AN EXTENSIBLE TESTING ENVIRONMENT FOR MOBILE AD-HOC NETWORKS

SHAO NING

NATIONAL UNIVERSITY OF SINGAPORE

2003

AN EXTENSIBLE TESTING ENVIRONMENT FOR MOBILE AD HOC NETWORKS

SHAO NING

(*B.Eng.(Hons.), NUS*)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2003

Acknowledgements

Many individuals have provided advice, encouragement, and support to me in the completion of this thesis. I would like to express gratitude to my supervisor, A/P Dr. A. L. Ananda. He provided guidance and timely help that enabled me to complete my program of study and provided direction to my research. I have learned substantially from his uncompromising emphasis on quality and meaningful research, and the importance of the writing medium in research.

I would also like to thank my fellow researchers, Shao Tao, Zhu Yingjie, and Venkatesh Obanaik for their critical suggestions and advice in various discussions that set the direction and focus for my research. Every discussion with them proved significant in advancing my thesis.

I appreciate the excellent and prompt services provided at Center for Internet Research which have greatly accelerated my experiment and testing.

Finally, I wish to acknowledge my parents being the source of inspiration in my studies.

Contents

List of Figures

List of Tables

List of Abbreviations

AODV	Ad Hoc On-Demand Distance-Vector
COW	Copy-On-Write
CSGR	Cluster Switch Gateway Routing
DSDV	Destination-Sequenced Distance-Vector
DSR	Dynamic Source Routing
LCC	Least Cluster Change
MANET	Mobile Ad Hoc Network
NIC	Network Interface Card
TORA	Temporally Ordered Routing Algorithm
UML	User Mode Linux
WRP	Wireless Routing Protocol
ZRP	Zone Routing Protocol

Summary

In this thesis, we propose a scalable, extensible and high-fidelity emulator for mobile ad hoc network emulation. Generally speaking, there are three approaches towards testing of ad hoc network protocols and applications. They are test-bed approach, simulation approach, and emulation approach. Emulation approach combines the strengths from both test-bed and simulation.

Our emulator is able to emulate at least 6 mobile nodes without actually moving the nodes physically. These emulated mobile nodes are connected by a mobility-aware virtual switch. The virtual switch dynamically enables/disables the connections among emulated mobile nodes. In this way, these emulated mobile nodes move virtually in our emulator.

In the current version, the emulator runs on a single Linux machine. Depending on the computing capacity of the machine, number of emulated mobile nodes varies. The emulator is scalable, because it is able to run on a cluster of Linux machines with some modifications in the virtual switch and its controllers. If each physical machine is capable of emulating n mobile nodes and a cluster has m participating physical machines, it is possible to emulate a total number of $n \times m$ mobile nodes in the cluster.

The emulator is extensible. It is implemented using *Perl*. Extensions and additional features can be added to the emulator easily, since *Perl* is good at fast prototyping. For instance, we can add a module to the virtual switch to introduce packet corruption at a given rate.

Each mobile node runs in an *User Mode Linux* (UML) [1]. From the host point of view, each UML runs as a normal application in its user space; for the applications running in UML, UML is a Linux operating system, which is exactly the same as a real Linux. Unmodified Linux applications can run directly on UML. Therefore, common daemons, which run on Linux machines, can run on our virtual mobile nodes without any modification. For example, real *TCP/IP* stack can run properly inside our virtual mobile nodes. Applications like *ssh*, *telnet* and *ftp* can run in our virtual mobile nodes without knowing that it is running on top of virtual nodes. Hence, the emulation results have high fidelity.

The architecture of the emulator is motivated by *MobiEmu* [2], which was developed by Yongguang Zhang and Wei Li. However, our emulator differs from MobiEmu in many ways, such as the tasks of master/slave controllers, emulation synchronization, utilization of UML, and the use of a virtual switch. We explain these differences in Chapter 3 and Chapter 4.

The design of the virtual switch is motivated by *uml_switch* [1], which was developed by Jeff Dike. Our virtual switch extends the functionality of the original uml_switch by adding the mobility-aware feature. Furthermore, the implementation language is different. Uml_switch was written in *C*, while our virtual switch is written in *Perl*, which is fast and suitable for prototyping.

