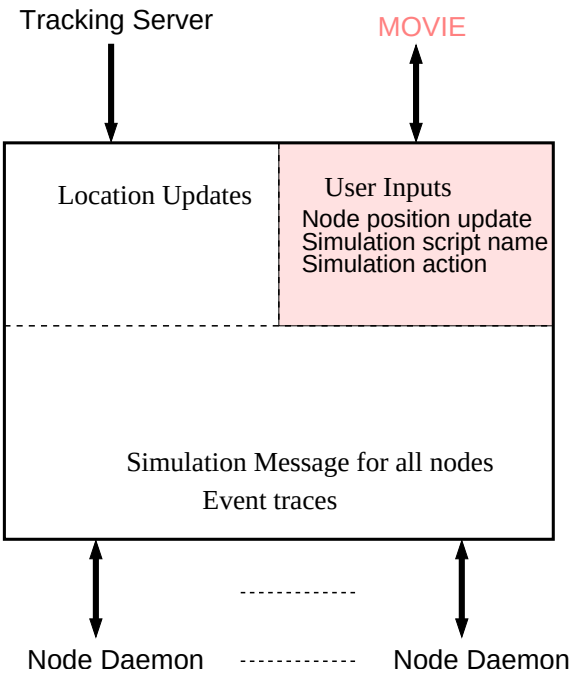

Figure 4.3: *The software architecture of the control daemon.*

daemon also receives continuous feeds of the location of each node in the testbed from the tracking subsystem.

The other important piece of software on the control node is the graphical user interface for MiNT, called MOVIE. MOVIE provides a one-stop portal for visualizing all the resources present in the testbed, as well as gives the flexibility to the user to change node configurations and control experiments. The only view of the testbed that a remote user gets is through this interface, thus MOVIE acts as the *eye and hand* of the testbed. Details of MOVIE are discussed in the Section 4.2.

The software architecture of MiNT is designed in such a way that the control daemon and the node daemon act as the gateways for communication among the different software entities residing at different places. All communication is done using message passing interfaces, with different message types being used to communicate to different modules and trigger different events. The advantage of this

approach is that it is relatively simple to enhance the model with new software entities as and when it is required to be added either to the control node or the testbed nodes.

4.2 MOVIE: Mint cOntrol and Visualization InterfacE

Existing wireless network simulators and testbeds lack capabilities for real-time visibility into the detailed dynamics of protocols under test. When it comes to experiment control, none of the commonly available tools expose sufficient interfaces for finer control over experiment and the flexibility to steer the experiment towards a fruitful direction. We have designed MOVIE, Mint cOntrol and Visualization InterfacE, to be a true "eye" and "hand" of the testbed allowing complete monitoring and management capabilities respectively for all resources present in MiNT through a single unified view (Figure 4.4). Besides being a network management interface, MOVIE also provides the interface to control experiment execution. Thus, MOVIE is an integrated network and experiment management system. It provides *real-time* display of individual node positions and inter-node signal-to-noise ratios (SNRs), protocol specific state variables, node/link liveliness, pair-wise end-to-end routes, network traffic load distributions. In terms of management of the testbed, MOVIE allows control over all node level parameters (*viz.* transmit power of a wireless NIC, that are exposed by the card as APIs), as well as individual links (enabling or disabling a link for communication through packet filtering). As an experiment control interface, MOVIE allows user to manage an experiment from start to finish, beginning with configuration of the testbed topology, followed by loading of simulation scripts or protocol modules, and finally collection of the experiment results in

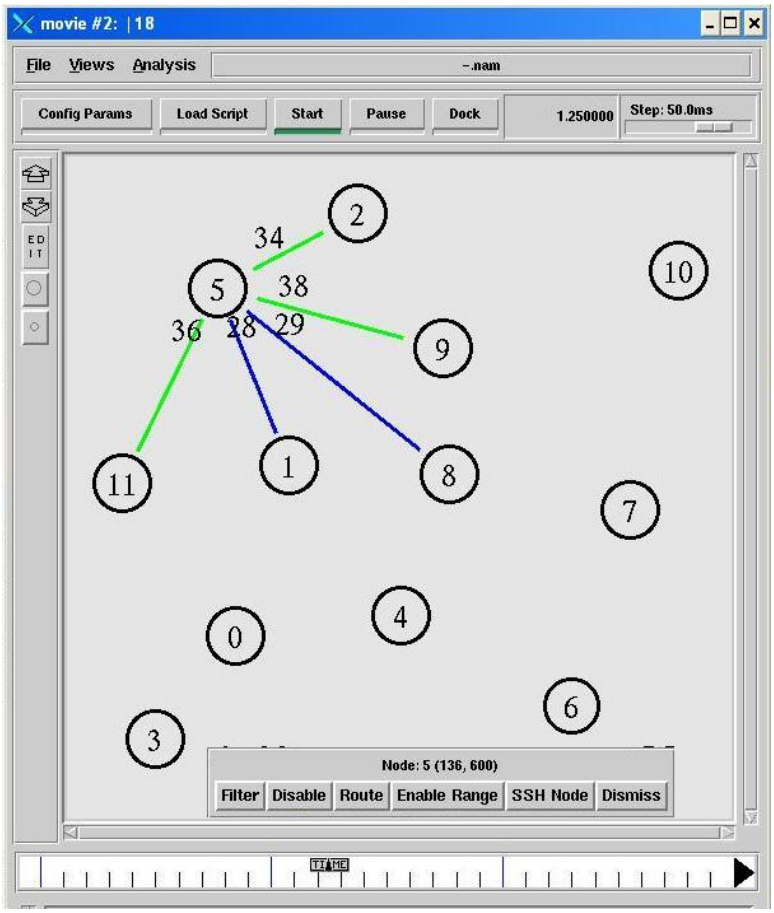

Figure 4.4: *MOVIE GUI acts as the front-end to the testbed, and supports all management, control, and visualization functionalities. Each node icon represents the actual position of a physical node in the testbed (as shown in Figure 4.7). Nodes are physically moved by dragging the corresponding icons in the GUI. The number on each link represent the signal quality for the link in that direction. MOVIE can be used to set the network-wide parameters, and override them on a per-node basis.*

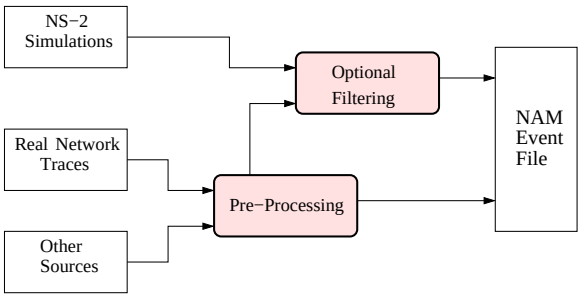

Figure 4.5: *A block diagram of components in NAM.*

an online or offline manner. Combined with a real-time view of the progress of the experiment, and the flexibility to pause simulation runs and rollback states, MOVIE virtually allows the user to control an experiment at a much finer granularity than previously allowed by any experiment management system. We present some of these advanced experiment control features in Section 4.3.

4.2.1 Network Animator (NAM) Preliminaries

The graphical user interface for MiNT is derived from Network Animator (NAM) [EHH+99]. NAM is one of the earliest network visualization tools, and is most widely used in conjunction with the network simulator *ns-2*. NAM uses a Tcl/Tk interface and is designed for animating time-indexed network simulation events which are stored in file as output of a simulation run, or even from real network traces (this requires some preprocessing). Figure 4.5 [EHH+99] shows how NAM event file is generated. The important steps in the design of NAM is determining the topology layout, followed by display of the time-indexed network events in a manner that visualizes the progression of the protocol activity. NAM also supports multi-resolution views into the event dynamics, as well as setting of monitors for tracking protocol-specific states. All inputs to NAM is a list of

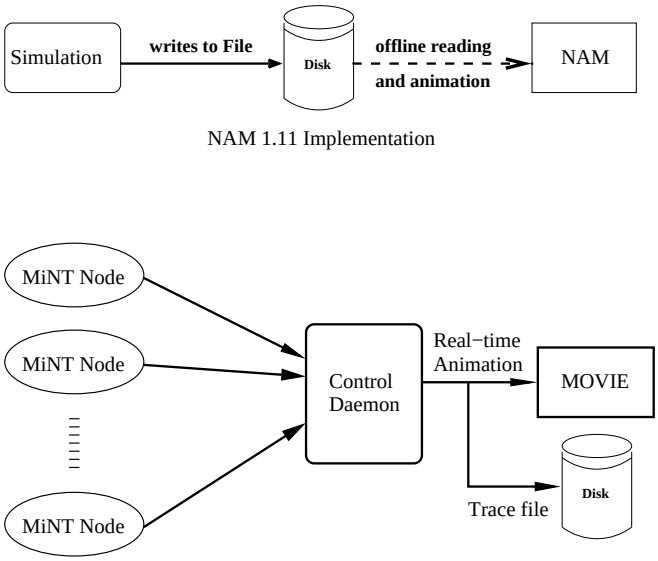

Figure 4.6: *The schematic shows the difference in the implementation of NAM and MOVIE.*

$< attribute, value >$ pairs. Based on the $attribute$ type, NAM displays different objects, like nodes with node id and links with link characteristics. NAM also supports some basic editing functions for scenario generation through addition of nodes, links and protocol agents in the NAM interface; corresponding simulation script is auto-generated.

The implementation of NAM (version 1.11) follows a very simple design, depicted in the block diagram of Figure 4.6. A NAM event file is generated either as a result of a simulation run, or through pre-processing network event traces. This trace file is read by the Network Animator as an input for displaying the progression of events after an experiment. Current implementation of NAM does not support an online display of events.

We have augmented NAM in several ways in order to come up with MOVIE. First of all, we modified NAM to be able to display events in real-time. The

events are generated at multiple nodes in the testbed and must be displayed through MOVIE. Each testbed node feeds the data to the control daemon residing on the control node via the node daemon in real-time. At the beginning when the control daemon starts, it opens a pipe to write to a buffer, which is a circular buffer capable of storing a configurable number of events. Instead of reading from a file, NAM is modified to read from this buffer, and displays the events in real-time. There is however a delay in the events actually happening in the testbed and those that are shown in MOVIE because of transmission latency from the testbed nodes, and some preprocessing involved before writing the data into the buffer. The pre-processing involves sorting the events into a time-indexed order at the control daemon, as well as, writing the events to a file for offline animation. Figure 4.6 shows the design change to NAM for coming up with the real-time display feature for MOVIE.

MOVIE also enhances NAM with several features related to visualization of statistics, management of the overall setup, and control of the experiments. Some of these are display of link strength, routes; exposing several interfaces for extensive control, like managing the resources through this single interface; and designing new interfaces for controlling experiments, like breakpointing feature. These features and their implementations are discussed in detail in the subsequent sections.

4.2.2 MOVIE Visualization Features

In order to display in real-time different entities, such as *nodes, links, and routes* present in the testbed, MOVIE requires a periodic update from the testbed nodes. The testbed nodes in turn must capture the packets exchanged in order to gather the related data about the nodes, links and routes. In MiNT, each testbed node captures all the packet exchanges using one of the three interfaces that is not used for experiment. This interface sniffs at the RF level all packets in its neighborhood. An important point to note here is that, covering the entire transmission domain of

Figure 4.7: *The webcam shot of the testbed. The color patch on each node uniquely identifies the node as well as tells its position and orientation.*

all nodes requires multiple nodes to monitor the traffic [YYA04]. There are two ways to set up the monitoring infrastructure: first approach is to let the experiment nodes themselves perform the monitor functionalities and sniff the packets in their respective neighborhoods; the second approach is to keep the monitor nodes and the experiment nodes separate. The second case requires careful placement of monitor nodes for a complete coverage of all transmissions, and is known to be a difficult problem; on the other hand, all MiNT nodes being equipped with multiple cards, it is possible to dedicate one card on each node for the sniffing role. The next step is to aggregate the packets collected in order to recreate the event dynamics in the testbed. Traces from all nodes are shipped to a central node and merged based on timestamp [YYA04]. This entails that all monitor nodes must be time synchronized at the start of the experiment. Same packet can appear in traces collected by multiple monitor nodes. During aggregation, these copies are culled to get a unified trace of the transmitted packets. Keeping duplicate packets however provides an useful piece of information. It provides information about all nodes that are in the same collision domain while a transmission is in progress.

Next we present several attributes that can be visualized in MOVIE. Each

testbed node is represented by an icon in MOVIE. Each node icon corresponds to the actual position and orientation of a node in the testbed arena, as shown in Figure 4.7. The accurate node position is provided by periodic feedback from the tracking server, and maintained on the control node. Double-clicking on a node icon opens a window that displays various *node attributes*. These include network card configuration, such as MAC address, radio channel, and BSSID. The window further displays the residual charge on the node batteries as estimated by the algorithm discussed in Section 3.5.1. Each of these are static values and maintained on each node separately. Using SNMP queries, the values for all these parameters are easily retrieved from each node.

Right-clicking on the node icon displays its hearing range neighbors, while left-clicking on it displays the node's interference-range neighbors. While setting up a topology for a wireless experiment, an important step is to understand the interference relationship among the neighbors. MOVIE can highlight both the communication and sense range neighbors for any specified node. Determination of hearing range neighbors is straightforward: The chosen node sends out a broadcast ping. All the neighbors that respond to the ping request are the hearing range neighbors of the chosen node. Determining interference range neighbors is relatively trickier: A node may not normally hear the transmissions from its interference range neighbors, but can sense their transmission. For 802.11a transmissions at $6\ Mbps$, if the transmit power of a card is increased by $2\ dBm$, then the sense range neighbors become hearing range neighbors. Hence, in order to determine sense range neighbors of a node, the transmission rate is set to $6\ Mbps$, wireless card transmit power is increased by $2\ dBm$, and then a broadcast ping is sent. The nodes that respond are the sense range neighbors for this node at the default transmission rate and transmit power.

Several new attributes are displayed in MOVIE in addition to the ones already

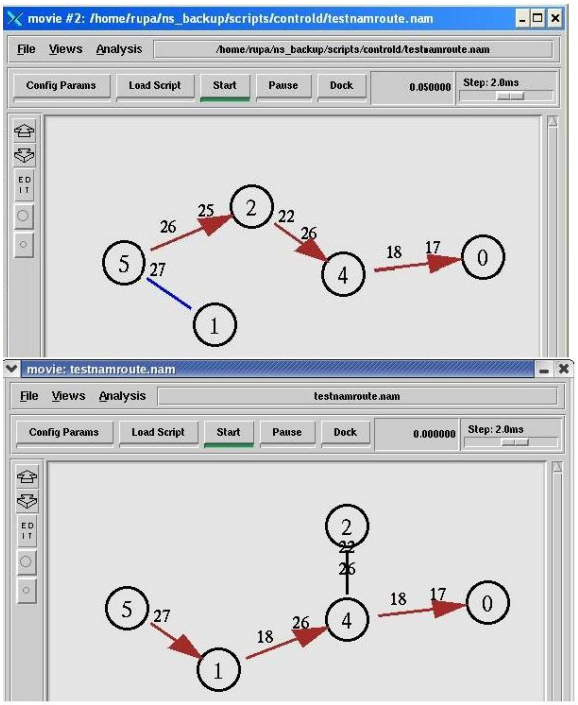

Figure 4.8: *The changes in multi-hop routes are displayed in real time in MOVIE. In the figure, the event of route switching from node 2 to node 1 is sent to the control node, and displayed in MOVIE in real time.*

present in NAM. The events in NAM are displayed based on a attribute list that is predefined. The new types and the attributes are added to this file, and are parsed with the same parsing engine as used by NAM. MOVIE displays *link attributes*, like signal quality, error rate, and traffic load on the link. These characteristics are measured in a passive manner so that measurements in an experiment are not perturbed. To measure the link error rate, each packet is stamped by a unique monotonically increasing sequence number. On reception of a significantly large number of packets, the number of lost packets is computed. This value is updated periodically on each node giving a measure of the loss rate on a specific link.

Display of *multi-hop routes* helps in understanding the routing dynamics during

an experiment. Protocol debugging can be much simplified through visualization of multi-hop routes discovered by the routing protocol (like AODV) in use. Each node maintains all the route table updates and periodically sends them back to the controller node. Changes in routes are also displayed in real time through MOVIE, as shown in Figure 4.8.

During development of new protocols, it is often necessary to view the changing values of different *protocol-specific attributes*, such as TCP's congestion window. MOVIE can display arbitrary attribute values as long as they are exported by the protocol developer as attribute-value pairs. For ns-2 based protocols, the attribute can be simply exported as part of the NAM file. For real implementations, we use the proc filesystem interface to export various attributes. For new protocols that plan to use the MiNT infrastructure, the values can be directly written in an attribute-value format. For older protocols, that do not log the data in the desired format, an adapter must be installed that converts the results from the files into the attribute-value format suitable for MOVIE. These values are then dispatched for display to MOVIE running on the central node.

4.2.3 MOVIE Control Features

MOVIE front-end (Figure 4.4) is designed to allow users to configure the testbed on an experiment-by-experiment basis. It provides all the necessary controls, collects detailed information from the testbed, and gives real-time status update to the users. Control activities of a user generate downstream data flow from MOVIE to the nodes. Visualization functions feed data into MOVIE for display. This subsection discusses the main control features supported by MOVIE. A discussion of other advanced control features, like pausing experiments, and rollback of simulation runs, are discussed in Section 4.3 after we explain the hybrid simulation technique.

One of the most important requirements for setting up the testbed is the ability to

configure each node. For each testbed node, the user can set various configuration parameters such as card transmit power, retry count, sensitivity threshold, and RTS threshold. Most of these parameters are set using standard wireless card API. The sensitivity threshold is the only one that is set by directly altering a card register. The user is provided a list of items to set the values of the different parameters. The control daemon then transmits each of the parameters to individual nodes. On receiving the value of these parameters, the node daemon makes calls to the wireless APIs to set each of these parameters.

Another key aspect in setting up an experiment is *topology configuration*. MOVIE allows a user to position the nodes at desired locations in the testbed by dragging the corresponding icons in the GUI. The movement of a node icon generates a destination point and triggers trajectory computation on the controller node (discussed in Section 3.4). The controller node then issues *move* commands to the node daemons on the corresponding nodes. The exact mechanism of how a Roomba movement is actuated based on a user specified signal is presented in detail in Section 3.2.2.

Given a certain placement of nodes, the node density can be altered by changing the transmit power level of the nodes. Further fine-tuning of topology is possible by selectively disabling individual links and routes. Links are disabled through use of MAC filtering function that drops all packets going from a specified source to a destination. Route disabling is done by periodically deleting the corresponding route table entry from all the nodes along the path.

4.3 Hybrid Simulation

An important goal in designing MiNT is to make it usable for running a variety of wireless experiments on it. A large section of wireless research till date has been

driven by simulations. Hence our objective is to provide a software environment that allows running these simulation experiments on MiNT with minimal modification. The value addition that MiNT provides to a simulation experiment comes from the use of realistic settings. For this purpose, we have developed *a hybrid simulation environment* that allows the execution of legacy simulation scripts, as well as new experiments; the advantage is that the experiment is exposed to real setups as opposed to modeled parameters. In this section, we discuss the hybrid simulation technique, followed by its detailed implementation for a specific wireless network simulator *ns-2*.

4.3.1 Hybrid Simulation Overview

The drawback of pure simulation in capturing actual behavior of protocols in a real setting is often attributed to the lack of detailed models for the physical layer properties, such as signal propagation and channel error characteristics. A common practice in most academic research to date is to use simplistic physical layer models. This is one of the prime reasons for the lack of simulation fidelity. Often coming up with a detailed model is an extremely difficult task due to the presence of a multitude of parameters; the simulation time for an experiment also grows significantly with increasing complexity. With growing interest in cross-layer designs of protocols, it becomes imperative to provide accurate results at different layers in the protocol stack. Hybrid simulation alleviates some of these problems faced by pure simulation.

We define hybrid simulation as a technique where some layers of the simulator's protocol stack are replaced with real entities. It is well known that majority of the inaccuracies in simulations stem from inadequate physical layer models. In our design, we replace the link layer, the MAC layer, and the physical layer of the simulator with wireless card driver, firmware, and real wireless channel respectively.

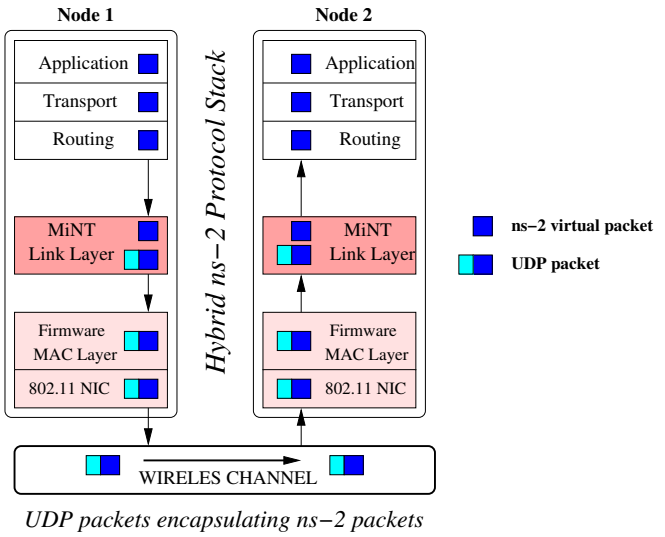

Figure 4.9: *The diagram shows the passage of a packet from one simulated node to another in hybrid simulation on MiNT. All event packets for other nodes generated in* ns-2 *are encapsulated in a UDP packet payload and given to the wireless card for actual transmission. The receiving node decapsulates the packet and inserts the event into the local event queue.*

The benefit of the hybrid simulation approach is that it requires minimal change to the already existing simulation code and scripts. The same simulation experiment can be used to obtain results in a realistic setting. The oft questioned effects of the physical layer models in simulation are corrected through use of real wireless channel.