We believe that our emulator can help MANET protocol and application developers to verify existing implementations and test new designs in various mobile scenarios in a convenient, efficient and safe way.

Abstract

Most current Mobile Ad Hoc Network (MANET) testing environments do not scale very well, due to the fact that each physical node only represents one mobile node. We have built an extensible MANET emulator in order to carry out large-scale experiments with high fidelity. By making use of virtual machine technology (User Mode Linux), each physical host is able to emulate 6 or more virtual mobile nodes, if computing capacity can support. These virtual mobile nodes are interconnected with a mobility-aware virtual switch. New features, like packets corruption, can be added to the virtual switch easily. Without any physical movements, the testing environment can emulate any mobile scenario. Consequently, both high scalability and extensibility can be achieved. Each virtual machine runs as if it is a real mobile node with CPU, hard-disk, and network interface cards. High fidelity can be achieved by running real ad hoc networking applications, such as ad hoc routing daemons, on top of individual virtual mobile node.

Chapter 1

Introduction

This Master's thesis deals with the design and implementation of a Mobile Ad hoc NETwork (*MANET*) Emulator. The emulator makes use of virtual machine technology to achieve high scalability, extensibility and fidelity. In this chapter, we introduce what motivates us to accomplish this work firstly. Then, the objectives and contributions of our work are presented. Finally, the organization of this thesis is introduced.

1.1 Motivation

With the rapid development of wireless communication technology, *mobile networking* has drawn lots of attentions from the research community in recent years. Due to the complex and indeterministic nature of the wireless communication channel, some knowledge and experiences we gathered previously in wired network no longer hold in this new field. At the same time, numerous new challenges are waiting to be tackled. This is a young and promising field with high research value.

There are two modes of operation in mobile networking. In the first mode, the mobile network is made up of mobile hosts and base stations. Base stations are used