We implement a hybrid simulator version of network simulator *ns-2*. The software architecture of *ns-2* follows the design as shown in Figure 4.9 for Node 1 and Node 2, with each layer modeled in software. In case of standard simulation, the simulator is usually executed on a single node (unless it is a parallel simulator), and each node in the simulation has a virtual node id. In hybrid *ns-2*, each physical

node runs one instance of the simulator, and a virtual node is mapped to one physical node. The hybrid *ns-2* protocol stack maintains the same layering, but replaces the link layer with an implementation of hybrid simulation link layer (LLEmu). The LLEmu layer receives the *ns-2* event packet from the higher network layer, encapsulates it into a real packet and transmits it to the appropriate physical node associated with the simulation node id. Thus, the lower layers for the real simulation are now replaced with physical entities, instead of modeled entities. This requires several changes to the *ns-2* simulator. We present these implementational changes in the following subsection.

4.3.1.1 Implementation of Hybrid Simulator

We present the challenges involved in implementing hybrid simulation capability into a standard discrete-event simulator, and detail the techniques we use to overcome these challenges for the ns-2 simulator.

Two key design components in a simulator are: (a) the way to model execution logic of different entities based on either events, activities or processes, and (b) the way the simulation time is advanced. *ns-2* is a discrete event simulator, where the execution logic is based on events, and the time is advanced at the pace of event execution time using a global virtual clock. In hybrid simulation, all packet communication is carried over real wireless medium. This leads to inconsistency between the virtual clock that determines the dispatch rate of simulation events, and the real-world clock that determines the transmission rate of packets over actual wireless channel. In order to overcome the timing problem associated with the use of a virtual clock, we use system clock on all the nodes, that are synchronized at the beginning of each experiment, to update the simulator's virtual clock. Events are now dispatched according to their real execution time instead of being executed as

soon as the previous event has finished execution. We use *ns-2*'s RealTime Scheduler with the following modification. RealTime Scheduler in *ns-2* yields execution control to the kernel while waiting for the next event timer to expire. Most operating systems however only implement a coarse process-level scheduling granularity ($10\ ms$). Due to this limitation, the control comes back to the *ns-2* scheduler only after $10\ ms$. In order to schedule events at a finer granularity, we use busy wait, and can process each event as soon as its timer expires. When a *ns-2* event packet is received from another node, the event packet is timestamped using the real system time, and inserted into the event queue.

We run an instance of the modified *ns-2* simulator on each node, and map one virtual node to a physical node. In general, *ns-2* generates all the events for each virtual node, and we must filter out the events that do not belong to the virtual node that this physical node corresponds to. If all the events are allowed to be generated, this leads to high processor usage. This could lead to some events getting generated after the real time has progressed beyond the time to execute this event. The correctness of hybrid simulation requires that events should not be scheduled in the past. For instance, if the amount of time spent in processing the simulator's execution logic is too large, then an event dispatching a packet to another node could be delayed and may be dequeued by the scheduler after the real time has advanced past its scheduled execution time. We prevent such delayed event execution by reducing the number of events that the scheduler needs to process. In our implementation, we make a simplifying assumption that only one virtual node is mapped to a physical node. However, since we execute unmodified ns-2 script on each physical node, it instantiates all the virtual nodes, including their traffic sources/sinks, on each physical node. Since we are binding only one virtual node to a physical node, therefore we prevent traffic sources on any other virtual node from generating any packet on this physical node. We identify the virtual node that is mapped to the physical node,

and only allow traffic generators, like FTP and CBR, associated to this virtual node to schedule events.

The internal packet format used in a simulator does not conform to the exact specifications of the real protocols. Hence, a packet from the simulator needs to be modified before it can be sent over the wireless medium. Current ns-2 implementation does not contain the protocol header fields needed for transmission over the wireless channel. In order to transmit an ns-2 packet sent from the routing layer onto the link layer, we implement a wrapper that encapsulates the ns-2 packets in a UDP packet payload, and delivers it to the destination node using standard socket layer. The address of the virtual node in the ns-2 packet is mapped to a core nodes IP address to which the packet is destined. Upon receiving the UDP packet carrying the ns-2 payload, the receiver node decapsulates the packet and inserts it into the local event queue. The logic for distributed execution of hybrid simulation over wireless channel is shown in Fig 4.9.

Our goal is to require minimal changes to the existing ns-2 scripts to execute them on the hybrid simulation platform. To provide a single-script abstraction, we kept the required changes independent of the individual core nodes. All changes are composed at the central distribution node, and same script is loaded on all the testbed nodes participating in an experiment. The changes to an existing script are: (i) the script must point to the MiNT link layer implementation instead of the ns-2 link layer, (ii) each testbed node is assigned a physical node-id that is used in the *ns-2* script. The physical node id for each node is preassigned and the ns-2 script reads it from an environment variable local to each node.

In our current design, only one virtual node is mapped onto a physical node. This might limit the the size of the network that can be tested in hybrid simulation by the number of physical nodes available. Careful observation reveals that it could be fundamentally impossible to share a physical node for multiple virtual

nodes given a single wireless interface. This is because if each of the virtual nodes sharing a physical node is sending enough traffic to saturate the channel, then multiplexing the wireless card would be impossible using the real clock. Also, it would be impossible to capture real MAC-level interaction, or effect of transmission over real wireless medium, for the virtual nodes that are mapped to the same physical node. For instance, assume a string topology of 3 nodes, where the first and the last node are out of each other's sense range. There are two flows, one between $N1$ and $N2$ (flow-1) and other between $N3$ and $N2$ (flow-2), active at the same time. Given two physical nodes, if $N1$ and $N2$ are mapped onto the same physical node, we fail to capture effects of real wireless medium on flow-1's packet transmissions; whereas, if $N1$ and $N3$ are mapped to the same physical node, then it would not be possible to capture the MAC layer interaction between $N1$ and $N3$.

Using multiple network interfaces, it is possible to virtualize multiple nodes on a single physical node with certain limitations. Let us assume that in the previous example, we map $N1$ and $N3$ on one physical node, and $N2$ on another. Each physical node is equipped with 2 wireless NICs. Then, flow-1 and flow-2 can use separate interfaces on non-overlapping wireless channels. Each interface has a separate IP address. To simulate multiple logical nodes, each physical node must run multiple *ns-2* instances, each of which uses separate wireless interface. The drawback of this approach is that it does not capture the interference between the two flows. In addition to that, the topology of the simulated network must be regular. The distance between $N1$-$N2$ and $N3$-$N2$ must be same, and the mobility patterns of $N1$ and $N3$ must be same. Otherwise it would not be possible to set up the correct topology.

4.3.2 Advanced Control Features for Hybrid Simulator

Most simulation tools at present is not rich in debugging features. It mostly supports a complete run of a scenario followed by offline analysis. We propose two features which will enhance the ability of a user to debug more interactively, as well as to steer experiments towards more meaningful direction based on current states of an executing experiment. First, a pause/breakpointing feature in hybrid simulation allows users to control a simulation run dynamically by *pausing* a simulation run at a user-specified breakpoint, inspecting its internal states and/or network conditions, modifying different simulation parameters, and resuming the run. Secondly, a *rollback* mechanism allows one to revert to a previous state of a long-running simulation, and resume from there with a different set of simulation parameters thus saving valuable simulation time.

4.3.2.1 Pause/Breakpointing Technique

In hybrid simulation mode, the simulator is running in a distributed manner across all nodes in the testbed. Debugging such a distributed application is a challenging task. In addition to simultaneous start and stop of an experiment on all nodes, MiNT simplifies protocol debugging by introducing other standard features of a typical debugger, namely *pause and breakpointing* of an experiment.

The implementation of the pause and breakpoint feature in MiNT is based on filtering of the event trace that is generated during the execution of the simulator. In order to enable this feature, the ns headers along with the field names and offsets are specified in a separate file. This description acts as the template for parsing each event. There are two main event matching logic that is implemented. First, a user-specified string (specified in the hybrid *ns-2* script) is searched in every event string that is generated on a node. For example, if a route error event (RERR) happens

on a specified node (say Node 1) then the simulation on node 1 could be paused, or this could lead to a breakpoint for the experiment. Secondly, it is also possible to specify a value for a particular header field. For instance, if a specified node sends a route message for a specified number of times, we may wish to pause and inspect some state. When a match is found either for a string or a specific header field value, then the subsequent action for pausing or breakpointing is triggered. For pause, a signal to pause the simulation is sent locally. If its a breakpoint, then the node daemon also notifies the control node to send a message to all other nodes in the experiment to pause the simulation.

The implementation of the *pause* feature in MiNT requires modification to the RealTime scheduler in the hybrid *ns-2*. Normally, the real-time scheduler sets the simulator's clock value to the system clock. To account for pause, the total pause period is measured and subtracted from the system clock to update the simulator's clock. When the simulation is paused, the execution of events pending in the event queue as well as those in transit to other nodes, is stalled. However, since the simulator's clock is also paused, no adjustment is needed to the time for the events in the event queue. In the pause state, the user is allowed to change the physical configuration of the testbed, or alter any physical parameters of the nodes in the testbed, like node positions or transmit power, before resuming the execution.

4.3.2.2 Rollback Mechanism

The *rollback* feature for an experiment running in hybrid simulation mode gives the flexibility to a user to repeat the experiment from a snapshot time in the past with modified parameters fed to the experiment. This saves on experimentation time as the entire simulation experiment need not be repeated from the beginning.

In order to implement this feature, the state of the executing process (hybrid *ns-2* in case of MiNT) is stored at regular intervals. On a rollback request, the saved

state is loaded and execution repeats from that point. The controller node triggers each node controller to fork the ns-2 processes running on a node and stores the process id of the forked process. On rollback, the stored process closest in time to the rollback time is selected. The remaining stored processes later in time to the executing process are purged from the stored process list. In order to maintain consistency of the display, once the node controllers report that the rollback operation has succeeded, then MOVIE also reverts all events it has processed till the rollback time by looking up the history. Once both MOVIE and each node controller have successfully completed the rollback initiation phase, the experiment is restarted from the user interface.

Note that this feature also rolls back the node positions and the *ns-2* script execution. It is, however, not possible to rollback channel conditions, which is a temporal physical phenomenon and not a software state.

4.4 Fault Injection and Analysis Tool (FIAT)

Fault Injection and Analysis Tool (FIAT) is the software component in MiNT used for installing and testing real implementations. Our aim is to make it easy for developers to use MiNT as a platform for testing and debugging implementations of different wireless protocols. To provide a development environment, it is important to include support for *debugging* implementations easily. Typically wireless applications and protocols, that are designed for multi-hop networks involving multiple nodes, are distributed in nature. This implies that testing and debugging applications on MiNT involves all the challenges of distributed system debugging. Traditional debugging techniques, such as tracing and breakpointing based on program counters and process states, is not helpful in a distributed environment. Several aspects of the system under test makes distributed debugging a hard problem. For

example, communication delay among participating nodes creates complex interaction patterns making it difficult to identify the state of the system at any given point in time; ordering of events could be different for each run making it difficult to trigger problem scenarios deterministically.

A simple technique to debug a distributed system is to trigger all possible execution paths in the protocol/application implementation, log the execution behavior, and look for anomalous behavior by analyzing the logs. For network applications, a difficult problem is to test the implementation for all possible network faults, like drop, delay, duplication or reorder of a packet. Often the testing method adopted is code instrumentation to generate a fault. Modifying implementations directly during testing is an inconvenient process, often requiring separate instrumentation effort for each scenario. We alleviate this tedious process by providing a tool in MiNT that allows developers to *conditionally trigger network faults* using a *scripting language interface*. The correctness of the implementation is also tested by matching the response of the system as a result of the fault against correct response behavior specified using the same scripting interface. Thus, FIAT is characterized by two important features: (a) Automatic triggering of realistic network faults based on user-specified network states, (b) Analysis of the system response to determine correctness of the implementation.

Figure 4.10 shows the individual software components in the design of FIAT. There are two main components, the scripting interface or the programming front-end that allows the user to specify the fault scenarios, and the fault injection and analysis engine, which introduces the faults and performs the testing. The programming tool is a declarative scripting language, called the Fault Specification Language (FSL). The fault specification in FSL is a $< condition, action >$ pair, where an action, such as a network fault event, is triggered when a condition is satisfied. The conditions are specified as a function of different packet types, and the number

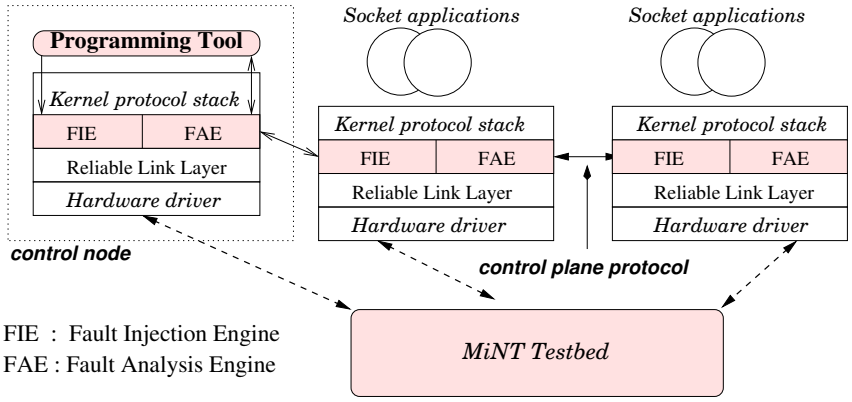

Figure 4.10: *The fault injection and analysis engine is used to test network stack implementation in kernel or user-level socket applications. Fault injection and analysis layers reside on each node. The control node hosts the programming tool that parses the user-defined fault injection and analysis scripts and initializes the FIEs and FAEs for each test case scenario through a control protocol. The control plane enables the components in the system to communicate during initialization, test case execution and error reporting.*

of packets of that type counted on a particular node. Each $< condition, action >$ pair in the specification is translated into a *table of rules* that are used for selecting packet types to monitor, as well as, set up action routines on successful evaluation of a condition. The technique used for monitoring packet types is called active probing. In active probing all packets at ingress and egress of a node is matched, and if it belongs to the set of packet types in the rule table, it is counted. On every update of packet count, the condition is evaluated and all nodes that are part of the action routine are notified to trigger the action events, which may be counter updates, assignment of variables, or introduction of faults by dropping, delaying or modifying packets. The purpose of the fault specification interface is to simplify the specification of test scenarios in FIAT. In order to do that we designed FSL with a rich set of primitives to capture all network fault events. This programming front-end is executed on a central controller node, that parses the user-defined script and installs the relevant data structures on testbed nodes. The same scripting language is also used for defining correct response of protocols to faults. The detailed syntax and semantics of FSL is presented in Appendix C.1

The second important component in FIAT is the software module that introduces the faults based on the user specified rules, and also matches the responses to infer correct program behavior. This main functionalities of this module are matching every ingress and egress packet for a match. On identifying a matching packet, this module updates counters that keeps track of observed occurrences of this packet type, transmits this updated counter values to other nodes, if required, and triggers different actions, like a network fault if conditions based on this counter is satisfied. This module sits below the kernel protocol stack, thereby allowing it to monitor all incoming and outgoing packet, and perform different packet filtering actions. There is an additional software layer, called the Reliable Link Layer (RLL) sitting below the FIE/FAE layer. The RLL layer is intended to ensure that a packet accounted

for by the sender side FIE/FAE layer is delivered reliably to the receiving node. This is important for the correctness because once a packet is accounted for by the FIE/FAE layer it is implicitly assumed that the packet has been delivered to the receiving node. In other words, the tool assumes the presence of a error-free physical layer, such that all faults introduced in the system are *completely synthetic* and *there is no unaccounted fault*. The implementation details of the FIE and FAE, as well as the RLL, are discussed in the following subsection.

4.4.1 Implementation of FIAT

The implementation of different data structures and their maintenance is at the heart of the implementation of the Fault Specification Language. The FIE/FAE, and the RLL, is implemented as a loadable kernel module for Linux 2.4 operating system. When a user is testing a prototype implementation of a protocol, FIE/FAE is loaded in the kernel. It is implemented in Linux as a pseudo network device driver. A pseudo network device is installed on the node running FIAT. All incoming and outgoing packets are directed to this pseudo device before being sent to the real network device. All packet matching operations, as well as introduction of network faults, like dropping, delaying or modification of packets are implemented as part of this pseudo driver.

4.4.1.1 Fault Specification language

The data structures used in the implementation of the FSL are a set of tables. The execution logic of the FSL interpreter revolves around creating the set of tables that can be used by the fault injection engine. The programming tool is a user level process active on the control node. The user writes a script using the specification language and submits it to the FSL parser through a command line interface. The

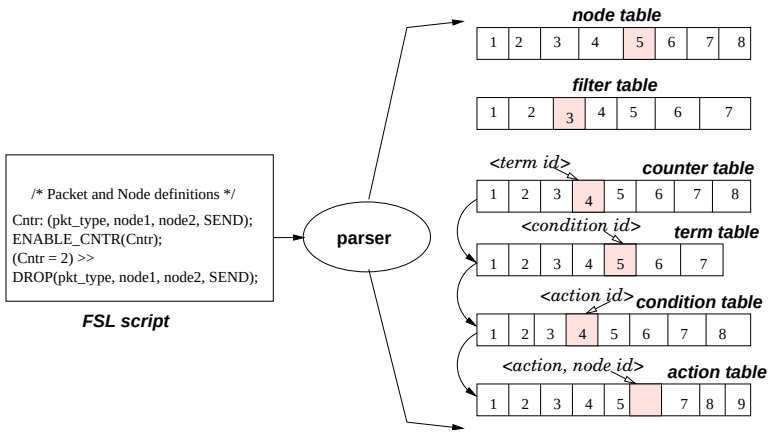

Figure 4.11: *The FSL parser generates six tables from the FSL script. The tables are sent to all the participating nodes. The packet and node definitions in the script set up the filter table and node table respectively. In the figure, a matched packet has affected counter* 4 *in the counter table. Counter 4 uses the term-id it maintains to index into the 5-th entry in the term table. Similarly, 5-th term entry indexes using the condition-id stored in it to trigger evaluation of the 4-th condition. If the 4-th condition evaluates to true, it will use the action index to trigger the 6-th action in the action table.*

interpreter parses the script to generate a set of six tables which are used to initialize each FIE and FAE involved in the test scenario. For simplicity, the complete set of tables are sent to each node and ready for use for the FIE/FAE, although only a subset of the entries in each table may actually be touched at a node.

The *filter table* and the *node table* are used for classification of each packet. Hence these are static tables, unless there is a variable defined in the filter table which is defined at run time. The rest of the tables are used to maintain Virtualwire's execution states across all the testbed nodes. There are four such tables, viz. counter table, term table, condition table, and action table. A $counter\ table$ contains the list of counters used in the scenario script. For each counter entry, the parser generates

pairs of $\{term\_id,\ condition\_id\}$ that are dependent on the counter's value, as well as, the nodes which need to be reached. A counter may appear in multiple terms and a term may appear in multiple conditions. Whenever a counter value changes we need to update the term as well as, reevaluate the conditions. Hence, it helps to tag which term and condition will get affected by a particular counter. A $term\ table$ is indexed by $term_id$s, with each entry storing term expression as a tuple comprising $counter_id$s or integer constant and relational operator connecting them. A term expression is evaluated and stored when the corresponding counter value changes. A $condition\ table$ is indexed by $condition\_id$. It stores the condition expression in terms of $term_id$s and logical operators connecting them. It also maintains a list of $\{node\_id,\ action\_id\}$ pairs so that whenever a condition is satisfied the action can be triggered. An $action\ table$ is indexed by $action_id$s with each entry storing the action to be performed and the corresponding node identifier. The interactions among these tables are shown in Figure 4.11. In FSL, one can specify a counter on a packet type on one node that can trigger the computation of a term maintained on a remote node. Similarly, a condition that is found to be satisfied on one node can trigger an action on another node.

4.4.1.2 Fault Injection and Analysis Engine

The control logic for detecting a packet of a particular type, and triggering action based on a rule is shown in Figure 4.12. Once an incoming or outgoing packet is matched to be present in the Filter Table, then the counter maintaining the observed count of that counter type is updated on that node. An *update_counter* routine sets the value for the counter defined for that packet type. An update of a counter triggers evaluation of the terms present on that node. A change in term state leads to evaluation of the condition. The condition may be local to that node, or may be composed of terms being evaluated on different nodes. In the latter case, the

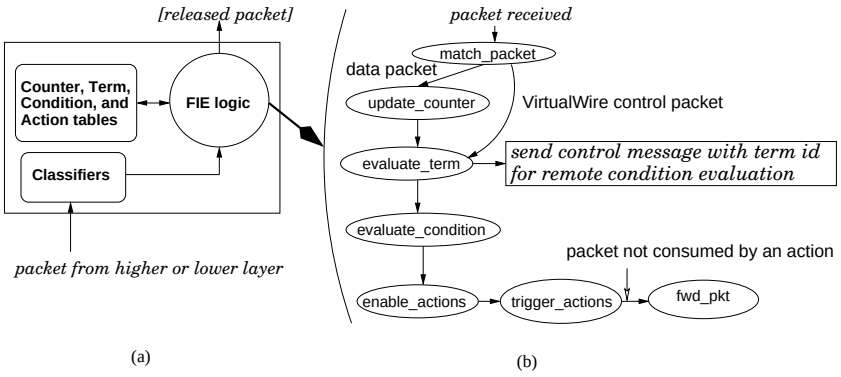

Figure 4.12: *(a) The software architecture of the Fault Injection/Analysis Engine. The "classifiers" denote the Filter Table and the Node Table used for trapping the packets to monitor. The rest of the tables maintains state information for the FIE/FAE. (b) The FIE control flow for every packet that matches a packet definition in the Filter Table. A matching packet denotes an event that will affect at least the counter table. New counter value can turn a term true, which will lead to a condition evaluation. If condition is satisfied it triggers an action. A fault type action, like drop will consume the packet, but a counter manipulation action will release the packet.*

changed term status is reported to the nodes where the condition is evaluated. We choose to evaluate the condition at the nodes, where an action dependent on that condition, might have to be triggered. This is followed by the triggering of actions, which can be a fault type or a counter-update operation.

Several challenges are addressed to implement a Fault Injection and Analysis Tool for a distributed multi-hop wireless environment. The FIE/FAE is based on the concept of probing the network traffic through a node. Due to its implementation as an additional layer in the protocol stack, it introduces extra processing overhead on each packet, thereby increasing the protocol processing latency. An ideal fault injection and analysis tool should be completely non-intrusive. Since this is not possible, therefore we aim at minimizing the processing latency of the FIE/FAE by only matching those packet types whose counters must be maintained at a node. Packets of other types bypasses this layer after matching a few fields in the header.

The underlying mechanism of FIE/FAE is based on counting packets, and using these packet counter states to evaluate conditions. Event triggering can be chained across multiple nodes, that is, events triggered on different nodes can lead to the triggering of events in other nodes. Hence it is necessary to transmit these messages in a timely manner to the respective nodes. However, in a wireless environment it might take multiple retries before messages can be successfully delivered. This could lead to failure in isolating some of the anomalous conditions in a protocol. Hence, successful execution of a scenario in this tool is *not a sufficient condition* to ensure implementation correctness. However, *violation of a condition reported by the tool definitely indicates a bug in the implementation*. Thus it is necessary to have an efficient control protocol for exchanging the states of the system across different nodes.

It could be possible that a control message must be delivered to all the nodes.

In other words control messages may sometimes need to be broadcast. In a multi-hop network where all the nodes are not in the same collision domain, the protocol design must take care of forwarding broadcast packets by intermediate nodes. The FIE/FAE layer of each forwarding node checks if the packet has a broadcast destination address, then it is forwarded. This is similar to a flooding mechanism used in many ad hoc network protocols for delivering control packets across all the nodes. In order to reduce the impact of the control traffic on the data channel, we use an alternate wireless NIC for exchanging control messages, thereby creating no interference on the data traffic.

4.4.1.3 Reliable Link Layer

The unreliable wireless links introduce a problem with accounting of packets by the FIE/FAE running on a sender node. Once FIE/FAE on the sender has accounted a packet as transmitted, it implicitly assumes that the packet is successfully delivered to the receiver. This implies that if the packet is lost in transit due to corruption, then the accounting in FIE/FAE layer will be incorrect. Therefore, we must ensure that a packet accounted by a sender-side FIE/FAE is reliably delivered. A way to prevent the system from slipping into an incorrect state is to create a "controlled" environment in spite of the losses due to the error-prone wireless channel. We design a *Reliable Link Layer (RLL)* to prevent channel errors from causing packet drop when the FIE/FAE is unaware of such packet loss. The RLL guarantees reliable delivery of packets passed to it by the FIE/FAE layer. If multiple retries from the Reliable Link Layer of the sender fails, the test run is aborted and must be repeated.

The RLL is implemented as another layer right below the fault injection layer, as shown in Figure 4.10. All packets that are identified to be packet types to be monitored are passed on to RLL. At this layer, another header is appended to the packet and is sent out on air. The receiving RLL on successful reception of the

packet responds with an acknowledgment packet, and sends the packet up to the fault injection layer on that node. If the sender does not receive an acknowledgment for a timeout period, it retransmits the packet. For this purpose, the sender currently maintains the packet till it has successfully transmitted the packet. If the packet transmission does not succeed in spite of multiple retries, then the packet is finally discarded from the sender RLL, and an error notification is sent to the FIE layer indicating the experiment scenario to be aborted.

4.4.1.4 Extending FIAT for Hybrid Simulation

The fault injection and analysis tool is currently implemented to test only real implementations. However, it is worth noting that the breakpointing mechanism in hybrid simulation uses a very similar technique of packet matching and filtering based on header fields, as used in FIAT. In this subsection we present a design for utilizing the already implemented packet matching and filtering mechanism of *ns-2* based hybrid simulation for incorporating the fault injection and analysis mechanism of FIAT.

Similar to the implementation of FIAT for real implementations, for hybrid simulation also the fault injection and the reliable link layer are inserted at the lowest point in the network stack, as shown in Figure 4.13. The *ns-2* event packets are all directed to the MiNT link layer implementation, which is the layer that encapsulates a *ns-2* virtual packet into a real UDP packet and sends it over a socket connection. When the FIAT mode is enabled, just like enabling the breakpointing mode, then all packets are redirected to the fault injection and analysis layer instead of being sent out directly over the socket. Each packet is matched for header fields that the user has specified using the same scripts that are used for specifying the breakpoint rules. Once the events are matched, the control plane implementation follows the same technique as in FIAT implementation. The only difference in this case

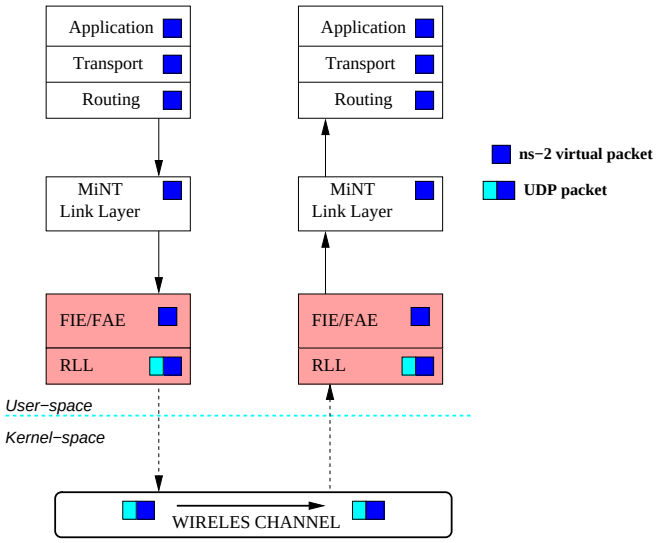

Figure 4.13: *Extension of FIAT for hybrid simulation requires all* ns-2 *event packets to be forwarded to the Fault Injection and Analysis layer, just as it is done for breakpointing implementation in hybrid simulation. The packet is matched based on the user defined packet filters, and if no match is found it is sent out as a UDP packet for the designated node(s). On a match, the fault injection rules are triggered. The reliable link layer necessary for FIAT is also implemented in user-space.*

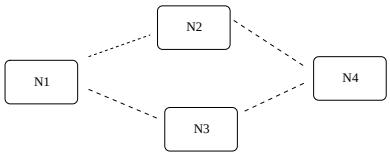

Figure 4.14: *The topology to keep in mind while understanding the fault specification script in Algo 2. Node 1 generates a route request for Node 4 in this scenario. The RREPs are lost due to bad links and fail to reach the routing layer of Node 1.*

is that the control protocol is implemented at the user level instead of residing in the kernel. The complete implementation of FIAT for hybrid simulation will be a user-space implementation.

4.4.2 Example Application of FIAT

In this section, a simple example is used to introduce the use of the fault injection technique to detect (in)correctness of a protocol implementation. Given an implementation of AODV, we show how to verify the implementation of one of its basic mechanism of route discovery. According to the RFC for Ad-hoc On-Demand Distance Vector (AODV) routing [AOD] a route request is generated by a node whenever there is a packet ready to be sent to a destination, and the originating node does not already have a route to the destination. For example, in Figure 4.14, N 1 has a packet to send to N 4. Therefore, N 1 creates a Route request (RREQ) packet and broadcasts it. On receiving the first RREQ, the destination node (N 4) responds by sending a unicast Route reply (RREP) packet. However, if the RREP packet is lost, then the originating node (N 1) must wait for a fixed period of time, and then rebroadcast the RREQ packet. According to the RFC, the number of times the RREQ should be retried before sending a destination unreachable message to the application is determined by a tunable parameter, $RREQ\_RETRIES$. This

Algorithm 2 Fault Specification Script: The script tests the implementation correctness of response on AODV route request (RREQ) failures.

```
1:  SCENARIO TCP_RREQ_Fail 200sec
2:
3:   /* count RREQ packets */
4:   RREQ : (aodvRREQ, node1, ALL, SEND)
5:   /* count RREP packets */
6:   RREP : (aodvRREP, node4, node1, RECV)
7:
8:   (TRUE) >> ENABLE_CNTR( RREQ );
9:      ENABLE_CNTR( RREP );
10:  /* fault injection */
11:  ((RREP > 0)) >> DROP aodvRREP, node4, node1, RECV;
12:
13:  /*** Analysis Script ***/
14:  ((RREQ > 2)) >> FLAG_ERR ; /* incorrect behavior */
15: END
```

is by default set to 2, but can be modified. Assuming that in a particular implementation, the value of $RREQ\_RETRIES$ must be set to 2. The purpose of the script is to verify that indeed this is the case without going into source code of the implementation.

The script maintains a count of the number of RREQs sent out from node 1 and the number of RREPs received from node 4. The fault is injected by dropping any RREP that reaches node 1 from node 4. The RREP packet is dropped at node 1 at the fault injection layer, thus creating a scenario where the RREP packet is lost. After waiting for a timeout period ($NET\_TRAVERSAL\_INTERVAL$)

the originating node, node 1, resends the RREQ packet. The RREP sent back from node 4 is again dropped as before. Since, the maximum number of RREQ to be tried for a single packet is supposed to be set to 2, therefore, the final line in the script checks that if the RREQ packet count goes above 2, then an error is flagged.

4.5 The Users' View of MiNT

Using a testbed for experimentation requires understanding the steps involved starting from configuration of experiments to data collection and analysis. In any large scale testbed, it is essential to understand the complete workflow for running an experiment. This problem of effective workflow management of experiments for remote users has been observed by maintainers of various large scale testbeds, like PlanetLab and Emulab. PlanetLab proposes Plush [ATSV06] for managing applications running over a large-scale distributed system. Similarly, Emulab is also trying to come up with an integrated workflow management for remote users [ESS+06]. This section presents, with the aid of screenshots for different stages, the steps involved in the life-cycle of an experiment in MiNT. Different tools discussed in the earlier sections come together to present the complete experiment environment.

The overall set up of MiNT with respect to a user, as well as the MiNT administrator is shown in the schematic of Figure 4.15. The key players in the set up are the administrator and the remote users. The resource they want to control are the MiNT nodes. For an administrator, she has access to the MiNT gateway node, as well as the tracking server. For a remote user, the access is limited to the gateway node. From the gateway node, the MOVIE interface is used to gain all access to the MiNT nodes. Thus any restriction that is required for user access of the testbed can be setup at the gateway node. The details of the workflow for the administrator and a remote user is presented in the following subsections.

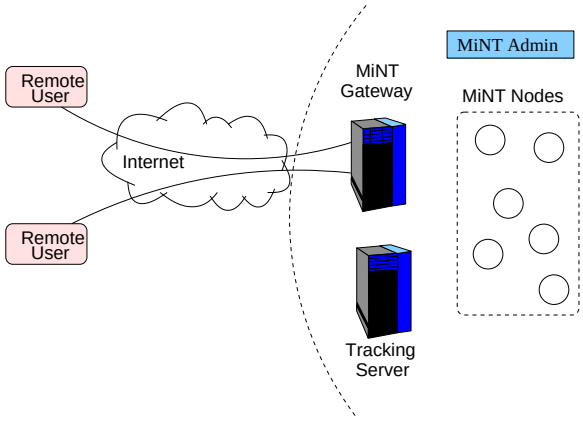

Figure 4.15: *The schematic shows the setup for accessing MiNT from a remote machine.*

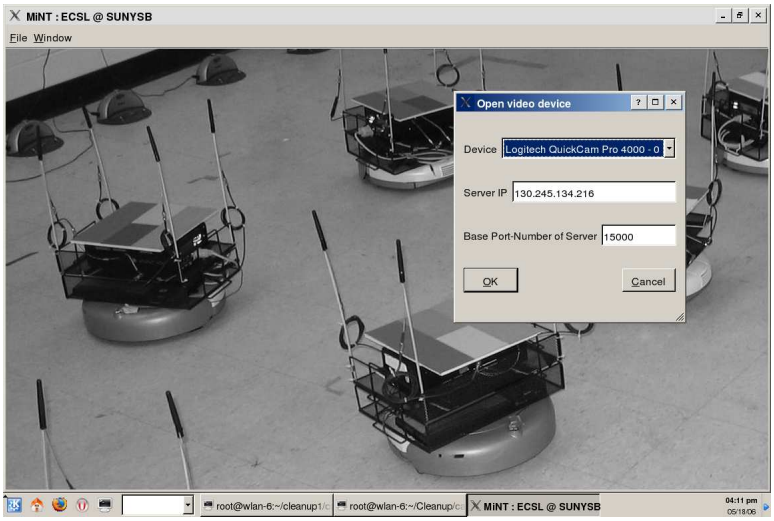

Figure 4.16: *The screenshot shows the initial set up screen for the tracking subsystem.*

4.5.1 MiNT Administrator's Role

The administrator's role is to bootstrap the entire system before a remote user can login and experiment on MiNT. The administrator must login to the gateway node, that functions as the control node for the MiNT nodes, and boot it up. This starts

Figure 4.17: *The screenshot shows the views from 6 different webcams.*

the DHCP server that waits for incoming address requests from the MiNT nodes. Next, MiNT nodes are booted, and each of them acquire a DHCP address, connects to the control node, and waits for any commands to come in from the control node. Finally, the tracking system is booted. Booting tracking system involves providing the control server IP address, and the port over which to transfer the tracking data to the control server. This is shown in Figure 4.16. Once the tracking system starts collecting the images, the administrator can check if all the cameras are functioning correctly by checking the feeds from all the webcams, as shown in Figure 4.17. Another necessary step for the administrator is to create a user account for any remote user on the gateway node.

4.5.2 Remote User's Workflow

A user of the MiNT system can reside anywhere as long as she can connect to the gateway node over the Internet. Another requirement on the users' system is the availability of a software that can export the display from a Linux machine. For

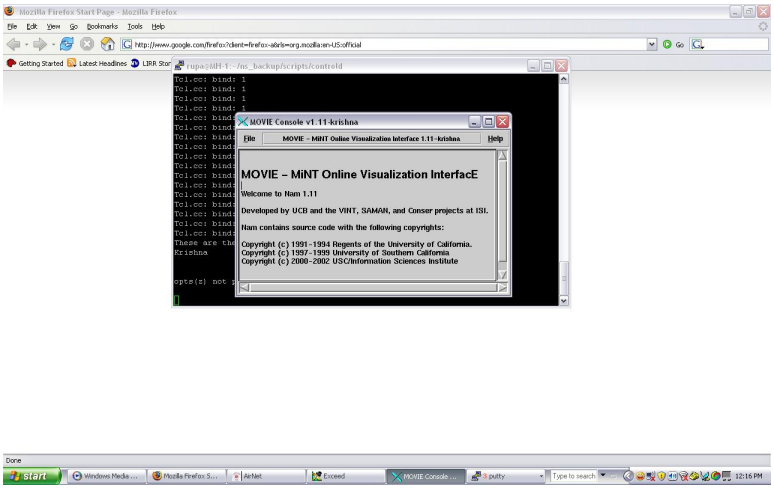

Figure 4.18: *The MiNT login screen. This is the first screen of MOVIE that the user sees after logging in to the gateway node, and launching the controller.*

example, a user working on a windows system may use the XWin32 software for accessing the MiNT gateway node. Assuming an account has already been created for the user on the gateway node, she can now login to the gateway using XWin32, and open a shell. Then the user launches the graphical user interface for MiNT, that is called MOVIE. This is shown in the screenshot of Figure 4.18.

The next step a user looks for is to create her own experiment and launch it on MiNT. First, let us assume that a user would like to create a simulation experiment, either hybrid simulation or a pure *ns-2* simualtion. Besides generating the scripts, as described earlier in Section 4.3, the user must also configure the physical entities in the testbed that cannot be configured directly through a script, as is usually done in pure simulation. In order to set up an experiment on MiNT, a user may need to configure the following, the testbed topology, the applications to execute on the testbed nodes, the mobility pattern of the nodes, and the node parameters. First, the testbed topology must be configured once the hybrid simulation script is loaded following the steps as illustrated in the screenshots of Figures 4.19 - 4.22.

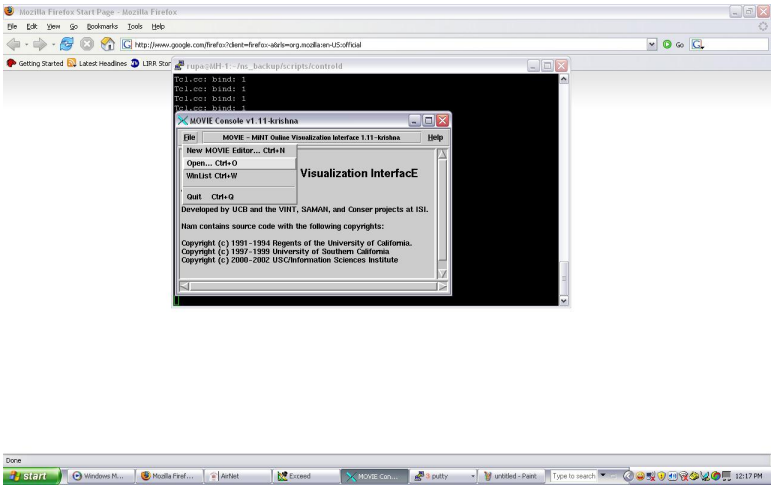

Figure 4.19: *Step 1 for loading a hybrid simulation script for execution.*

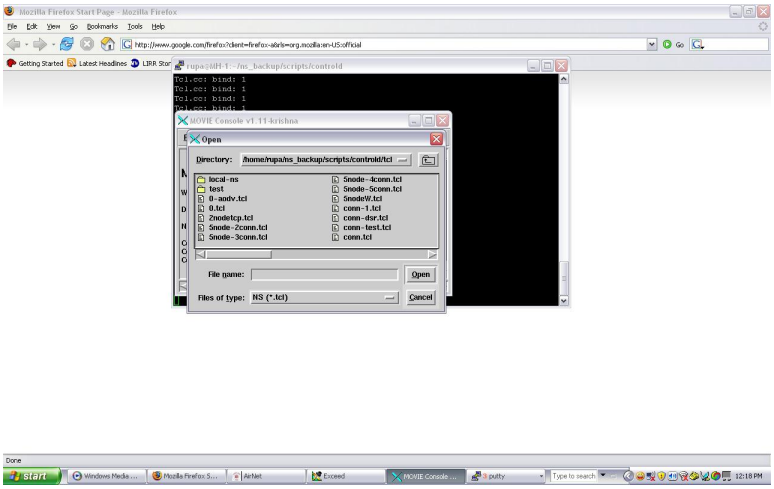

Figure 4.20: *Step 2 for loading a hybrid simulation script for execution.*

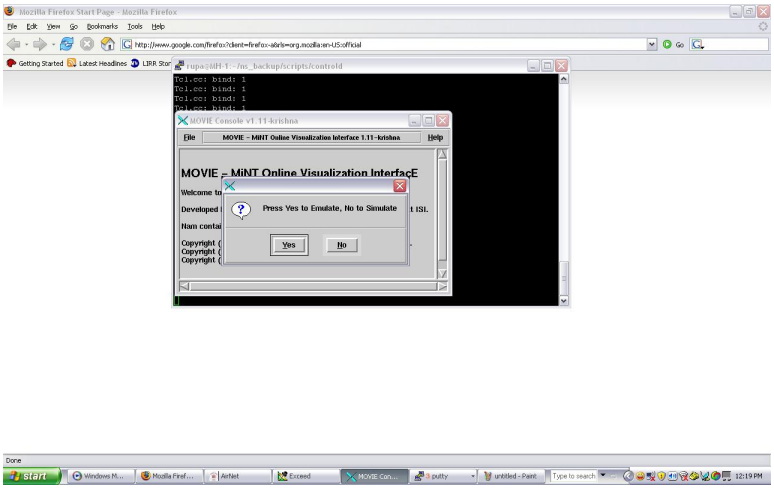

Figure 4.21: *Step 3 for loading a hybrid simulation script for execution.*

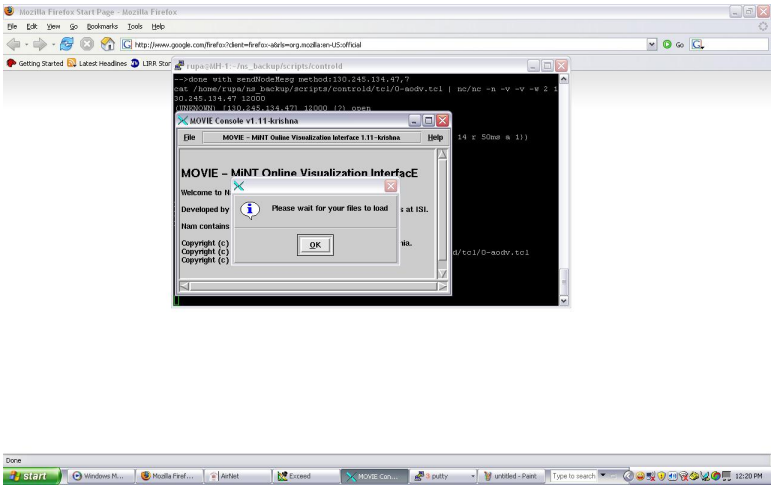

Figure 4.22: *Step 4 for loading a hybrid simulation script for execution.*

MOVIE allows users to move the node icons that shows up on the canvas to create a topology. The movement of the node icon in MOVIE leads to physical movement of a node in the testbed. Following the node movement, the link characteristics get altered, and this is displayed back to the user, as shown in Figure 4.23. Based on the link quality feedback, the user can further fine-tune the position of the testbed node if necessary. In order to configure different parameters on a node, like the wireless network card transmit power, the user has the option of configuring it globally for all nodes, or it can be done individually for each node. Global parameter setting is done through the *Config Params* button on the menu tab in MOVIE. For setting parameters individually, right-clicking on the node icon pulls up a list of tabs, as shown in Figure 4.26, that can be used for configuring each node. For describing node mobility pattern, the user specifies the intermediate positions and final destinations, along with their relative temporal offsets with respect to the beginning of the simulation run. From these information, instead of statically computing a global trajectory for each moving testbed node, MiNT relies on a run-time collision avoidance algorithm that dynamically resolves possible collisions among testbed nodes by halting some of them when collisions become imminent.