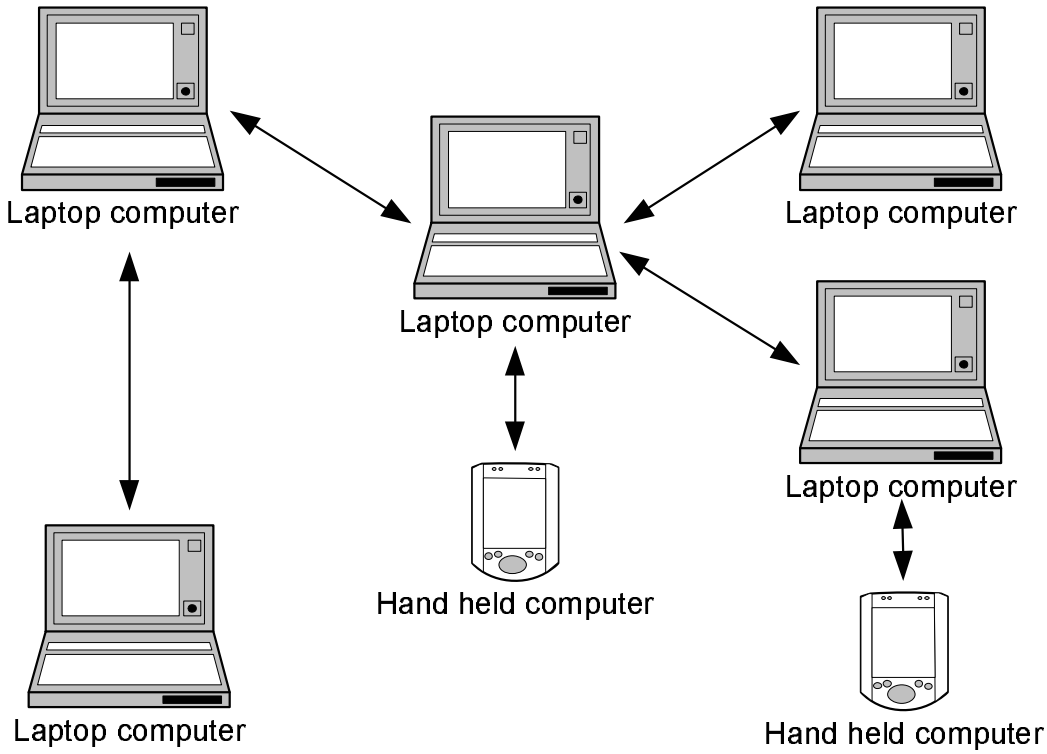


Figure 1.1: An example of an Ad Hoc Network

to forward traffic among mobile hosts or between a mobile host and a wired network. Usually, this is called *infrastructure* mode. In the second mode, the mobile network is set up and maintained dynamically by a group of mobile hosts, without the presence of any communication infrastructure, or base station. This mode is regarded as *Ad Hoc* mode. Figure 1.1 illustrates a simple ad hoc network. Nowadays, the infrastructure mode is the dominating mode used in mobile networks. However, ad hoc mode is gaining popularity because of the requirements from both military and commercial applications.

For example, when there is a need for certain networking applications, people used to think that they must hook their computing devices to a wired network. However,

in practice, sometimes there is no such wired network infrastructure available. This is where ad hoc networking can help. Suppose people in a few neighboring offices want to have a informal discussion. If they don't have wired network access, the best solution is setting up an ad hoc network among the participants of the discussion. It is safe to assume that not all the participants have the knowledge to configure the routing table by hand. Therefore, the formation and maintenance of the ad hoc network must be carried out automatically.

Specialized routing protocols have been developed for ad hoc environment. Well-known ad hoc routing protocols includes *DSDV (Destination-Sequenced Distance-Vector)* [3], *AODV (Ad Hoc On-Demand Distance-Vector)* [4], *DSR (Dynamic Source Routing)* [5] and *TORA (Temporally Ordered Routing Algorithm)* [6].

The evaluation of ad hoc routing protocols is non-trivial. Generally speaking, researchers carry out ad hoc network testing using one of the three approaches:

Test-bed Approach The most intuitive approach is setting up a real test-bed, which consists of a number of mobile hosts. Each mobile host has a wireless adaptor that is configured to ad hoc mode. In this way, ad hoc routing protocols are tested directly on real machines.

Simulation Approach Simulation is the technique of representing the real world by a computer program. Both software and hardware used in simulation approach are not real. In other words, in simulation approach, "ad hoc routing protocols" and "mobile hosts" are only components of a program, which is developed in a specific simulation environment. Many simulation environments coupled with

well-designed protocol modules and configuration utilities have been developed by researchers, such as *ns2* [7], *GloMoSim* [8], and etc.

Emulation Approach Emulation can be considered as the technique of representing hardware A by a computer program and another hardware B. The computer program is named *emulation layer*. With the help of the emulation layer, hardware B will behave exactly the same as hardware A. Therefore,unmodified real protocols can run on the emulator in the same way as they are running on real mobile hosts. In this way, emulation can take the strength of both test-bed and simulation. In recent years, the investigation of emulation approach has become a hot-spot in research community. Lots of papers have been published in this area.

Each of the above mentioned testing approaches has its own advantages and disadvantages:

- Testing results collected from test-bed approach are convincing. This is because that the tests are carried out with real software on real hardware. However, due to the high cost involved with hardware, space constraints, and limited manpower, test-bed approach does not scale well. Normally researchers use only several mobile hosts in their test-bed;

- Simulation approach has the advantage of high scalability and low hardware cost. It is not surprising that in a single test, hundreds of mobile hosts can be easily simulated. However, this approach suffers serious drawback from portability and extensibility. A particular protocol implementation developed on a

simulator is difficult to be ported to real systems. At the same time, simulator usually provides a set of specialized APIs for developers. As a result, developing applications on a simulator and writing extensions for a simulator are both difficult;

- Emulation combines the strength of both test-bed and simulation. Emulation tests with real software components directly. In this way, emulation results usually have high credibility, and the portability is no longer a problem. In a simulation environment, a number of general purpose APIs are available to developers. Therefore, it is easier to develop applications and write extensions in emulation approach than in simulation approach. At the same time, without costly investment in hardware, emulators usually have good scalability. These advantages makes emulation a promising testing approach in the MANET research community.