Once all the setup is complete for a hybrid simulation experiment, the MOVIE shows a button *Play*, that is used to launch the simulation experiment simultaneously on all the nodes. It is possible to control the experiment from this point onwards either by pausing it, or by setting filters to stop the experiment on anomalous events. During the experiment, values of several parameters are dynamically updated and displayed in MOVIE. At the end of the experiment, the data collected on each node can be viewed through MOVIE by fetching the file from a node. Again, in this case, it is possible to look at individual files from each node, or just collect all the files using a single click of a button. Figure 4.27 shows the result of an experiment that is collected from a specific node.

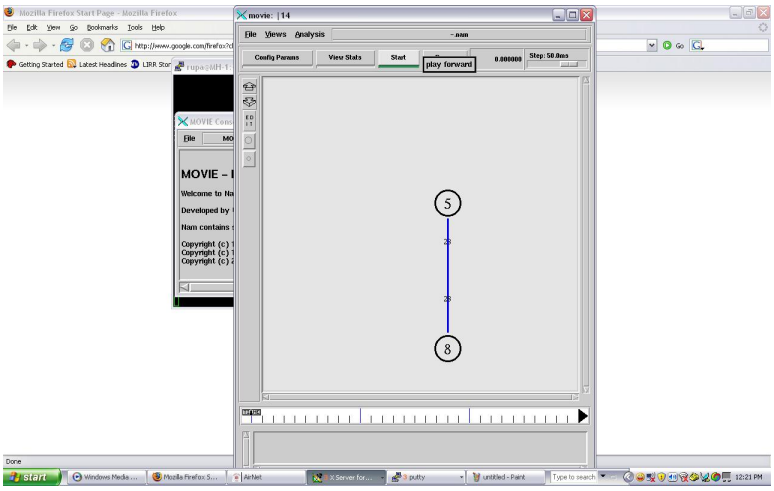

Figure 4.23: *Executing a hybrid simulation script already uploaded on the chosen MiNT nodes through a single click in MOVIE.*

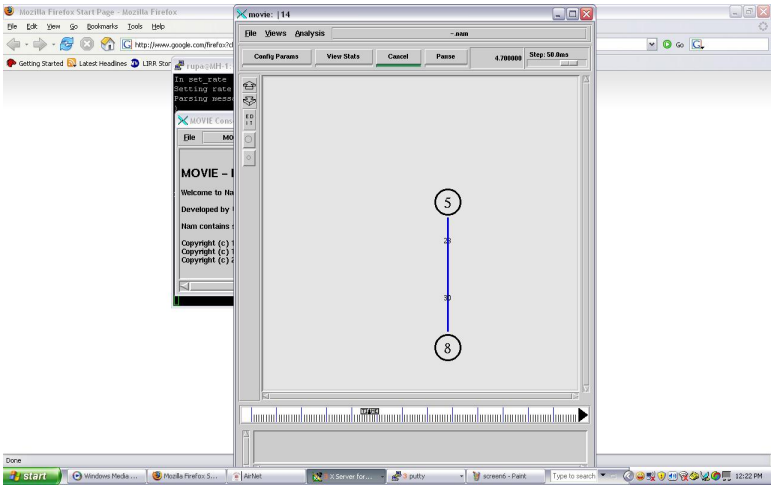

Figure 4.24: *A hybrid simulation experiment in progress.*

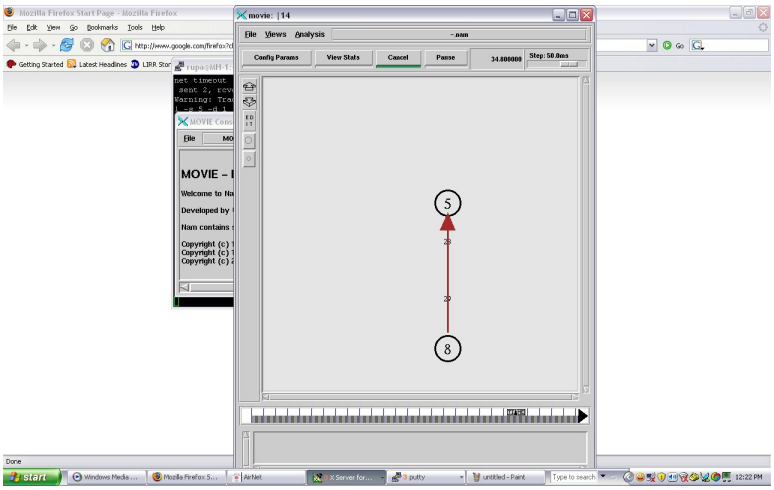

Figure 4.25: *A hybrid simulation experiment showing changes in routes and signal strengths.*

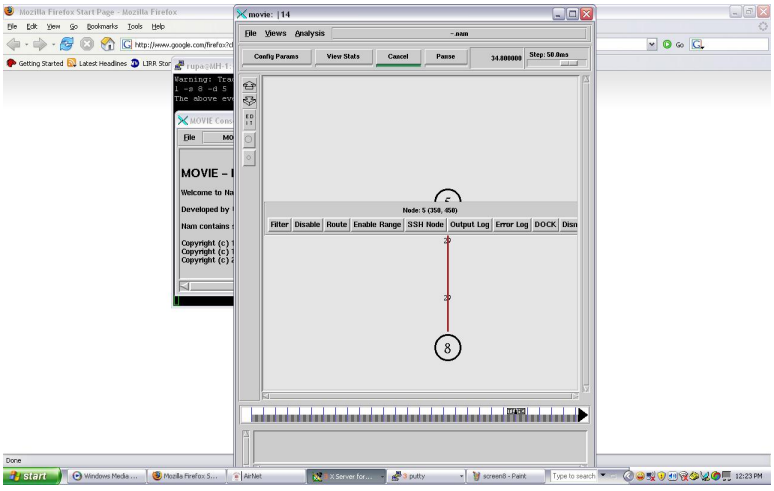

Figure 4.26: *The screenshot shows the ability to configure data collection on a per-node basis. The same interface is also used by a user to configure node level parameters on a node-by-node basis, instead of setting them globally.*

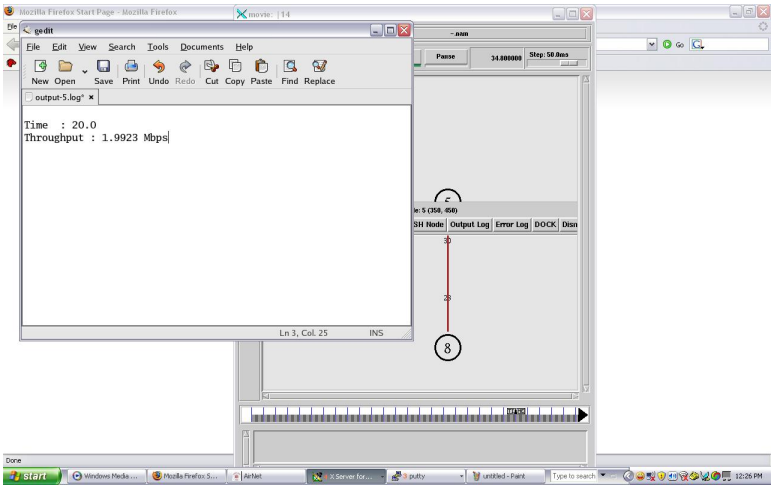

Figure 4.27: *Inspecting the output of a hybrid simulation experiment through MOVIE.*

Chapter 5

Evaluation of MiNT

This chapter deals with the analysis of different aspects of the testbed related to the design and software tools as discussed in the earlier chapters. We start by evaluating the fidelity of experiments on MiNT. Our main focus is to determine whether the technique of miniaturization distorts the results that are derived through experimentation on MiNT, as compared to a similar experiment without applying miniaturization. Next, we analyze several algorithms that are used in the software tools applied in MiNT. We mainly study the algorithms used for collision avoidance and automatic re-charging. The benefits of hybrid simulation and insights revealed through hybrid simulation are also presented. Experimental results on the the Fault Injection and Analysis engine are presented as part of the evaluation of MiNT. Finally, we present a case study using a wireless protocol that we have analyzed on MiNT.

5.1 Fidelity of MiNT

We have introduced miniaturization as a new technique for conducting experiments on wireless networks easily. However, it remains to be verified if miniaturization

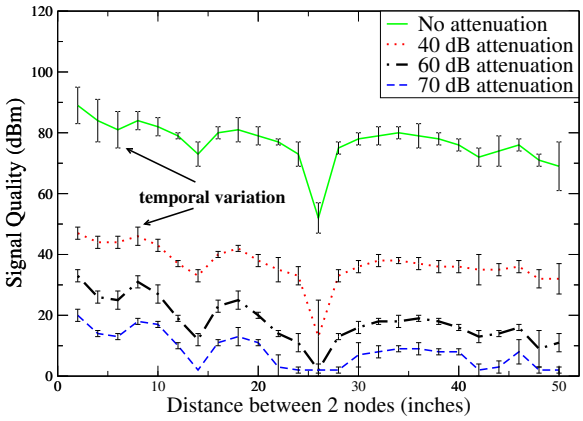

Figure 5.1: Graph showing variation of signal quality at different overall attenuation on a link. It also shows the extent of temporal variation of the signal quality at each sample point. The signal quality varies non-monotonically over distance because of multi-path fading. The variation of signal quality for the attenuated and the non-attenuated case follows the same pattern.

affects the conclusions that are drawn from the experiments on MiNT relative to a typical wireless testbed. In this section, we show that miniaturization does not alter the key characteristics that are important for the wireless experiments. However, it is worthwhile to mention here that we are not aiming to mimic a typical wireless testbed in every respect, rather we are providing a platform that should be viewed as another set-up for such experiments without losing the generality of the conclusions drawn from the experiments. If the goal is to design an exact replica of the non-miniaturized testbed, a scaled down version of the testbed can be built, as shown in the works of Kansei testbed [EARN06]. Our approach in showing the fidelity of MiNT is by comparing results of experiments focused on each layer in the network stack, where one run is on MiNT and the other run is on an non-miniaturized version (no attenuators in the signal path) that closely resembles the MiNT set-up in terms of channel conditions.

Signal propagation is a key aspect of the wireless physical layer. We study the impact of attenuation on signal propagation characteristics in MiNT. In this experiment, we use 2 nodes connected in ad hoc mode and apply different levels of attenuation. We compare the resulting spatial distribution of signal quality (SNR) with that of the non-attenuated case. Figure 5.1 shows the variation of signal quality reported by the card firmware, when signal attenuation on the path is varied from 40 dBm to 70 dBm. The signal quality is measured at 2 inches granularity. The same graph also shows the extent of time variation of signal quality at each sample point.

The figure shows that the signal quality variation is non-monotonic. There are intermediate regions where the signal is weaker relative to the neighboring regions, or even fades completely. These regions of weak signal quality, termed *dark spots*, are primarily a result of multi-path fading. When the attenuation is removed completely, the signal quality improves, but the nature of its variation is preserved. The IEEE 802.11-1999 standards [IEE] also show similar non-monotonic distribution of signal quality. Furthermore, signal quality at any point for the attenuated and the non-attenuated cases show similar temporal variations.

Figure 5.1 also indicates how to configure a topology in MiNT. For example, when 70 dB of attenuation is applied, within a radius of 4ft (48 in) there are regions of good connectivity (16 dBm) and complete disconnectivity (2 dBm). Reducing signal attenuation and keeping the space unchanged makes the entire space better connected. By adjusting attenuation level to a specific research task's needs, one can trade off the minimum signal quality with the physical space requirement of the set-up. As the maximum communication range of a node at 70 dB attenuation is 4 ft, it should be possible to set up a multi-hop 16-node mesh network in a 12 ft x 12 ft space.

In this experiment, we study the impact of attenuation on fairness property of

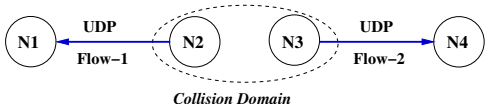

Figure 5.2: String topology where the two sender nodes are in the same collision domain and contend for access to the shared wireless channel.

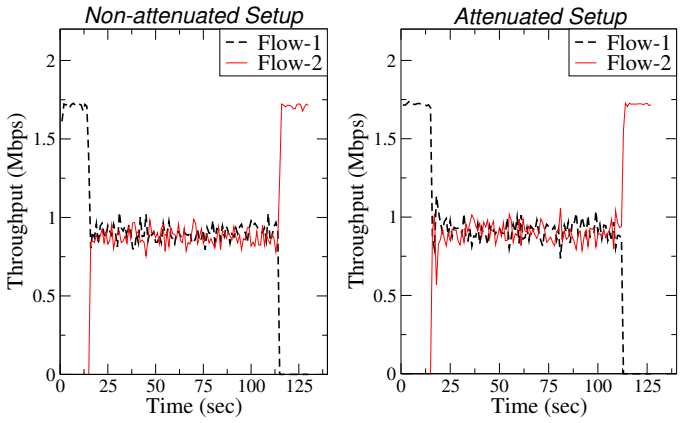

Figure 5.3: This graph shows bandwidth sharing between two unicast flows when the senders are in the same collision domain, as shown in Fig 5.2, for an attenuated and non-attenuated set-up. The link quality between the two contending nodes is kept same across both set-ups. The channel is shared equally in both cases proving that the MAC layer is unaffected by introduction of attenuation.

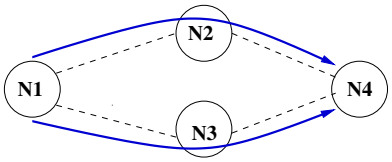

Figure 5.4: The 2-hop topology used to run the AODV protocol experiment. The same topology was replicated with and without attenuation on MiNT keeping the link quality same.

channel access algorithm. We set up a string topology of 4 nodes, as shown in Fig 5.2. Node $N2$ is sending unicast traffic to node $N1$, and node $N3$ to node $N4$. Since $N2$ and $N3$ are in the interference range of each other, they contend for access to the shared wireless medium. We compared two different set-ups – one with attenuators and the other without attenuators – while keeping the link quality same across both set-ups.

Fig 5.3 shows the instantaneous throughput of the two UDP flows for both the cases. As soon as the second flow starts, the channel is shared equally between the two contending flows. The bandwidth sharing behavior is same in the attenuated and the non-attenuated case.

In this experiment, we show that the behavior of the routing layer protocols is not affected by introducing attenuators on the signal path. We use a 4-node network topology, where the end nodes are connected over 2 hops, as shown in Fig 5.4. In this experiment, we use AODV-UU [Ad-] protocol to route packets between $N1$ and $N4$. The link quality is maintained same across the attenuated and the non-attenuated runs.

In each experiment, the route between node $N1$ and node $N4$ (chosen by AODV-UU) is made to fail by artificially failing the intermediate hop. Figure 5.5 depicts the time taken for new route discovery when such a failure occurs. The time taken in both attenuated and non-attenuated cases varies between 7 ms to 12 ms,

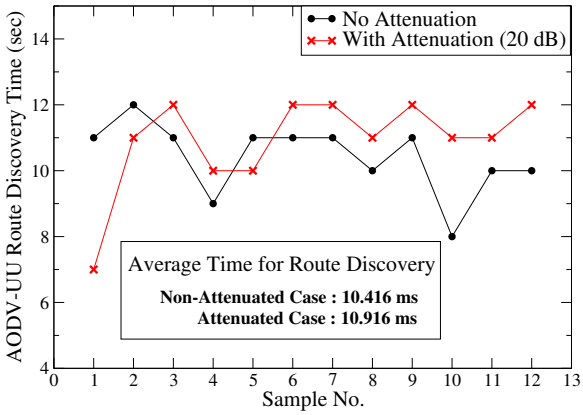

Figure 5.5: This graph shows the comparison of AODV-UU [Ad-] Route Discovery time in attenuated and non-attenuated set-up using the topology shown in Figure 5.4. The route discovery time varies between 7 ms to 12 ms, and the average time for attenuated and non-attenuated cases are 10.916 ms and 10.416 ms respectively.

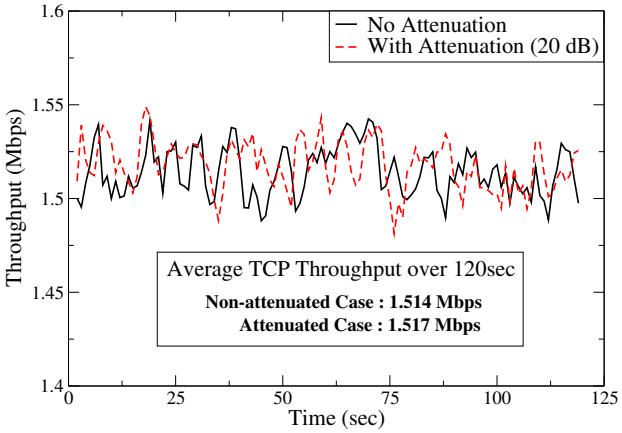

Figure 5.6: This graph shows the throughput of a 1-hop TCP flow. The first set-up does not use any attenuator, while the second one uses a 20 dB attenuator. The link quality is kept same in both the experiments.

and the average over 12 samples is 10.416 ms and 10.916 ms for the non-attenuated and the attenuated case respectively.

To prove that the transport layer is unaffected by attenuators, we use a 1-hop TCP experiment. We use 2 nodes connected in ad hoc mode and measure the throughput of a TCP connection between them. The link quality is again maintained same across the attenuated and the non-attenuated set-up.

Figure 5.6 shows the TCP throughput over 120 sec, averaged over 3 sec periods. The long-term average for the TCP flows are 1.514 Mbps and 1.517 Mbps for the non-attenuated and the attenuated case respectively. Even the instantaneous variations are similar in nature, suggesting that the transport layer behavior is not affected by use of attenuation.

5.2 Evaluating MiNT features

MiNT has several interesting features like the collision avoidance during mobility of the nodes, automatic re-charging of the nodes. In this section, we evaluate and study scalability of these features. There are several parameters that can be tuned to get optimal performance for the different features that have been discussed earlier. Wherever possible we will also present the effect of changing the parameters.

5.2.1 Tracking Accuracy and Scalability

The tracking system is required to get accurate position/orientation information of each node in the testbed. Ideally, if each step of the Roomba movement is exact, then it is possible to figure out the current location of the node, based on the initial position and the steps executed. However, due to floor friction and mechanical non-homogeneity of the Roombas, the Roomba movement is not exact. Figure 5.7 shows the inconsistency in Roomba movements for both the *move forward* as well as the *rotate* commands. This makes it necessary to design a full-scale vision based

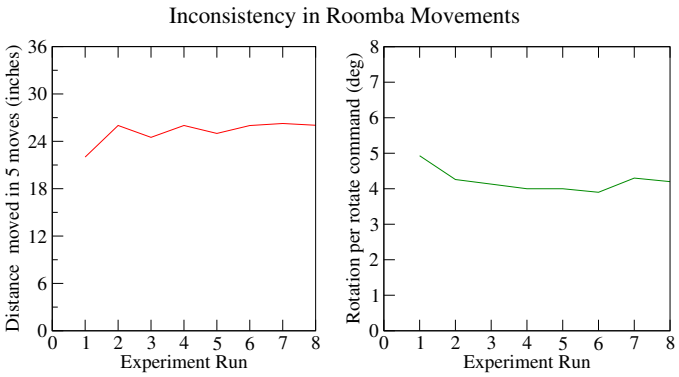

Figure 5.7: *Roomba movement with each set of move/rotate commands. The distance traveled and rotation performed are not consistent making it difficult to track the Roomba position based on steps executed. This inconsistency makes the vision-based tracking system indispensable to keep track of current node positions.*

tracking system that can provide a more accurate locationing information than that based on odometry.

We measured the *locationing accuracy* of the tracking system. The inaccuracy is measured as the difference between the location and orientation of a MiNT node as returned by the tracking system and its true coordinates. The mean error in the coordinate distance is 0.95 inches with a standard deviation of 1.17 inches. The mean error in orientation is 3.36 deg with a standard deviation of 2.77 deg.

Another important factor is the *scalability* of MiNT-m's object tracking algorithm with increasing number of nodes. This is measured as the time taken for end-to-end tracking (including frame grabbing, node identification, node locationing, and merging of location data from multiple tracking servers) as the number of nodes in the testbed increases. In Figure 5.8, we plot the time taken by the tracking server to produce one set of node locations. Although the tracking is done in parallel on all the tracking servers, multiple nodes could be clustered within the area

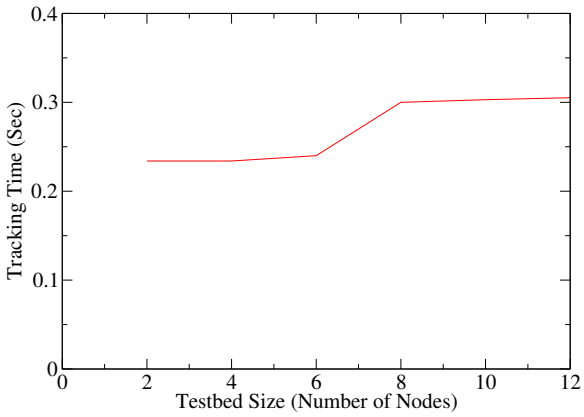

Figure 5.8: *Scalability of the tracking system in terms of nodes tracked. With increasing number of nodes the time to locate each node increases. With 12 nodes, the tracking system can produce one location update every* $0.3sec$.

covered by one tracking server. This leads to an initial increase in the tracking overhead with number of nodes. With 12 nodes, the tracking system could produce one location update for all the nodes every $300\ msec$. The maximum number of nodes that can fall within the coverage area of a tracking server is limited. Hence, as the testbed scales further and more tracking servers are added, the tracking overhead should not increase any further.

One of the physical constraints we had was the height of the ceiling. If the webcams could be placed higher up, then each webcam could cover a larger area thus scaling the testbed size. To evaluate this theory, we scaled down the size of the images we got from the webcams, and ran the tracking algorithm on them. Even when each webcam image is shrunk to 1/16th of its original size (equivalent to placing the camera at 4 times the current height), the tracking system works well. In particular, the tracking system's locationing error just increased from $0.95\ inches$ to $2.32\ inches$, while the error in reported orientation increased from $3.36\ deg$ to $4.11\ deg$.

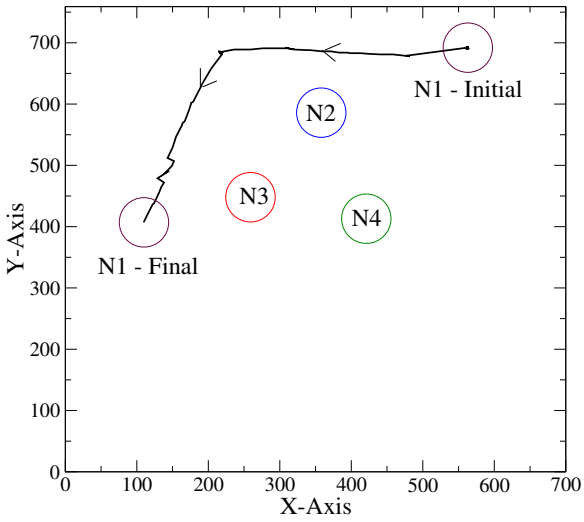

Figure 5.9: *Node path trace collected from MOVIE: Given a destination a node can detect the presence of obstacles, and find a collision free path.*

Number of Nodes	**Reconfiguration Time (sec)**
2	18
3	38
4	47
5	82
6	100

Table 5.1: *Topology re-configuration time increases with the size of the testbed.*

5.2.2 Collision Avoidance

Figure 5.9 shows the path followed by a node as a result of the trajectory determination algorithm used in MiNT. There are three other obstacle nodes on the mobile node's path. The trajectory determination algorithm takes the obstacles into account, and generates a path that avoids these obstacles. The line in the figure shows

the resulting path.

To get an estimate of the overhead introduced by the combination of position/orientation tracking and collision avoidance algorithm, we perform the following experiment. We measured the time taken by a mobile node to move from initial marked position to final marked position once with (i) tracking system and collision avoidance turned *on*, and next with (ii) tracking system and collision avoidance turned *off* and the mobile node moving through the same path selected in case-(i). The time taken for case-(i) was 31 sec, while that for case-(ii) was 26 sec, showing that tracking system and collision avoidance algorithm combined induce a 20% overhead on configuration time.

Table 5.1 presents the topology reconfiguration time. In each experiment run, all nodes started in parallel from fixed initial positions around the corner of the testbed arena. The final position of each node was chosen randomly and kept constant for all the experiments. As more nodes are introduced and they try to reach their destination in parallel, there are effectively more dynamic obstacles present in the environment leading to increase in the time to reach the final topology configuration. Most current testbeds either do not support experiment-by-experiment topology reconfiguration, or require several hours to come up with a specified topology. Comparatively, MiNT reconfiguration takes time of the order of minutes.

5.2.3 Auto Re-charging

We measured the charge and discharge times for the two batteries. This information is used by the residual charge estimation algorithm to predict when a particular node needs to be re-charged. With a fully charged battery, the RouterBoard lasts around 13.5 hours without performing any network or hard disk operations. The runtime reduction due to different activities are: 2.05 sec for every 1M network operations, and 8.82 sec for 1K disk operations. The runtime reduction due to IR operations

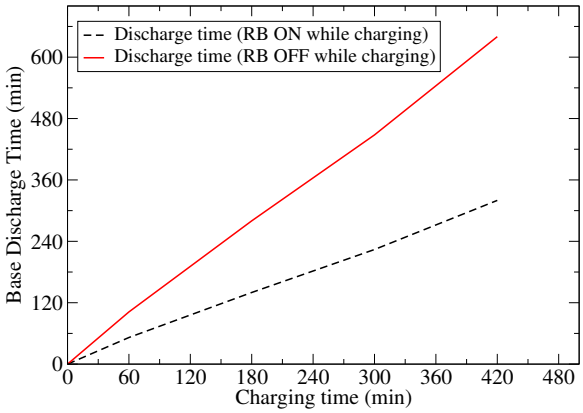

Figure 5.10: *Charge and discharge cycles of a MiNT node. If the RouterBoard is halted during charging, the battery charges faster, hence it runs longer as shown by the increase in discharge time.*

is negligible. On the other hand, the Roomba battery lasts for 2 weeks without movement, and can perform 13840 moves till the battery dies. Since the number of mobility commands executed are less than the RouterBoard activities, therefore the RouterBoard usually depletes faster. The full charging time for the Roomba battery is also 3 hours, which is less than the time to charge the RouterBoard's battery.

Figure 5.10 shows the base discharge time (no network card or hard disk activity) for the RouterBoard battery when the node has been charged for different periods of time. The linearity of the discharge time with respect to charge time simplifies the algorithm used to estimate how much charge the battery has accumulated for a certain charging duration.

If the RouterBoard is active during the charging process, the battery gets depleted while charging. This leads to a faster discharge during operation. To increase the lifetime, the RouterBoard is put into a *halt* state during charging, and powered up using Wake-on-wireless LAN feature available on the wireless network cards

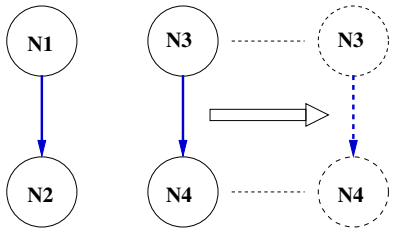

Figure 5.11: Topology used for understanding the impact of signal propagation characteristics on channel access pattern determined by the MAC layer. Node pair $N1$-$N2$ is kept fixed at one position, while node pair $N3$-$N4$ is moved away from $N1$-$N2$.

before putting it back into operation. This technique produces a substantial improvement on a testbed node's battery lifetime, as shown in Figure 5.10.

5.3 Evaluating Hybrid Simulation against Pure Simulation

In this section we present a comparative evaluation of software-only *ns-2* simulations and hybrid *ns-2* simulation executed on MiNT. The main difference between pure *ns-2* simulation and hybrid simulation is that the latter replaces the simulated link, MAC, and physical layers with real implementations and real wireless channel. We study the impact of physical layer characteristics, *viz.* signal propagation and error characteristics, on data transfer rates for both the platforms.

5.3.1 Signal Propagation

In this experiment we demonstrate the impact of signal propagation on experimental results in pure simulation and hybrid simulation. We use 2 unicast flows, between nodes $N1$-$N2$ and $N3$-$N4$, as shown in Fig 5.11. The MAC layer on the senders

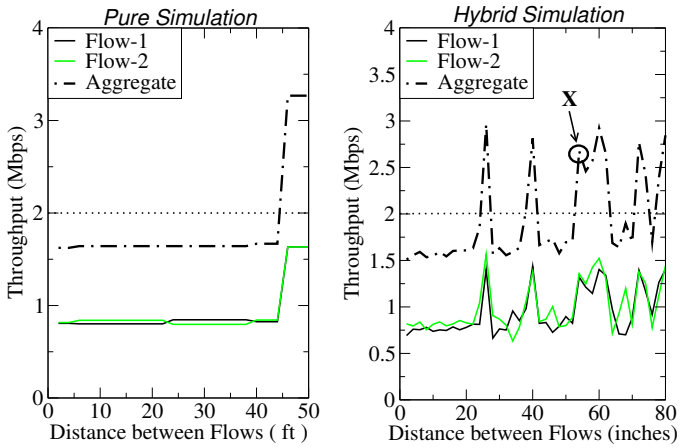

Figure 5.12: This graph shows the difference in experimental results obtained from a similar set-up in two different environments – pure simulation and hybrid simulation on MiNT. It shows the impact of signal propagation characteristics on the behavior of the MAC layer. The graph shows the throughput variations of two unicast flows, shown in Figure 5.11, as they are moved away from each other. Use of two-ray ground propagation model in pure simulation leads to the MAC layers of the senders perceive the other sender's transmission till they are out of "sense range". In hybrid simulation, the signal quality variation is non-uniform, and the senders move in and out of sense-threshold, and hence the non-uniform throughput variation in hybrid simulation.

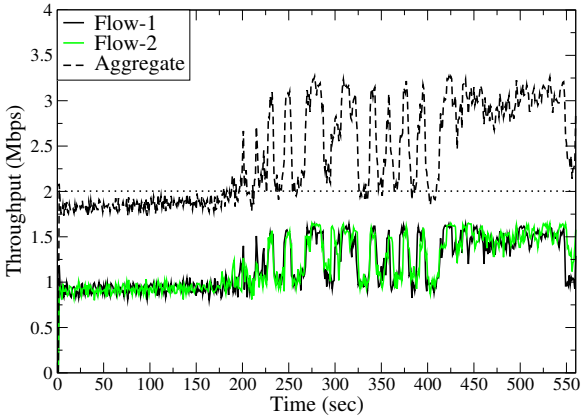

Figure 5.13: The graph captures the impact of temporal variation of signal strength on MAC layer interaction between 2 nodes at a point (X in Figure 5.12). We use the set-up shown in Figure 5.11. Initially the senders can sense each other, hence the two flows are not active simultaneously. After a while, the signal quality drops, and the senders can no longer sense each other, resulting in two flows being active at the same time.

$N1$ and $N3$ senses the channel before transmitting. Channel is perceived busy if signal from one active sender, say $N1$, reaches the other sender, say $N3$. If $N1$ cannot sense $N3$ then the two flows will be active simultaneously, giving higher throughput to both flows.

In our experimental set-up, we replicated the same topology in *ns-2* and MiNT. In *ns-2*, we use the two-ray ground propagation model, with a ratio of 1:2 for hearing range and sense range (22ft : 44ft). In MiNT, the signal propagation is dependent on the environment, and this determines whether one node can hear/sense another node's activity. In *ns-2* the channel capacity is set to 2 Mbps. In MiNT, we set the card's transmission rate to 2 Mbps. For both cases we use a CBR traffic source on $N1$ and $N3$ to pump packets of size 1000 bytes at 2 Mbps, that ensures that both senders are constantly trying to access the channel.

Fig 5.12 shows the throughput of each flow as well as their aggregate in pure simulation using *ns-2*, and hybrid simulation using *ns-2* on MiNT. In pure simulation, till the point the two senders are within the sense distance (44 ft), the flows are constantly interfering. Therefore, the throughput of each flow is around 0.75 Mbps, giving an aggregate throughput around 1.5 Mbps. As soon as the senders move out of sense range, the flows stop interfering and the aggregate throughput shoots up to 3.2 Mbps. Unlike in pure *ns-2* simulations, where the throughput of each flow stays *uniform* at 0.75 Mbps till the distance exceeds the sense range, in hybrid simulation, there are distinct variations in throughput, especially at 26in and 40in distances, where the senders cannot sense each other.

The non-uniform distribution of throughput in hybrid simulation is explained with reference to the signal quality graph for 70 dB attenuation, shown in Fig 5.1. When the signal quality drops due to the presence of a " dark spot", the two senders fail to sense each others' transmissions. Therefore, the two flows can send packets at the same time.

We also observe that with increasing distance the number of spikes in throughput increases. At shorter distance, even if the senders fall in dark spots of each other and cannot communicate, they can still sense each other. However, with increasing distance, the dark spots completely isolate the two senders.

Additionally, there are points where the temporal variation of the signal quality is large. In Fig 5.12, we have marked one point X at distance 58in, where the aggregate throughput is less than the peak value. This is explained using Fig 5.13, that captures the temporal variation of flow throughput at a point using interaction between the two flows. The interference is initially higher leading to channel contention between the senders, but later the interference fades, and both the flows can pump data simultaneously. Pure simulation fails to capture this non-uniform spatial and temporal variation of throughput, which is an artifact of signal propagation characteristics.

5.3.2 Error Characteristics

In this experiment, we show the difference in error characteristics captured using pure *ns-2* simulation and hybrid simulation running on MiNT. We use a CBR traffic source to pump data from one node to another. In pure simulation, we choose an error model that is most commonly used in *ns-2*-based simulation studies, where each packet is corrupted based on a uniform random variable and pre-specified error probability. On the other hand, errors in hybrid simulation occur due to the ambient noise in the environment.

Fig 5.14 plots successful and unsuccessful packet transmission in simulation, hybrid simulation, and real-world communication. The results show that simple bit error models in simulation could produce qualitatively different behavior than those observed in real radio channels as seen on MiNT. Therefore, testing wireless protocols that depend on accurate bit error characteristics becomes much easier and

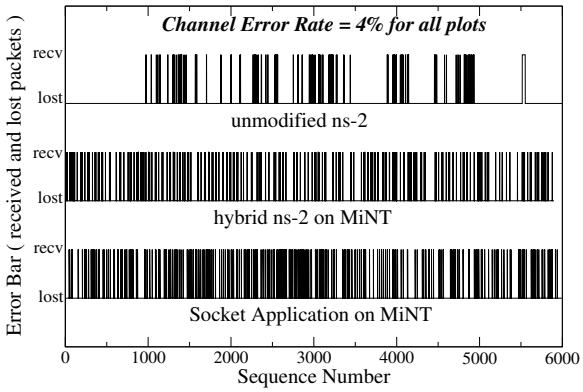

Figure 5.14: The graph plots the packet errors encountered, represented as lost sequence numbers at the receiver, when a 1 Mbps CBR traffic source sends 1000 bytes data packets between 2 nodes for 1 minute duration in a software-only *ns-2* simulation, and in a hybrid *ns-2* simulation run on MiNT. It also shows the packet error rate for the same two nodes when data is sent using socket applications over UDP in real implementation. The packet error rate is fixed to facilitate comparison. produces realistic results with the use of hybrid simulation technique.

5.3.3 Hybrid Simulation Performance

The core computing platform we use is a RouterBoard-230 that has a 266 MHz CPU and is processor-limited. As we add more processing overhead on the system, the maximum throughput we can achieve goes down. We measured the throughput degradation of a single hop as we enable different features: remote tracing, per-packet local tracing, experiment breakpointing, and experiment rollback.

We first look at the impact of different forms of tracing on the maximum throughput achieved between two communicating MiNT nodes. Table 5.2 lists the results. Interestingly, even without tracing, *ns-2* application agents could only achieve 20.5 Mbps as compared to 33 Mbps achievable by a simple UDP flow

Tracing Type	Throughput (Mbps)
No Tracing	20.51
On-line Remote Tracing	17.043
Per-packet Application Tracing	8.423
Per-packet Router Tracing	7.83
Per-packet MAC Tracing	16.08
Per-packet Full Tracing	5.53

Table 5.2: *Hybrid-ns throughput as the tracing is turned on. Due to tracing overhead the available throughput drops.*

running between the same nodes. This is because of the additional processing overhead introduced by *ns-2*, in contrast with a simple UDP sender that needs almost no processing to prepare a packet. To achieve 20.5 Mbps throughput, we did few optimizations to *ns-2* such as use of heap scheduler instead of dynamic hash-based scheduler. This was required to avoid stalling of *ns-2* during the frequent re-hash operation done by the hash scheduler.

Any form of tracing introduces further CPU processing overhead due to string operations done by *ns-2*. The on-line remote tracing is done only for selected events and hence results in least degradation (3.5 Mbps). Per-packet tracing introduces maximum overhead. But even with full per-packet local tracing and on-line remote tracing turned on, the nodes could achieve 5.53 Mbps, enough to saturate a channel operating at 6 Mbps link rate. Current hybrid-ns implementation copies entire protocol packet including its payload between user-space and kernel-space for sending/receiving. As simulated protocols do not care about the payload, one potential optimization is to only copy protocol headers.

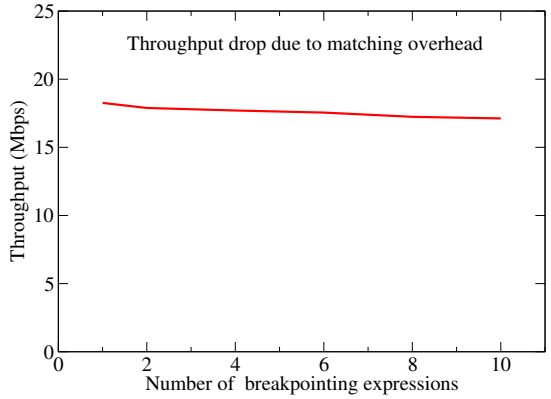

Figure 5.15: *This graph shows that the throughput of hybrid simulation will drop as expressions are added to the breakpoint list.*

Another feature of hybrid-ns is *breakpointing* of experiments. This feature requires matching expressions to trigger the breakpoints when the event occurs. Since matching different fields incurs overhead, the throughput reduces. In Figure 5.15, we show the impact of increasing number of breakpoint expressions on the throughput of hybrid simulation. Despite the CPU bottleneck, the overhead increases only slightly with increasing number of expressions. This is because the expressions are only checked once for each packet, limiting the extra processing burden introduced by breakpointing.

We similarly evaluated the performance of *rollback* feature. This feature requires regular snapshot (using fork() system call) of *ns-2* process running on every MiNT node. Linux kernel's fork() system call automatically uses copy-on-write technique to avoid copying of all the pages at the fork time. This spreads out the throughput degradation to a few seconds after the fork() system call. With even a 1 minute snapshot granularity, the overall throughput degradation was less than 0.25 Mbps.

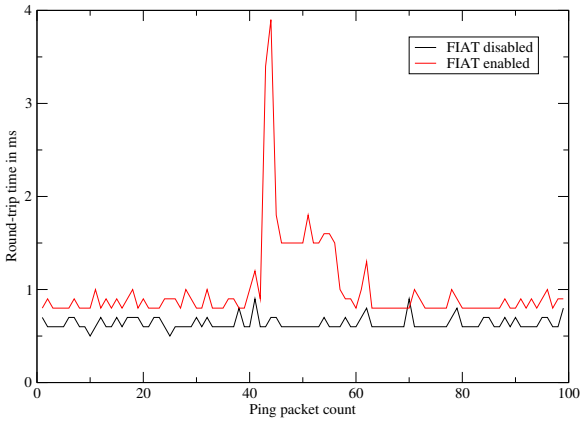

Figure 5.16: *Comparison of round-trip timing for ping with packet size 1200 bytes, and FIAT enabled and disabled.*

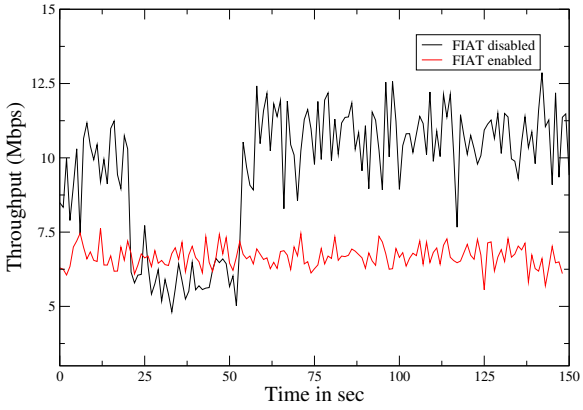

Figure 5.17: *Comparing the throughput measured for UDP traffic between 2 nodes with and without FIAT inserted in the protocol stack.*

5.4 Evaluating Fault Injection and Analysis Tool

In this section, we evaluate the Fault Injection and Analysis Tool by showing how it affects normal performance of a protocol. This indicates the degree of intrusiveness of the tool. Next, we show with an example on the implementation of AODV routing protocol from Uppsala University how FIAT can be used to study the correctness of protocol implementation.

We first conduct a simple ping test where we send out 100 ping packets with a size of 1200 bytes between two nodes over the wireless medium. We ensure that the link quality is good such that the link losses will not introduce additional delays in our measurement. We are interested in measuring the processing latency due to FIAT. The experiment is repeated with and without the Fault Injection layer inserted in the network stack. The number of rules actively matched when FIAT is inserted is three. Figure 5.16 shows the round-trip time for the ping requests. The average time taken for the case without FIAT is around 0.6 ms, whereas after inserting Fault Injection layer the average round-trip time has gone upto 0.8 ms. This shows that on an average the Fault Injection Layer is introducing an overhead of about 30% in the system we are using. The processing latency can be reduced by using faster processors.

We conduct another experiment with the same setup. In this case, we measure the UDP throughput between the same two nodes with and without the Fault Injection layer inserted in the protocol stack. The result is similar to the observed result in the ping experiment. From Figure 5.17, unless there is a variation in the channel quality, the overhead introduced by the packet processing functions of the FIAT layer is around 30%. With increasing number of rules, the processing latency will increase further, hence it is advisable to select the rules carefully to keep it as low as possible when using FIAT.

The following experiments are aimed at showing how FIAT can be used in practice. We choose an implementation of the AODV routing protocol from Uppsala University, AODV-UU for testing. In the fault injection script we specify rules to drop all the HELLO messages that are sent out from a node. The messages are dropped at the receiver node, whose IP address in this case is 192.168.100.196. The sender node's IP address is 192.168.100.189. Initially, the sender node finds a route to the destination node, and that can be seen in Figure 5.18. It shows a *VAL*

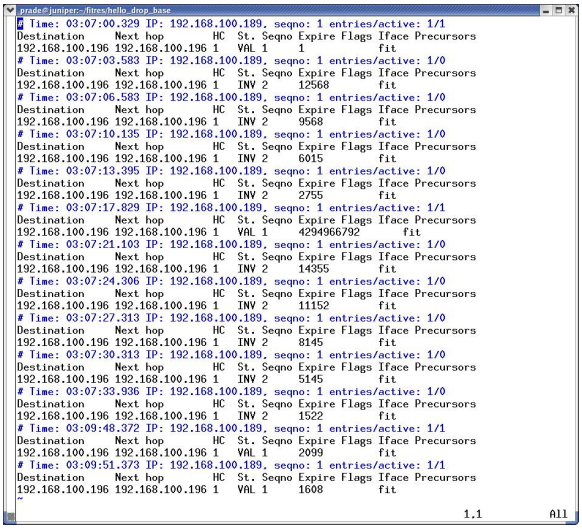

Figure 5.18: *The figure shows the Routing Table output for the AODV-UU. It shows that as the HELLO packets are admitted by the FIAT layer, the path is restored proving that the implementation is correct.*

route from 192.168.100.189 to 192.168.100.196. But after the script is started, the HELLO packets are dropped, and the route becomes invalid, indicated by the *INV* flag in the routing table. The log of events for the routing protocol are shown in Figure 5.19, where it can be seen that HELLO message failure is received. Finally, when the script finishes, the HELLO messages again go through, and the route is restored. This indicates the correctness of the implementation of the HELLO protocol in AODV-UU.