One of the recent advancements in *virtual machine* technology is used in ad hoc network emulators. Virtual machines are *Virtual Operating Systems* running on top of a real *Host Operating System*. From applications' point of view, a virtual machine is exactly the same as a real OS. Applications can run directly on the virtual machine without being modified or recompiled. System/hardware resource access inside virtual machines are *virtualized*. In other words, applications running on the virtual OS have no direct access to the real underlying hardware. For example, in the virtual machine, hard-disk access is virtualized as the access of a file system in the real host operating system. Thus, a well-controlled protection of the host OS is achieved.

The virtual machine technology used in the implementation of our ad hoc network emulator is "User Mode Linux" [1]. UML is a virtual machine, which can run most applications that the host Linux can. Therefore, using UML, it's possible to run a group of independent virtual machines on one real machine. Each mobile host is emulated by an UML. In this way, on a single machine, multiple mobile hosts can be emulated.

1.2 Thesis Objectives

In this thesis, we present the design and implementation of a mobile ad hoc network emulator, which is scalable, extensible and of high-fidelity. The main objectives of our work are:

- To design and implement an emulator, which is capable of emulating multiple mobile nodes on a single physical machine efficiently by using User Mode Linux;
- To design and conduct systematic experimentations to evaluate the effectiveness of the emulator and measure its performance;
- To generalize an emulator framework from the master/slave control architecture, which is used in the implementation of our emulator. The general emulator framework should be flexible enough to be configured to support both infrastructure mode and ad hoc mode mobile network emulation.

1.3 Contributions

Current mobile ad hoc network emulators use one machine to emulate one mobile host. This limits total number of mobile hosts participating in the ad hoc network emulation. The main contributions of our work include:

- We have designed and implemented an ad hoc network emulator. The emulator is capable of emulating 6 mobile hosts or more on a single physical machine, if computing capacity can support. Virtual machine technology (UML) and mobility-aware virtual switch play important roles in our emulator. At the same time, our emulator is designed to be extensible. Porting it to a computing cluster mainly involves adding an extension to the virtual switch;

- We have designed and conducted experiments using a systematic method. We verify the effectiveness of the emulator by testing reachability and routing correctness. Throughputs under various network scenario are measured to evaluate the performance of the emulator;

- We have generalized the master/slave control architecture of our emulator as a framework for mobile network testing. By providing configurable APIs, new features, like monitoring routing overhead, can be easily added into the framework.

1.4 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 gives a description of the problems faced by MANET researchers, and a literature survey of the technological

backgrounds needed to understand the rest of this thesis. Chapter 3 provides detailed information on the design of the emulator. Chapter 4 explains implementation considerations. Experimentation design and experimental results are summarized and analyzed in Chapter 5. Chapter 5 also discusses future works. Finally, Chapter 6 summarizes the thesis.

Chapter 2

Background

This chapter explains the problems experienced by MANET researchers firstly. Then, a survey of the technological backgrounds needed to understand the rest of this thesis is given.

2.1 Problem Description

To understand MANET protocols and applications well, researchers need to be familiar with some concepts and characteristics of mobile ad hoc network. James A. Freebersyser and Barry Leiner summarized the fundamental issues [9] that MANET researchers should know. These issues include:

Mobile Scenario Mobile scenario generally describes how a set of mobile nodes move around in the terrain and the resulting connectivity. Mobile scenario is influenced by three factors: *network size*, *connectivity* and *network topology*. Usually, network size is regarded as the number of nodes in the network. Alternatively, it can also refer to the area covered by the network. Connectivity refers to a number of issues. It may refer to neighbors within the communication range

of each node. It may also refer to link capacity between two nodes. Network topology is a concept expressed by the mobility of network nodes. As the network nodes move, connectivity changes, as well as the network topology. Ad hoc routing protocols must adapt to these changes;

User traffic The characteristics and types of user generated traffic play an important role in MANET design. Certain patterns of user traffic may be very sensitive to connectivity changes. Some network applications may tolerate packet loss. Knowledge on user traffic can be helpful in both design and experimentation of MANET;

Operational environment Operational environment refers to how radio channels are affected by surroundings, such as terrain or interference;

Energy Normally, network nodes in MANET operate on batteries. If a MANET design does not take energy into consideration, mobile nodes may soon run out batteries. As a result, the mobile network will have less nodes alive, and overall connectivity is hurt seriously;

Regulatory A MANET design must comply with existing regulations on its emitting power, for legal and public health reasons;

Performance metrics Evaluation of MANET design is important. Performance metrics like protocol overhead, packet loss, throughput, delay and so on, must be carefully chosen to suit the user requirements;

Cost High performance-cost ratio will lead to well acceptance by MANET users.