In the second case study involving FIAT, we use 4 nodes which are in a multi-hop setup. There exist multi-hop paths with hop count two between source node (IP addr: 192.168.100.196) to destination node (IP addr: 192.168.100.190) through other intermediate nodes. Initially, when packets are sent from the source to the destination, the AODV protocol discovers the route through node with IP address 192.168.100.191. After a fixed number of packets are exchanged, FIAT is used

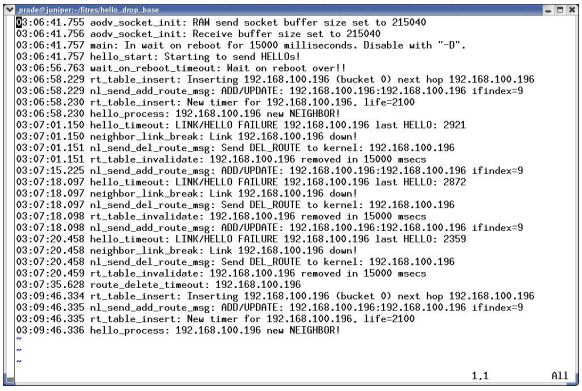

Figure 5.19: *The log of the events when the HELLO packets are dropped by the FIAT layer at the receiver, and then when it is again restored.*

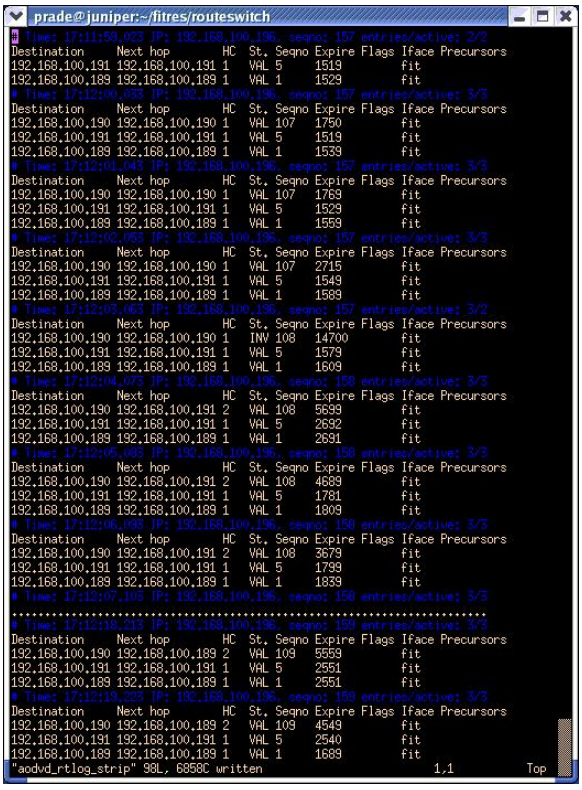

Figure 5.20: Routing Table log – part I

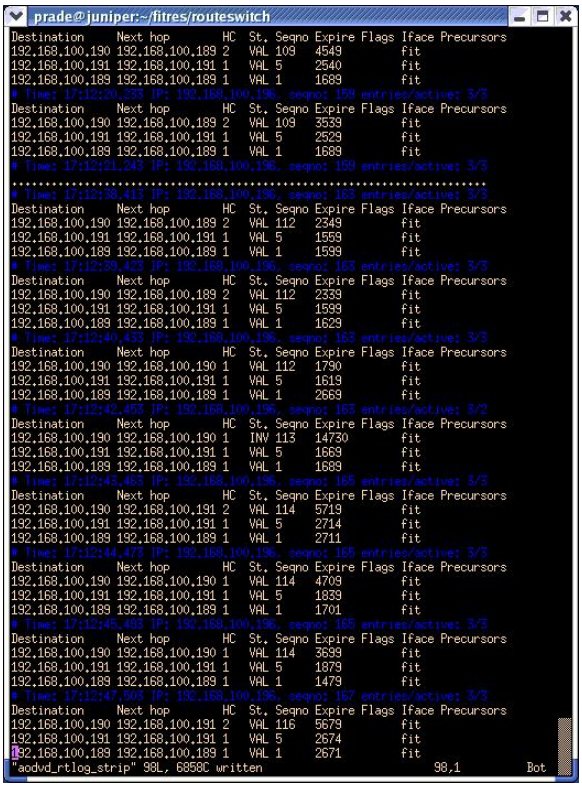

Figure 5.21: Routing Table log – part II

to drop all packets going through 192.168.100.191, thereby disabling the path to the destination node. On route failure, the AODV protocol successfully discovers the alternate path through node 192.168.100.189. Again, when the path through 192.168.100.191 is restored by allowing all packets, this path is rediscovered, and the route switches. The routing table in Figure 5.20 and 5.21 recorded the changes as it happened during the protocol run indicating the correctness of route discovery implementation in AODV-UU.

5.5 Case Study: Ad-Hoc Transport Protocol (ATP)

In order to demonstrate the usefulness of MiNT as an experimentation platform, we studied a cross-layer MANET transport protocol called ATP [Kar03] using MiNT's hybrid-ns simulation feature. Although, ATP has been comprehensively evaluated in pure *ns-2* simulations, its behavior has not been studied on a real testbed. Our experimental study revealed several important characteristics of the protocol that we discuss in this subsection.

In ATP scheme, every intermediate node measures queuing and transmission delays for each packet passing through it. The sum of exponentially averaged queuing and transmission delays yields the *average packet service time* experienced by all the flows going through the intermediate node. In the equilibrium condition, each ATP flow attempts to maintain exactly one data packet on every router along the path [Kar03]. The service time therefore reflects the ideal dispatch interval for all the flows competing over the bottleneck link. Every packet bears the maximum service time encountered on any of the intermediate hops. The bottleneck service time is communicated back to the sender, which adjusts its packet dispatch interval to match this service time.

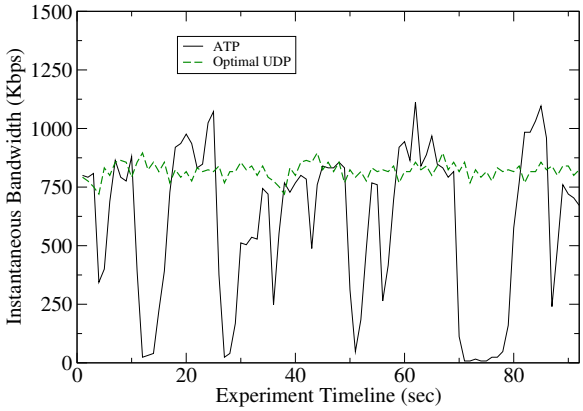

Figure 5.22: *Fluctuations in ATP's bandwidth estimation. The channel bandwidth is fairly constant, as seen by an optimal UDP stream sent over the same 3 hops.*

5.5.1 ATP's Inaccurate Bandwidth Estimation

One of the key issues we observed with ATP is bandwidth under-estimation. Figure 5.22 shows the fluctuations in bandwidth estimation by an ATP flow originator performing FTP upload to a node that is 3 hops away. The same figure also shows the optimal bandwidth as seen by a UDP flow. The optimal flow was found by sending a UDP stream at different rates until the maximum was achieved. The stability of the optimal UDP flow suggests that the channel bandwidth fluctuations are negligible, and most of the bandwidth fluctuations are internal to the ATP protocol itself.

Further experimentation revealed the root of the problem to be the service time measurement metric proposed by ATP. The problem with the overall-service-time approach is that it couples the queue-size management with rate estimation, which leads to traffic fluctuations, and in turn non-optimal estimation of channel bandwidth [RJV94]. More concretely, it is very hard to maintain exactly one data packet from each flow on every router. If there is even a slight change in transmission time of a single packet, a queue (say of two packets) builds up on the router for the rest of

the epoch. Clearly, an extra packet in the queue does not indicate any change in the network bandwidth. However, in the next epoch ATP sender proportionally reduces its sending rate (to half in this example) to bring the queue down to one packet. It is in these epochs, that an ATP sender under-estimates the path bandwidth.

5.5.2 ATP's Flow Unfairness

It has been shown in [Kar05] that ATP gives flow fairness. In Figure 5.23, taken from [Kar05], it can be seen that as a second flow is introduced, the bandwidth is shared fairly between the two flows. However, we tested several scenarios and came up with two common ones where ATP's behavior shows significant unfairness. These scenarios are depicted in Figure 5.24. The first example (Figure 5.24 (a)) corresponds to the hidden terminal scenario. Here one wireless link's transmission is inhibited by another link, eventually leading to unequal bandwidth allocation between the two. Table 5.3 shows the resulting bandwidth distribution between the two flows. Flow 1 originating from the hidden node gets much lesser bandwidth than Flow 2 originating from the inhibiting node. Although one could attribute this problem solely to the 802.11 MAC layer, this problem can indeed be addressed at the transport layer as demonstrated by fairness of optimal UDP flows going over the same network.

The second example (Figure 5.24 (b)) corresponds to a general channel space sharing scenario. Concretely, ATP allocates a radio channel's bandwidth fairly among flows from a single node, rather than among all flows from all nodes that share the radio channel. As a result, a flow emanating from a node with fewer flows tends to get a larger than fair share of channel bandwidth. This is shown in Table 5.4 where Flow 4 gets much larger than its fair share, while Flow 1 and Flow 3 suffer.

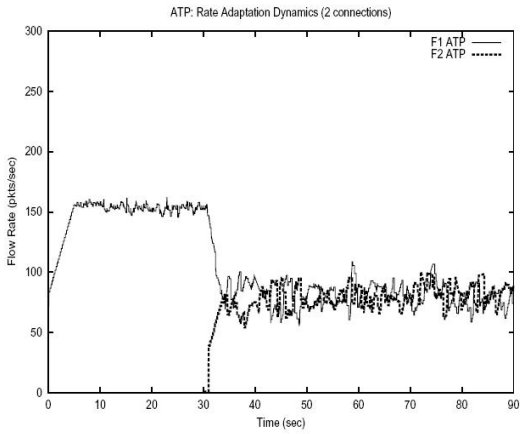

Figure 5.23: *This graph from [Kar05] shows the simulation results of running two flows that uses ATP. It can be seen that the bandwidth is shared fairly between the two flows when the second flow is introduced.*

Flow	**ATP Thruput (Kbps)**	**Optimal Thruput (Kbps)**
Hidden (Flow 1)	570.4	985.8
Inhibitor (Flow 2)	1104.6	1186.4

Table 5.3: *ATP accentuates the hidden terminal problem.*

Flow Id	**ATP Thruput (Kbps)**	**Optimal Thruput (Kbps)**
1	383.0	641.6
2	590.1	641.6
3	484.7	641.6
4	1010.8	641.6

Table 5.4: *ATP's fairness in a general channel sharing scenario.*

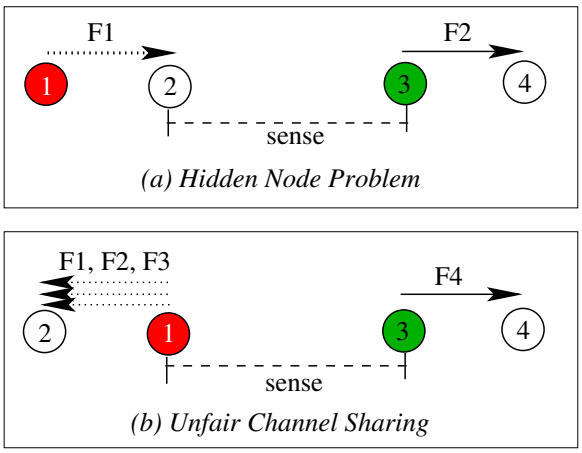

Figure 5.24: *ATP's unfairness scenarios: The wireless node getting a lesser than fair share of bandwidth is numbered 1 (and colored in red or white), whereas the one getting a larger share is numbered 3 (and colored in green or black). (a) Node 1 lacks information about Node 3's transmissions, attempts its communication at inopportune times, and eventually backs off unnecessarily. (b) Flow F1, F2, F3, and F4 all share the same channel, but ATP, like most other transport protocols, allocates more bandwidth to F4 than to others.*

5.5.3 Discussion

This protocol study demonstrates the usefulness of different MiNT features. Specifically, the ability to perform hybrid-ns simulations enabled us to re-use the pure ns-2 code written for ATP. Similarly, the ability to reconfigure topology through MOVIE enabled us to come up with specific topologies that revealed the protocol weaknesses. MOVIE enabled us to visualize the queue size on every ATP router, thus helped pinpointing the reason behind ATP's fluctuations.

5.6 Case Study: Evaluating Ad-hoc Routing using ETT metric

In this case study, we evaluate the impact of using a routing metric, called Expected Transmission Time (ETT) in ad-hoc routing protocols [DPZ04]. We focus in understanding the benefit of using testbeds over simulation, if any, in evaluating new routing metrics that are being proposed for different types of wireless networks. It has been pointed out in prior research that in most real deployments of multi-hop wireless networks the shortest path metric used for routing may not yield the best performance [CABM03]. An alternate routing metric proposed is the Expected Transmission Count (ETX) [CABM03], that measures the expected number of transmissions, including retransmissions, needed to send a unicast packet across a link. The measurement of ETX path metric is based on the loss rate on the channel, and does not take into account the link bandwidth. ETT augments this by factoring in the link bandwidth in order to select the link metric for routing. ETT can be defined as the expected amount of time it would take to successfully transmit a packet of some fixed size S on that link. The measurement of ETT takes into account the link loss rate as well as the bandwidth of the link. The path metric based

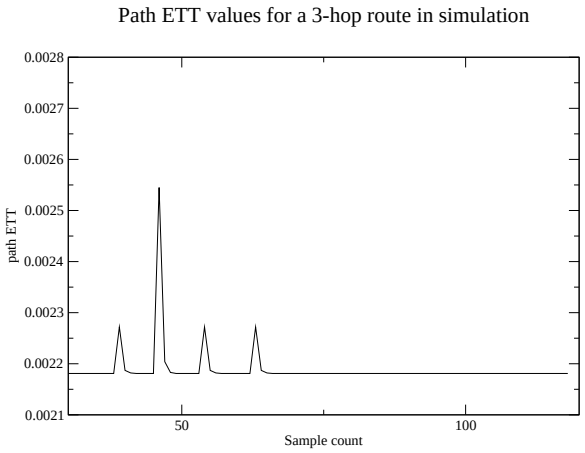

Figure 5.25: *The variation of link ETT is limited in simulation due to lack of error models and temporal variations in the link. This is reflected in a relatively stable route ETT in the graph.*

on ETT adds up the ETT for each link, and the routing selects the one with a lower value of the summation. For a path with n hops, the path metric is thus defined as, $\sum_{i=1}^{n} ETT_i$.

In our experiment, first we use *ns-2* to compute path ETT metric for a 3 hop route in a simulation environment with 20 nodes. We use the shadowing model, with a path loss exponent for shadowed urban area ($\beta = 3.0$), and the shadowing deviation (δ) value set to 2. We measure the change in the path ETT value at regular intervals in order to study how it varies in simulation. A similar experiment is conducted on MiNT where we measure the path ETT metric for 2 routes starting from the same source and ending at the same destination, but taking 2 distinct routes. In the testbed, the variation of the path ETT is again measured at regular intervals for the two routes. The results of the simulation is shown in Figure 5.25. It shows very little fluctuation in the path ETT values collected over a period of time. The same data for the MiNT, shown in Figure 5.26 shows frequent fluctuations of the

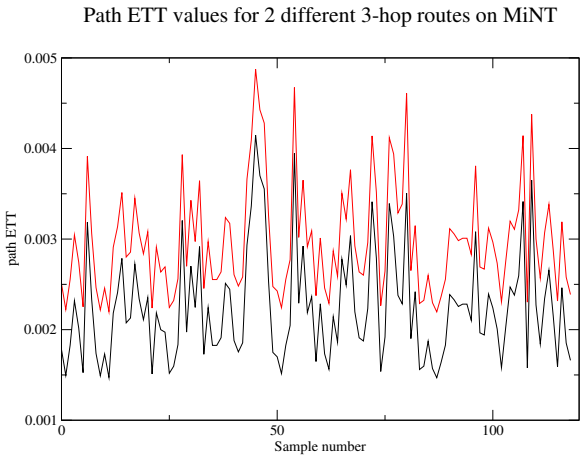

Figure 5.26: *The link ETT in MiNT varies over time leading to a fluctuation of the route ETT. The graph shows the variation of route ETT for 2 routes between a source and destination.*

route ETT values on both the routes, and often the shift in the values is dramatic enough to warrant a switch in the route for better performance. This is an effect of changing link conditions in the testbed due to temporal variations in the link, as well as changes in the error rate due to external interference. This shows that while studying the impact of path metrics, like ETT it is essential to rely on testbeds.

We did another macro measurement to show how choice of path will affect throughput. In this experiment, we choose a source destination pair on MiNT, and configure 2 wireless NICs on all the nodes in the path in two non-overlapping channels. Traffic is sent along both the paths, and the throughput recorded. The variation in the throughput along both the paths, as shown in Figure 5.27 shows that path ETT metric should be computed at regular intervals in order to make optimal choice of the route for flows. Path ETT metric as seen in simulation will not be able to highlight this fact effectively, showing the importance of using MiNT-like testbeds for such studies.

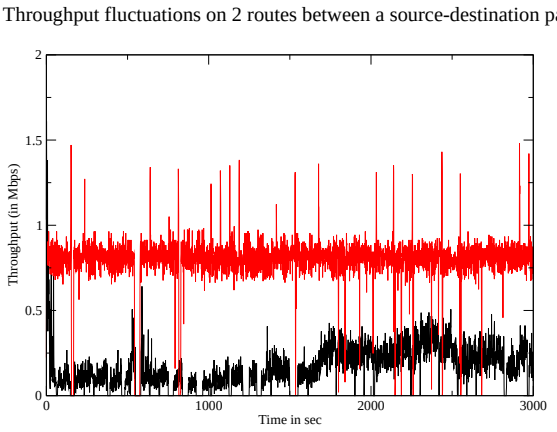

Figure 5.27: *The graph shows the variation of throughput between a source destination pair between two routes. The fluctuations in the throughput indicates that route ETT metric must be computed frequently if optimal path has to be used at all times.*

5.7 Critique on Remote Usage of MiNT

We have designed MiNT to be used as a remote experimentation platform. However, we did encounter a few hurdles while using MiNT remotely. In this section, we record our experience of using MiNT to run a simple hybrid simulation script from a remote location[1]. In this experiment, we use 8 nodes to set up a wireless mesh network. We introduce 3 competing flows and record their throughput. The main purpose of this experiment is to test the ease of topology setup, setting up an experiment, and collection of results.

The first step in setting up an experiment on MiNT is to configure the topology by selecting the appropriate number of nodes. The only interface to a remote user for selecting the nodes is MOVIE. It allows one to select the nodes that are present in the testbed arena. The nodes show up as icons in the GUI. The user can configure

[1]This experiment was carried out from India over a 256 Kbps broadband connection with the testbed setup at Stony Brook University, New York

topology based on the link quality and signal strength among the different nodes. In order to change topology, the remote user drags the icons in the interface and the nodes are displaced physically in the real testbed, with simultaneous adjustments of the link qualities. However, physical movement of the nodes is not instantaneous, hence the user must wait for the change to happen. Often, due to nature of the connectivity of the remote user, the updates of the link quality is bursty, thereby giving the user a non-smooth view of the changes in the testbed. One remedy to this could be providing the user with a ”real” view of the testbed by streaming the data that is anyway collected by the tracking webcams. The images captured by the tracking server could be fed to a remote user to provide her with a physical view of the testbed, in addition to the GUI view.

The other problem that we often encountered is related to the movement of the Roombas. When the Roomba is required to make an angular movement, it often goes into a loop. Instead of stopping after performing a fixed number of angular motions, it runs into an infinite circular motion. This appears to be a bug in the firmware. The solution lies in using the newly released API which provides better control over the movement instead of going through the Spitfire-based hack that we have used for the current version of the node. It has been tested that use of the new API for controlling Roomba movement is much more robust that the current design. We plan to switch over to the new API in the next prototype.

Another requisite step in setting up an experiment is to configure the different parameters on the nodes. For example, in pure simple one can configure the transmit power, sensitivity threshold and other similar parameters. In order to configure such parameters on MiNT, as a remote user one needs access to each node card parameters. Currently, MiNT does provide access to several card parameters through MOVIE, but it is quite likely that it may not match the range of parameters that are available in pure simulation. In such cases it might be required to extend the card

APIs, or may be required to switch to different cards in order to get more extensive APIs.

Chapter 6

Conclusions

This chapter summarizes the contributions and results of the dissertation. We conclude by presenting some future directions that can be pursued on this topic.

6.1 Summary of the Dissertation

The design of a miniaturized multi-hop wireless testbed, and tools for making it usable, is the key focus of the dissertation. It is worth noting that pure simulation forms the cornerstone of most research in the domain of wireless research. A simulation experiment can only be as accurate as the models that are used to capture the behaviors of the different layers, like the physical layer, MAC layer, and so on. Hence, lately there has been quite a few works that have started looking at the pitfalls of pure simulation. It has been pointed out that often the inaccurate modeling of the wireless physical layer, like the signal propagation and link error characteristics, leads to inaccuracies in the results. It is possible to come up with more detailed models for these layers, but at the cost of higher computation. The other alternative being embraced gradually across the community is to build multi-hop wireless

testbeds, and validate the simulation results on these testbeds. But setting up a full-fledged multi-hop wireless testbed with a tens of nodes is a non-trivial task. The difficulty often stems from the fact that nodes has to be placed physically apart from each other by a large distance in the order of hundreds of meters. Managing such a spread out testbed is an administrative nightmare. Keeping this in mind, we came up with a novel scheme of designing a *miniaturized* multi-hop wireless testbed.

We have shown that through use of radio signal attenuators it is possible to scale down the communication distance between two wireless nodes. With this as our building block, we created a multi-hop wireless testbed of 12 nodes in a space of $11\ ft$ by $14\ ft$. Mobility is an important aspect of wireless experiments. In order to make the nodes mobile, we have engineered the Roomba robotic vacuum cleaners to act as our mobile platform. A lightweight embedded system from RouterBoard is used as the computing platform and has been mounted on the Roomba. It is fitted with four wireless interfaces, out of which one is dedicated for control and the remaining three are used for experiments. The fidelity of this miniaturized setup is tested against unmodified setup in the dissertation. It is shown that across each layer in the network stack the consistency of the results are preserved compared to the non-attenuated version of the testbed.

In order to have the testbed operational, some other solutions that are indispensable includes a mechanism to track the position of the nodes in the testbed arena, ways to move the nodes around in the testbed arena without collision, and ability to keep the testbed operational with minimal human intervention. We have designed a vision-based tracking system by ceiling-mounting multiple webcams. The webcam feeds are merged to compose the snapshot of the testbed. Simple image processing techniques are used on the captured images to identify each node. We show that with 7 colors it is possible to identify a large set of nodes. The technique for collision-free movement of the nodes makes use of the periodic feeds from the

tracking system to detect proximity of nodes. Based on the proximity of the nodes, all nodes are stopped, and the nodes are allowed to move one by one. One downside of the method is that it is a greedy approach to move the nodes to their correct location without much attention to the time taken. We show that with 6 nodes the time taken to move the nodes around can go up to a few seconds. Finally, we want to keep the testbed operational 24x7. This means that the batteries on which the nodes are running must be recharged automatically without operator intervention. The Roomba provides a auto-docking feature and we engineer that feature to automatically dock the nodes for recharging.

Besides designing the infrastructure for setting up the multi-hop wireless testbed, we have also designed and implemented a suite of software tools to be used on MiNT. Since simulation is in vogue for a while now, we designed a hybrid simulation technique that allows using the existing simulation scripts, with minimal modification on the testbed, but can improve the accuracy of the results by eliminating the use of models at the lower layers. In the hybrid simulation approach, we replace the link, MAC and physical layer with the real implementations. All packets are exchanged over the real wireless medium, thus eliminating the use of models for the signal propagation or link error characteristics. We have shown that the use of hybrid simulation can improve the accuracy of the results as compared to the pure simulation results. Another tool designed to reduce the development time for implementors of protocols on the testbed is the Fault Injection and Analysis Tool (FIAT). The tool injects user-defined faults at runtime based on scripts written by the user, and flags errors if there is an anomalous behavior. We show the intrusiveness of the tool in the context of protocol testing. The results indicate that the use of the tool does not modify protocol behavior and can be used effectively in debugging protocol implementations.

Another useful software component in MiNT is the MiNT Visualization and

Control Interface (MOVIE) which is the front-end to the entire testbed. MOVIE provides all the standard features to control the nodes in the testbed. Each node is represented as an icon in MOVIE, and their respective positions in the testbed is updated in real-time. Besides acting as the interface for the resources in the testbed, MOVIE also acts as the control center for running experiments. It has new interfaces compared to the existing GUIs through which one can rollback or pause execution of hybrid simulation experiments.

Finally, we have used MiNT to execute one hybrid simulation experiment using a wireless protocol, called Ad-Hoc Transport Protocol (ATP). This is a demonstration of how to use the testbed to execute simulation experiments that were tried previously in the pure simulation mode. The experience of using the testbed remotely was also insightful for us. We also summarized the pros and cons of using the testbed remotely, where we used MiNT over a broadband connection from India while the set up was installed at the University site in Stony Brook.

6.2 Future Work

There are some immediate issues that are not addressed in this dissertation, which forms a list of short-term goals for the future work. There are few other directions which are more longer term. In this section, we summarize some of these future directions.

A simple extension of the Fault Injection and Analysis Tool (FIAT) would be to use it in conjunction with the hybrid simulation. The main mechanism of FIAT is packet matching. The same packet matching technique can be built into hybrid simulation by allowing all *ns-2* packets to pass to a layer that inspects the packets before dispatching it for transmission. The scripting mechanism for defining faults and the action triggering mechanism can be used as it is. The details of how FIAT

can be extended for hybrid simulation is explained further in Section 4.4.1.4.

Another feature of convenience that is lacking in MiNT at present is a physical view of the testbed. The only view of the testbed that the user gets is through MOVIE, the graphical user interface for the testbed. MOVIE does show node location in real-time by changing positions of node icons through periodic updates from the testbed. However, when one is accessing over a slow network link, the updates are often slow to come, and the user is left waiting to figure out if his changes took place. It would be convenient to have a webcam interface that shows the live status of the testbed. Since, webcam feeds are anyway captured in the testbed, the same can be streamed to the user. The user may optionally choose to enable it.

Currently MiNT uses 12 nodes. The plan is to extend it into a bigger testbed with larger number of nodes. This will certainly throw up issues with scaling and robustness of the nodes. From our experiences with managing the testbed, we realized that robustness of such a system put together using off-the-shelf components is often a cause for concern. Therefore, as the testbed is extended into a larger setup, there will be needs for better and more efficient interfaces for management of the testbed. For example, ways to detect faults in each node, namely identifying which network card is failing, could be useful tools to reduce downtime of the testbed.

As the testbed is extended, other means of tracking the nodes can also be explored. A mesh of IR receivers can be created on the floor of the testbed, and their positions can be marked. If each node is now equipped with the IR emitter, then the location of the node can be approximately determined. To get accurate positioning, this can be combined with the odometer reading that can be accessed using the Roomba Serial Console Interface (SCI). The introduction of the Roomba Serial Console Interface also opens up new possibilities for designing some of the subsystems in the testbed. The details of Roomba SCI can be used for redesigning some of the subsystems has been discussed in detail in Section 3.7.

Appendix A

Steps to Assemble a MiNT Node

A.1 Hardware Components with Vendor List

The following list of equipments is a minimal set of items required to build a MiNT node. This list assumes that the user is familiar with some common terms, like soldering and the equipments necessary for performing the action, as well as, some items, like Velcro, used for fastening two objects.

A.1.1 Wireless Computing Platform

The following are the necessary items for building the wireless computing platform.

1. **RouterBoard RB-230**

   - *Vendor: http://www.mikrotik.com*
   - Comments: RouterBoard 230 is a small form-factor embedded computing board with 266 MHz CPU on it. The board does not come with any accessory, like the chassis which must be purchased separately from the same vendor. Hence, one must buy a Compact Flash card which is used

as the non-volatile storage for storing the OS images, and other applications. If the storage requirement is high, then it is necessary to add a higher capacity disk on to the RouterBoard. The RouterBoard has a IDE port. Additionally, it is required to buy the RAM separately.

2. **RB-14 PCI Extension Board**

   - *Vendor: http://www.mikrotik.com*
   - Comments: Since there is only 2 slots for mini-PCI cards on the RB-230 it is required to buy this PCI adapter that allows connecting 4 mini-PCI cards to the RB-230. All the wireless cards we are using are mini-PCI cards. Additionally, because of size and design limitations of the RB-230 chassis, it is required to buy a PCI Riser card, RB-71 from the same vendor.

3. **2.5" Hard Disk**

   - *Vendor: Any standard disk manufacturer*
   - Comments: This is the same hard disks used in laptops. It is advisable to check the Routerboard specification which clearly mentions the pin count of the disk. The IDE cable also needs to purchased separately for connecting the disk to the IDE connector on the board. Unfortunately, the RB-230 chassis is not designed well to accomodate a disk, hence requires some improvisation to place the disk inside the box.

4. **Wireless Network Interface Cards**

   - *Vendor: http://www.netgate.com*

- Comments: The network interfaces have to be fitted with antennas, and that requires connectors. The connector types need to be carefully identified to make it easier to choose the antennas with connector types.

5. **Fixed-point Attenuators**

   - *Vendor: http://www.hdcom.com*
   - Comments: A point to remember while choosing the attenuators is that they are available for different frequency range of 2.4 GHz, and 5.8 GHz. While using 802.11a enabled wireless NICs, it is required to choose attenuators that can attenuate signals in the 5.8 GHz range. Secondly, since we are using fixed attenuators, it is useful to purchase multiple attenuators that can be combined to create various attenuation levels.

6. **Low-gain Antennas**

   - *Vendor: http://www.netgate.com*
   - Comments: These are 2.4-5.8 GHz Omnidirectional Swivel Antenna with cable and U.fl connector. With each choice of antenna careful attention must be paid to the connector types. The cable connecting the antenna to the wireless card should have the matching connector types to avoid use of additional adapters in between, which also leads to leakage of signal. In our design, we have the attenuators between the antenna and the card connector.

7. **Battery**

   - *Vendor: Any portable laptop battery manufacturer*

- Comments: The tradeoffs in chooisng a battery is the size/weight of the battery to battery lifetime before recharge. In our current setup, we are using *118 Watt Hour Universal Li-Ion Laptop battery (NBMATE-118)*.

A.1.2 Mobility Setup

The following items are required for building the remote controlled mobility platform.

1. **Roomba Robotic Vacuum cleaner**
   - *Vendor: htt://www.irobot.com*
   - Comments: These are standard Roomba vacuum cleaner available from any consumer electronics shop at retail price of $249.
2. **SpitFire Universal Remote Controller**
   - *Vendor: http://www.SmartHomeUSA.com*
   - Comments: The Universal remote controller can be fitted with an extension cable to for pin-pointing the IR beam to specific directions. This is a stick-on block with a very small size that is useful in the overall assembly of a MiNT node.
3. **Diodes**
   - *Vendor: RadioShack*
   - Comments: Diodes with reasonably high rating as available in RadioShacks for hobby electronics is required for devising the common charging circuitry of the Roomba battery and the laptop battery used for charging the RouterBoard (refer Figure in 3.5.1).

A.2 Assembly Instructions

This section covers the assembly instructions for putting together the items mentioned in Section A.1 for building a stand-alone MiNT node.

A.2.1 Hardware Assembly Instructions

A MiNT node is assembled in completely modular fashion. The steps involved in assembling a node are: (a) assembly of the wireless computing box, (b) assembling the mobility platform, including the universal remote controller device, (c) putting modules (a) and (b) together.

A.2.1.1 Wireless Computing Platform Assembly Instructions

1. Open RouterBoard chassis and insert the memeory module on to the RB-230. The CF card can also be inserted at this point if an image has already been burnt into it for booting using that image. Otherwise, it is possible to insert the CF card later. It is required to burn an image into the CF card and that requires any available card reader.

2. The PCI riser card, RB-71 is placed on the PCI slot on the board. The PCI-to-miniPCI adapter, RB-14 is connected to the riser card. Four miniPCI wireless NICs are placed on to the slots on the RB-14. Before placing the cards in the slots, it is advisable to attach the connectors securely (if needed it is better to use clear tapes to secure the connector end that sits on the NIC).

3. The placement of the hard disk inside the chassis requires some improvisation because the chassis is not designed for placing a hard disk. A suitable place to attach the hard disk is on the smaller side of the box near the IDE connector on the board. To secure the hard disk in the chassis, two small grooves are

cut on the box, and for cooling purpose a hole is drilled on the body of the chassis. Remember to put the jumpers on to the disk because the disk will appear as the slave, and the CF card is the master.

4. 2 openings on each of the longer sides of the chassis is created to allow for the antenna connectors to come out.

A.2.1.2 Mobility Platform Assembly Instructions

1. The Roomba battery charges when it places itself on the docking station. While the Roomba battery we charge the laptop battery simultaneously. First, figure out the positive and negative charging leads on the Roomba by measuring the voltage using a multimeter. Next, solder a diode to the +ve lead on the Roomba to prevent the Roomba from drawing current from the laptop battery. Solder another wire to the -ve end. The other end of the +ve and -ve cable goes to a coaxial charging tip that plugs into the laptop battery for charging (refer Figure in 3.5.1

2. Since the laptop battery charges both the Routerboard and the Universal Remote control device (Spitfire), therefore we need a Y-connector (available in RadioShack). With proper charging tips connected to the other end of the Y-connector, we can charge both the devices from the same battery.

3. For programming the universal remote control device, one spitfire must be programmed manually using the remote control of the Roomba. The Roomba codes for different actions are recorded in one Spitfire. We require all the Spitfires to send out *exactly* same signals. In order to remove any perturbation that may happen while training several Spitfires separately and manually, it is advisable to use a chip programmer device to copy the learned code (stored

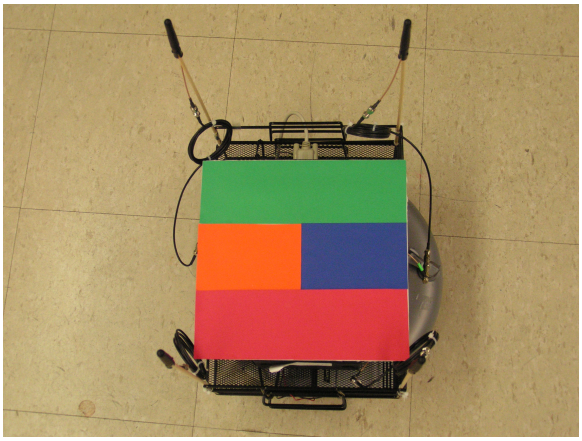

Figure A.1: *The top view of a MiNT node.*

in EEPROM – ATMEL chip) onto the other Spitfires. Finally, the IR extender is used to stick the IR emitting block to the Roombas IR receiver.

A.2.1.3 Putting It All Together

1. In order to place the entire setup on the Roomba, it is necessary to create a two-layer rack. Choose a suitable stackable rack and fix it onto the Roomba using screws. We use stackable wiremesh trays for the purpose. *Ensure that the placement of the trays does not hinder free movement of any part of the Roomba, specially the head.*

2. The battery and the Spitfire is placed on the lower rack. The Routerboard is placed on the top rack The antennas are placed on four corners.

On assembly a MiNT node may look similar to the picture shown in Figure A.1 and Figure A.2.

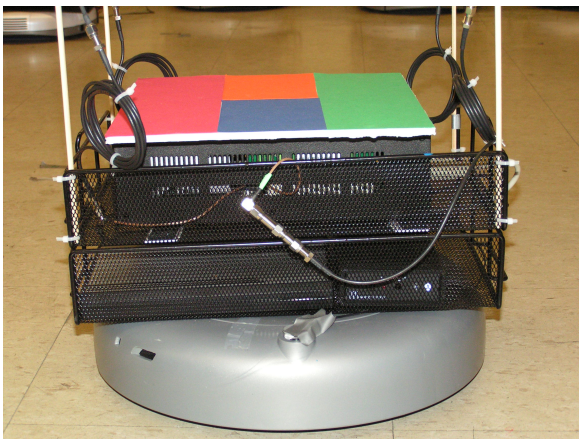

Figure A.2: *The side view of a MiNT node.*

A.2.2 Software Installation Tips

The RouterBoard RB-230 can be booted with a distribution, called the pebble distribution from http://www.nycwireless.com. The driver used for the wireless cards is available from http://madwifi.org. The driver is specific for cards using Atheros based chipset.

During the network setup, it is useful to allow one interface to acquire a DHCP address so that the device is accessible as soon as it boots up.

A.3 Cost Break-up of a 12-node MiNT set-up

We present in Table A.1 a cost break-up for setting up a 12-node testbed of MiNT nodes. This inlcudes the expenditure for setting up the 12 core nodes, in addition to the other infrastructure costs, *viz.* the tracking system, and the control node.

Item	Cost ($)
Wireless Node	
RouterBoard RB-230	330
Wireless NICs and antennas	100x4 = 400
MiniPCI Adapter	65
Attenuators	40x6 = 240
Hard Disk	75
External Laptop Battery	170
Spitfire	135
Roomba	250
Total	**1665**
Tracking Server	
Desktop PC	300x3=900
QuickCam 4000	100x6=600
Total	**1500**
Control Server	
Wireless NICs and antennas	100x3 = 300
Desktop PC	300
Total	**600**

Table A.1: *Cost breakup of MiNT infrastructure.*

Appendix B

Hybrid Simulation

B.1 Hybrid Simulation Script

We present a sample hybrid simulation script using *ns-2*.

```
set opt(mode)        dsrtt
set opt(ctype)       sum
set opt(settimer)    55
set opt(hello)       2
set opt(filename)    test
set opt(start)       5.0
set opt(end)         25.0

set BgNodes  0
set BgFlows  0
proc getopt {argc argv} {
  global opt
```

```
  lappend optlist cp nn seed sc stop tr x y mode settimer ctype
  for {set i 0} {$i < $argc} {incr i} {
set arg [lindex $argv $i]
if {[string range $arg 0 0] != "-"} continue
set name [string range $arg 1 end]
set opt($name) [lindex $argv [expr $i+1]]
  }
}
getopt $argc $argv
set ActiveFlows     90 ;# number of activeflows



source emul-header.tcl


#Define a 'finish' procedure
proc finish {} {
global rec0 rec1 namtrace tracefd filename ns opt
global sink sink0 tnode
global ActiveFlows Tot script_start
#Close the output files
set now [$ns now]
set t [expr $now - $opt(start) - $script_start]
puts "Time : $t"
set sum 0
set count 0
```

```
set avrg 0
set actbytes [$sink0 set bytes_]
set thruput [expr $actbytes/$t]
set thruputM [expr ($thruput * 8)/ 1000000.0 ]
lappend olist $thruputM
for { set i 0 } { $i<$Tot } { incr i } {
set totbytes($i) [$sink($i) set bytes_]
set tr($i) [expr $totbytes($i)/$t]
set tr_($i) [expr ($tr($i) * 8)/1000000.0]
lappend olist $tr_($i)
set avrg [expr $avrg + $tr_($i)]
if { $totbytes($i) != 0 } {
set sum [expr $sum + $totbytes($i)]
set avrg [expr $avrg + $tr_($i)]
incr count
}
}
if { $count != 0 } {
set avrg [expr $avrg/$count]
#set dsroh [expr $dsroh/(1.0*$sum)]
}
puts "Throughput : $olist \n"
close $tracefd
close $namtrace

exit 0
}
```

```
#####################################################
# Define a procedure which periodically records the
# bandwidth received by the traffic sinks.

proc record {} {
global rec0 rec1 totbytes0 totbytes1 bw0 bw1
global sink0
sink1

#Get an instance of the simulator
set ns [Simulator instance]

#Set time for recalling procedure
set time 1.0

#How many bytes have been received by the traffic sinks?
set bw0 [$sink0 set bytes_]
set bw1 [$sink1 set bytes_]

# Total number of bytes recived so far
set totbytes0 [expr $bw0 + $totbytes0]
set totbytes1 [expr $bw1 + $totbytes1]

#Get the current time
set now [$ns now]
#Calculate the bandwidth ( in bps ) and write it to the files
```

```
puts $rec0 "$now \t [expr ($bw0*8)/($time*1e6)]"
puts $rec1 "$now \t [expr ($bw1*8)/($time*1e6)]"

#Reset the bytes_ values on the traffic sinks
$sink0 set bytes_ 0
$sink1 set bytes_ 0

puts "$now: $totbytes0 $bw0"
#Re-schedule the procedure
$ns after 1.00002  "record"

}
############################################################

################### TRAFFIC GEN + SINK ####################
set tcp_win 4
proc create-udp-sink { node } {
global ns
set sink [new Agent/LossMonitor]
$ns attach-agent $node $sink

return $sink
}

#create a sink for TCP traffic
proc create-sink { node } {
global ns
```

```
        set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $node $sink

return $sink
}


#attch FTP to TCP
proc attach-ftp-traffic { node sink } {
global ns
global tcp_win

        # Create a TRANSPORT agent
        # Attach it to the node
        set source [new Agent/TCP/Reno]
        $source set class_ 2
        $source set window_ $tcp_win
$ns attach-agent $node $source

# Create a traffic agent
        # Set its configuration parameters
set traffic [new Application/FTP]

        # Attach traffic source to the traffic generator
        $traffic attach-agent $source
        #Connect the source and the sink
```

```
$ns connect $source $sink

        return $traffic
}

#attch CBR to UDP
proc attach-cbr-traffic { node sink size rate } {
        global ns

        # Create a TRANSPORT agent
        # Attach it to the node
        set source [new Agent/UDP]
        $ns attach-agent $node $source

        # Create a traffic agent
        # Set its configuration parameters
        set traffic [new Application/Traffic/CBR]
        $traffic set packetSize_ $size
        $traffic set rate_ $rate

        # Attach traffic source to the traffic generator
        $traffic attach-agent $source
        #Connect the source and the sink
        $ns connect $source $sink

        return $traffic
}
```

```
################################################

############## MAIN ###############
set totbytes0 0
set totbytes1 0
set bw0 0
set bw1 0
set conn_start_time 0

set Tot [expr $ActiveFlows]
set OldTot $Tot
set Tot $BgFlows
for { set i 0 } { $i<$Tot } { incr i} {
$ns at [expr $conn_start_time + $i*0.1] "$conn($i) start"
}
set script_start [expr $conn_start_time + $Tot*0.1]
for { set i 0 } { $i<$Tot } { incr i} {
$ns at [expr $script_start + $opt(end) + 1]  "$conn($i) stop"

}
set conn_start_time 10
set tnode $node_(9)
set sink0 [ create-udp-sink $node_(3) ]
set conn0 [ attach-cbr-traffic $node_(9) $sink0 1200 2000K ]

set sink0 [ create-sink $node_(1) ]
set conn0 [ attach-ftp-traffic $node_(2) $sink0 ]
```

```
set sink1 [ create-udp-sink $node_(1) ]
set conn1 [ attach-cbr-traffic $node_(8) $sink1 1200 2000K ]

set script_start $conn_start_time
$ns at [expr $conn_start_time + 0.0] "$ns update-display"
$ns at [expr $conn_start_time + 0.0] "$conn1 start"
$ns at [expr $conn_start_time + 4.0] "$conn1 stop"

$ns at [expr $conn_start_time + $opt(start)] "$conn0 start"
$ns at [expr $conn_start_time +  $opt(end)] "finish"

#Run the simulation
$ns run

############## MAIN - END ###############
```

The following is the emulation header that must be included in a typical *ns-2* script. This is the file emul-header.tcl

```
set val(ll)             LL/LLEmu   ;# link layer type
set val(ifqlen)         50         ;# max packet in ifq
set val(nn)             12         ;# number of mobilenodes
set val(rp)             AODV       ;# routing protocol
set val(x)              320
set val(y)              280
set filename emulation

remove-all-packet-headers
add-packet-header RTP TCP IP ARP LL

set ns [new Simulator]
$ns use-scheduler RealTime

set tracefd [open $filename.tr w]
$ns trace-all $tracefd

set namtrace [open $filename.nam w]
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

create-god $val(nn)

set chan_1_ [new $val(chan)]
```

```
$ns node-config -adhocRouting $val(rp) \
                -llType $val(ll) \
                -macType $val(mac) \
                -ifqType $val(ifq) \
                -ifqLen $val(ifqlen) \
                -antType $val(ant) \
                -propType $val(prop) \
                -phyType $val(netif) \
                -channel $chan_1_ \
                -topoInstance $topo \
                -agentTrace OFF \
                -routerTrace OFF \
                -macTrace OFF \
                -movementTrace OFF \

set PhyNode     $env(NSNODEID)
puts "This Node id = $PhyNode"
for {set i 0} {$i < $val(nn) } {incr i} {
        if { $i == $PhyNode } {
                $ns node-config -phyId $PhyNode
        } else {
                $ns node-config -phyId -1
        }
        set node_($i) [$ns node]
        puts "SCRIPT: node  $i Initialized"
}
```

Appendix C

Fault Injection and Analysis Tool

C.1 Fault Specification Language

The purpose of the fault specification interface is to simplify the way the test scenarios are defined in the fault injection and analysis tool. In order to do that we have defined a Fault Specification Language that provides a rich set of primitives to capture all network fault scenarios. The programming front-end is executed on a central controller node, that parses the user-defined script and installs the relevant data structures on the testbed nodes.

Here we give a brief description of the Fault Specification Language. A user defines multiple fault injection and analysis experiments as scenarios she wants to the protocol to run through, and these are executed as a batch job, making it easier to parse through the results rather than the log files. Each scenario defined in FSL is an unordered set of rules, that are $condition >> action$ pairs. An action is triggered whenever a condition is satisfied. The language provides primitives to define conditions and actions.

```
-----------------------------------------------------------

VAR SeqNoData, SeqNoAck;

FILTER_TABLE
TCP_data_rt1 : (34 2 0x6000),  (36 2 0x4000),
                    (38 4 SeqNoData),  (47 1 0x10 0x10)
TCP_syn : (34 2 0x6000),  (36 2 0x4000),
            (47 1 0x02 0x02)
END

NODE_TABLE
node0  00:46:61:af:fe:23  192.168.1.1
node1  00:23:31:df:af:12  192.168.1.2
END
-----------------------------------------------------------
```

Figure C.1: **Filter Table and Node Table:** Examples of Packet Definitions and Node Definitions. The packet definitions are used to distinguish different packets in TCP protocol. The node definitions comprise the MAC address and the IP-address.

C.1.1 FSL Type Definitions

The goal of FSL is to define different network characteristics. The data types are specific to this purpose. There are three data types, packet definition, node definition, and counter definition.

i. *Packet Definition :* Packet definition is used to specify the packet types that will be monitored by FIAT's fault injection and analysis layer. Each packet is defined as a comma-separated list of tuples, where each tuple consists of the

starting offset of the bytes to match, number of bytes to match, an optional bit mask, and the hex pattern to look for. A packet definition is a logical ANDing of all the tuples. The optional bit mask gives the flexibility for defining a match at bit-level. The list of all packet definitions precedes the scenario definitions and is referred henceforth as the *Filter Table*. Defining packets completely at compile time limits the expressiveness of FSL because often certain fields are generated by the protocol during execution, e.g. sequence number field of a TCP packet. Hence the language provides the freedom to define a variable that is used as the dummy field in a packet definition at compile time. Later at run-time $ASSIGN\_FLTR$ primitive is used to instantiate the dummy field. Figure C.1 shows a Filter Table where TCP_data_rt1 and TCP_ack_rt1 are packet types that get defined at run time.

ii. *Node Definition :* Node definition gives the mapping from hostname to its MAC-address and IP-address. These definitions help in matching source and destination nodes for the packets using the hostnames. The IP-address is needed for initial set up of the data structures across all nodes. This list of node identifiers will be referred to as the *Node Table*. An example Node Table is shown in Figure C.1.

iii. *Counter Definition :* A counter definition in FSL is used to count the events, which is basically the send/receive event of a specific packet type. It can also be used as a local variable on a node. In this case, the counter has to be explicitly controlled by the user-specified instructions.

The syntax for defining a counter which counts an event is,

$$counter{-}name\ :\ packet\_type,\ sender\_node,\ receiver\_node, cntr\_initialization\_node$$

The value of the $cntr\_initialization\_node$ determines whether the counter

ASSIGN_CNTR(counter id)
ENABLE_CNTR(counter id)
DISABLE_CNTR(counter id)
INCR_CNTR(counter id, value)
DECR_CNTR(counter id, value)
RESET_CNTR(counter id)
SET_CURTIME(counter id)
ELAPSED_TIME(counter id)

Table C.1: Counter-Manipulation Primitives and Syntax

is maintained on the sender node (SEND) or the receiver node (RECV). For example, if we have the counter definition as,

$$C1 \ : \ (TCP\_data, \ node1, \ node2, \ SEND)$$

then a counter identified by $C1$ will be maintained at $node1$ and it will count the arrival of packets of type TCP_data.

The syntax for defining a counter which is used as a variable for maintaining states, like the current system time or the value of another counter is,

$$counter - name \ : \ cntr\_initialization\_node$$

In this case, the $counter\_initialization\_node$ has to be mentioned explicitly and it must be defined in the Node Table. For example, the following definition initializes a counter $DIFF$ on $node2$.

$$DIFF \ : \ (node2)$$

DROP(pkt_type, node id, node id, SEND/RECV)
DELAY(pkt_type, node id, node id, SEND/RECV, duration)
REORDER(pkt_type, node id, node id, SEND/RECV, #pkts, order)
DUP(pkt_type, node id, node id, SEND/RECV)
MODIFY(pkt_type, node id, node id, SEND/RECV, pattern)
FAIL(node id)
STOP
FLAG_ERR

Table C.2: Action-Specification Primitives and Syntax

C.1.2 FSL Operators

The language provides a set of primitives for manipulating the counters and to specify the faulty behaviors. The primitives for dealing with the counters are shown in Table C.1 and the action primitives in Table C.2. The ENABLE_CNTR sets the default value for the counter to 0, whereas the ASSIGN_CNTR can be used to initialize the counter to a particular integer value or the value of another counter.

C.1.3 FSL Semantics

The fault and analysis specifications in FSL are part of the scenario description. The scenario definition is identified by some unique id. It comprises the counter definitions and a list of condition and action pairs.

A *term* in FSL is a boolean relation between two counter values, or between a counter value and an integer constant. FSL supports most of the C-like relational operators, viz. $>$, $<$, $\geq$, $\leq$, =, $\neq$. A *condition* is a logical expression of *terms*. The terms can be combined using relational operators, like AND, OR, NOT to represent complex conditions. If the condition is left empty, it is by default set to

be $TRUE$. For any scenario to begin the first condition needs to be empty. The scenario execution can be stopped by an action primitive `STOP`. A scenario that enters a control flow which may never be terminated by a `STOP` primitive, has to be terminated by the system through the fall-back inactivity timeout mechanism, which can be specified by the user as well. Thus if a condition is not evaluated or an action is not executed within a maximum inactivity period, a scenario is terminated. In summary, when a packet of a particular type is encountered, FIAT could trigger a counter update, which in turn could trigger a term computation, leading to a condition evaluation and eventually executing an action that can either be an injected fault or another counter update.

Bibliography

[Acr] Acroname Garcia Robot. http://www.acroname.com/garcia/tutorials/teademo/teademo.html.

[Ad-] Ad-hoc On-demand Distance Vector Routing - Uppsala University. http://user.it.uu.se/ henrikl/aodv/.

[Ami] AmigoBot. http://www.amigobot.com/.

[AOD] Ad hoc On-Demand Distance Vector (AODV) Routing RFC.

[ATSV06] J. Albrecht, C. Tuttle, Alex C. Snoeren, and Amin Vahdat. PlanetLab Application Management using Plush. In *ACM Operating Systems Review*, volume 40, 2006.

[CABM03] D. De Couto, D. Aguayo, J. Bicket, and Robert Morris. High-throughput path metric for multi-hop wireless routing. In *Mobicom*, 2003.

[Cam] Camstream Webcam Application for Linux.

[CDvG+02] Lars Cremean, William Dunbar, David van Gogh, Jason Hickey, Eric Klavins, Jason Meltzer, and Richard M. Murray. The Caltech Multi-Vehicle Wireless Testbed. In *Proc. of Conference on Decision and Control*, 2002.

[Cha02] Benjamin A. Chambers. The Grid Roofnet: A Rooftop Ad Hoc Wireless Network. Technical report, MIT Master's Thesis, Jun 2002.

[CL94] Douglas E. Comer and John C. Lin. Probing TCP Implementations. In *USENIX*, June 1994.

[CLH+05] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[CMS95] Joao Carreira, Henrique Madeira, and Joao Gabriel Silva. Xception: Software Fault Injection and Monitoring in Processor Functional Units. In *5th IFIP International Working Conference on Dependable Computing of Critical Applications (DCCA)*, 1995.

[DG94] D.B.Ingham and G.D.Parrington. Delayline: A Wide-area Network Emulation Tool. In *Computing Systems*, 1994.

[DJM96] Scott Dawson, Farnam Jahanian, and Todd Mitton. ORCHESTRA: A Fault Injection Environment for Distributed Systems. In *26th International Symposium on Fault-Tolerant Computing (FTCS)*, 1996.

[DPZ04] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in Multi-Radio, Multi-hop Wireless Mesh Networks. In *Proc. of Mobicom*, 2004.

[DRK+06] Pradipta De, Ashish Raniwala, Rupa Krishnan, Krishna Tatvarthi, Jatan Modi, Nadeem Ahmed Syed, Srikant Sharma, and Tzi cker Chiueh. MiNT-m: An Autonomous Mobile Wireless Experimentation Platform. In *Proceedings of Mobisys*, 2006.

[DRScC05] Pradipta De, Ashish Raniwala, Srikant Sharma, and Tzi cker Chiueh. MiNT: A Miniaturized Network Testbed for Mobile Wireless Research. In *Proceedings of Infocom*, 2005.

[EARN06] E. Ertin, A. Arora, R. Ramnath, and M. Nesterenko. Kansei: A Testbed for Sensing at Scale. In *Proceedings of 4th Symposium of Information Processing in Sensor Networks (IPSN/SPOTS track)*, 2006.

[EHH+99] Deborah Estrin, Mark Handley, John Heidemann, Steven McCanne, Ya Xu, and Haobo Yu. Network visualization with the VINT network animator nam. Technical Report 99-703b, University of Southern California, March 1999.

[EMU] EMULAB.

[ESS+06] Eric Eide, Leigh Stoller, Tim Stack, Juliana Friere, and Jay Lepreau. Integrated Scientific Workflow Management for the Emulab Netwrok Testbed. In *USENIX Annual Technical Conference*, 2006.

[Fal99] K. Fall. Network Emulation in the Vint/NS Simulator. In *Proceedings of 4th IEEE Symposium on Computers and Communications*, July 1999.

[GEW+03] Deepak Ganesan, Deborah Estrin, Alec Woo, David Culler, B. Krishnamachari, and Stephen Wicker. Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Technical Report UCLA/CSD-TR 02-0013, UCLA Computer Science, 2003.

[GK03] Scott Graham and P. R. Kumar. The Convergence of Control, Communication, and Computation. In *Proceedings of Personal Wireless Communication (PWC)*, 2003.

[Gri99] John Linwood Griffin. Testing Protocol Implementation Robustness. In *29th International Symposium on Fault-Tolerant Computing*, 1999.

[HBE01] J. Heidemann, N. Bulusu, and J. Elson. Effects of Detail in Wireless Network Simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, January 2001.

[HH02] E. Hernandez and A. Helal. RAMON: Rapid-Mobility Network Emulator. In *Proceedings of 27th Annual IEEE LCN*, 2002.

[HJ04] Y. Hu and D. Johnson. Exploiting MAC Layer Information in Higher Layer Protocols in Multihop Wireless Ad Hoc Networks. In *Proceedings of International Conference on Distributed Computing Systems*, Mar 2004.

[HRS95] Seungjae Han, Harold A. Rosenberg, and Kang G. Shin. DOCTOR: An integrated software fault injection environment for distributed systems. In *IEEE International Computer Performance and Dependability Symposium*, 1995.

[HWA] Heterogeneous Wireless Access Network Research Testbed http://systems.cs.colorado.edu/index.php?id=12.

[IEE] IEEE 802.11 Standards. http://standards.ieee.org/getieee802/802.11.html page28.

[JP03] G. Judd and P.Steenkiste. Repeatable and Realistic Wireless Experimentation through Physical Emulation. In *Proceedings of HotNets*, 2003.

[JSF+06] David Johnson, Tim Stack, Russ Fish, Dan Flickinger, Rob Ricci, and Jay Lepreau. Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In *Proceedings of Infocom*, 2006.

[Kar03] Karthikeyan Sundaresan and Vaidyanathan Anantharaman and Hung-Yun Hsieh and Raghupathy Sivakumar. ATP: a reliable transport protocol for ad-hoc networks. In *Proceedings of 4th ACM Mobihoc*, 2003.

[Kar05] Karthikeyan Sundaresan and Vaidyanathan Anantharaman and Hung-Yun Hsieh and Raghupathy Sivakumar. ATP: a reliable transport protocol for ad-hoc networks. In *Proceedings of IEEE Transactions on Mobile Computing*, Dec 2005.

[KCC05] Stuart Kurkowski, Tracy Camp, and Michael Colagrosso. A Visualization and Animation Tool for NS-2 Wireless Simulations: iNSpect . In *International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005.

[KGS06] Sanjit Kaul, Marco Gruteser, and Ivan Seskar. Creating Wireless Multi-hop Topologies on Space-Constrained Indoor Testbeds Through Noise Injection. In *In Proceedings of IEEE Tridentcom*, Mar 2006.

[KNG+04] David Kotz, Calvin Newport, Rober S. Gray, Jason Liu, Yougu Yuan, and Chip Elliot. Experimental Evaluation of Wireless Simulation Assumptions. Technical Report TR2004-507, Dartmouth Computer Science, 2004.

[KR01] J.T. Kaba and D.R. Raichle. Testbed on a Desktop: Strategies and Techniques to Support Multi-hop MANET Routing Protocol Development. In *Proceedings of MobiHoc*, 2001.

[KSK03] R. Karrer, A. Sabharwal, and E. Knightly. Enabling Large-scale Wireless Broadband: The Case for TAPs. In *Proceedings of Hotnets Workshop*, Nov 2003.

[KWK03] Andreas Kopke, Andreas Willig, and Holger Karl. Chaotic Maps as Parsimonious Bit Error Models of Wireless Channels. In *Proceedings of IEEE Infocom*, 2003.

[KZJL03] Almudena Konrad, Ben Y. Zhao, Anthony D. Joseph, and Reiner Ludwig. A Markov-Based Channel Model Algorithm for Wireless Networks. In *Wireless Networks*, volume 9, 2003.

[Lab] National Instrument: LabView. http://www.ni.com/labview/.

[LLN+02] H. Lunndgren, D. Lundberg, J. Nielsen, E. Nordstrom, and C. Tscudin. A Large-scale Testbed for Reproducible Ad Hoc Protocol Evaluations. In *Proceedings of WCNC*, 2002.

[MBJ99] D. Maltz, J. Broch, and D. Johnson. Experiences Designing and Building a Multi-Hop Wireless Ad-Hoc Network Testbed. In *CMU TR99-116*, 1999.

[NEwZA06] Vinayak Naik, Emre Ertin, Hong wei Zhang, and Anish Arora. Wireless Testbed Bonsai. In *International Workshop On Wireless Network Measurement (WINMee)*, 2006.

[NJG02] Michael Neufeld, Ashish Jain, and Dirk Grunwald. Nsclick:: bridging network simulation and deployment. In *Proceedings of the 5th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile systems*, 2002.

[NRN] National Radio Network Research Testbed http://www.ittc.ku.edu/view_project.phtml?id=198.

[NS-] The Network Simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[PACR02] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. In *Proceedings of Hotnets Workshop*, Oct 2002.

[PCB00] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The Cricket Location-Support System. In *Proc. 6th ACM MOBICOM,*, Aug 2000.

[PJL02] Krzsztof Pawlikowski, Hae-Duck Joshua Jeong, and Jong-Suk Ruth Lee. On Credibility of Simulation Studies of Telecommunication Networks. In *IEEE Communications Magazine*, 2002.

[RC05] Ashish Raniwala and Tzicker Chiueh. Architecture and Algorithms for an IEEE 802.11-based Multi-channel Wireless Mesh Network. In *Proc. of IEEE Infocom*, 2005.

[Riz97] Luigi Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. In *ACM computer Communications Review*, 1997.

[RJV94] Shiv Kalyanaraman Raj Jain and Ram Viswanatha. Rate Based Schemes: Mistakes to Avoid. In *ATM Forum/94-0882*, 1994.

[Rooa] Roomba Discovery. http://www.irobot.com/.

[Roob] Roomba Serial Command Interface. http://www.irobot.com/images/consumer/hacker/Roomba_SCI_Spec_Manual.pdf.

[RSO+05] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT Radio Grid Testbed for Evaluation of Next-Generation Wireless Network Protocols. In *To appear at the Wireless Communications and Networking Conference (WCNC'05)*, Mar 2005.

[SBBD03] S. Sanghani, T. X. Brown, S. Bhandare, and S. Doshi. EWANT: The Emulated Wireless Ad Hoc Network Testbed. In *Proceedings of WCNC*, 2003.

[SFKI00] D.T. Stott, Benjamin Floering, Zbigniew Kalbarczyk, and Ravishankar K. Iyer. NFTAPE: A Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors. In *4th IEEE International Computer Performance and Dependability Symposium (IPDS-2K)*, 2000.

[Skl97] Bernard Sklar. Rayleigh Fading Channels in Mobile Digital Communication Systems Part I: Characterization. In *IEEE Communications Magazine*, 1997.

[SL88] Z. Segall and T. Lin. FIAT: Fault-Injection based Automated Testing Environment. In *International Symposium on Fault Tolerant Computing*, 1988.

[Spi] Universal Infrared Remote Control From Any PC. http://www.innotechsystems.com/spitfire6001.htm.

[STP+05] Amit Kumar Saha, Khoa Anh To, Santashil Palchaudhuri, Shu Du, and David B. Johnson. Physical Implementation and Evaluation of Ad Hoc Network Protocols using Unmodified Simulation Models. In *Proceedings of ACM SIGCOMM Asia Workshop*, 2005.

[TBG+05] M. Takai, R. Bagrodia, M. Gerla, B. Daneshrad, M. P. Fitz, M. B. Srivastava, E. M. Belding-Royer, S. V. Krishnamurthy, M. Molle, P. Mohapatra, R. R. Rao, U. Mitra, C. Shen, and J. B. Evans. Scalable Testbed for Next-Generation Wireless Networking Technologies. In *Proc. of IEEE Tridentcom*, Feb 2005.

[TMB01] M. Takai, J. Martin, and R. Bagrodia. Effects of Wireless Physical Layer Modeling in Mobile Ad Hoc Networks. In *Proceedings of MobiHoc*, Oct 2001.

[TS00] Timothy Tsai and Navjot Singh. Reliablity Testing of Applications on Windows NT. In *International Conference on Dependable Systems and Networks (DSN)*, 2000.

[VBV+05] Nitin H. Vaidya, Jennifer Bernhard, V.V. Veeravalli, P.R. Kumar, and R.K. Iyer. Illinois Wireless Wind Tunnel: A Testbed for Experimental Evaluation of Wireless Networks. In *In Proceedings of SIGCOMM Workshops*, 2005.

[Vid] Video for Linux Resources.

[WLG02] Brian White, Jay Lepreau, and Shashi Guruprasad. Lowering the Barrier to Wireless and Mobile Experimentation. In *Proceedings of Hotnets Workshop*, 2002.

[YYA04] Jihwang Yeo, Moustafa Youssef, and Ashok Agrawala. A Framework for Wireless LAN Monitoring and Its Applications. In *Proc. of ACM Workshop on Wireless Security*, Oct 2004.

[ZBG98] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks. In *Workshop on Parallel and Distributed Simulation*, May 1998.

[ZHKS04] Gang Zhou, Tian He, Sudha Krishnamurthy, and John A. Stankovic. Impact of Radio Irregularity on Wireless Sensor Networks. In *Proceedings of Sensys*, 2004.

[ZJB06] Junlan Zhou, Zhengrong Ji, and Rajive Bagrodia. TWINE: A Hybrid Emulation Testbed for Wireless Networks and Applications. In *IEEE Infocom*, 2006.