MANET design and evaluation should take performance-cost ratio under consideration.

How well researchers understand the above mentioned issues affects the design of the MANET routing protocols. Before the design is ready for deployment, tests need to be carried out to find whether correctness, efficiency, and other objectives of the MANET routing protocol design have been achieved. Over years, many of MANET testing environments have been developed. They can be classified into three types:

- Testbed
- Simulation
- Emulation

Detailed discussion of the three types of testing environments is given in later part of this chapter.

The problems in the current MANET research challenge us to develop a testing environment, which takes the seven issues mentioned above into serious consideration. It is the advantages of emulation and the advancement of virtual machine technology UML that motivate us to design and implement an emulator with high scalability, extensibility and fidelity.

2.2 Literature Survey

In the section we present a comprehensive literature survey, which covers most important aspects of MANET and its testing environments.

2.2.1 Ad Hoc Network

The history of ad hoc network can be traced back to as early as 1970s. As the pioneering work, PRNet project [10] (initiated in 1972), studied efficient means of sharing a broadcast radio channel, and techniques used to deal with changing connectivity. It also demonstrated *store and forward* routing.

After its demonstration in the early 1980s, PRNet began to draw more and more attentions from DoD of US. At that time, MANET technology was not advanced enough in terms of radio channel design and routing management algorithm. However, the huge potential of MANET had been visioned by the United States military. Across the U.S. Army, Navy and Air Force, new research grants were given to exploit the MANET technology.

MANET technology evolved with the advancements in networking infrastructure and microprocessor technology. However, physical characteristics of MANET technology stay unchanged since the early work of MANET had been accomplished.

Although initiated as a military research project, MANET has been embraced by a number of commercial applications now. These applications [9] [11] include:

- Conferencing
- Home Networking
- Personal Area Networking
- Emergency Services
- Embedded Computing

- Warehouse Inventory Management
- Sensor Network
- Automotive Control & Communication
- Location/Context Based Mobile Services

As the MANET community grows further, more and more products developed for ad hoc networks will be available to meet the requirements of both commercial and military world.

2.2.2 Ad Hoc Routing Protocols

The task of ad hoc routing protocols is to establish route from a source to a destination correctly and efficiently in MANET environment. Therefore, ad hoc routing protocols must be able to adapt to mobility changes. Each mobile node in an ad hoc network behaves as both end host and router. The design of ad hoc routing protocols is quite different from the design of those well-studied routing protocols used in wired networks.

Besides the issues mentioned earlier in this chapter, MANET protocol designers faced additional challenges as pointed out in [9] :

Scalability *Aggregating routing information* available in the Internet keeps routing table small and easy to maintain. However, this property cannot be found in ad hoc networks. As a result, ad hoc routing protocols often suffer from large overhead of maintaining and querying routing table.

Node mobility causes additional scalability problems. Unless the relative position changes stay low, high mobility in an ad hoc network normally leads to frequent route changes. To adapt to these changes in topology, control messages are exchanged in order to maintain connectivity. Though ad hoc routing protocols try their best to keep routing overhead as low as possible, large amount of control messages need to be exchanged. In case the population is sufficiently large, it is possible that routing overhead may consume all bandwidth available [12]. Therefore, good ad hoc routing protocols consume only reasonably small portion of available bandwidth.

Power Limitation Each mobile node in an ad hoc network acts as both end host and router. Although forwarding traffic consumes part of battery power of each mobile node, it makes ad hoc routing possible. Ad hoc routing protocols should keep the routing overhead low, such that the processing of each node is simplified and the battery power can be saved.

Protocol Deployment and Incompatible Standards It may take some time and effort before useful standards in ad hoc routing emerge. Given the design freedom, there are many ad hoc routing protocols. Interoperability among them is a problem. *IETF MANET Working Group* is an organization that promote the standardization of MANET routing protocols.

Wireless Data Error Rate Bandwidth of wireless link is not as high as that of wired link and is very sensitive to the operating environment. At the same time, wireless communication channels also suffer from high bit-error-rate. Many

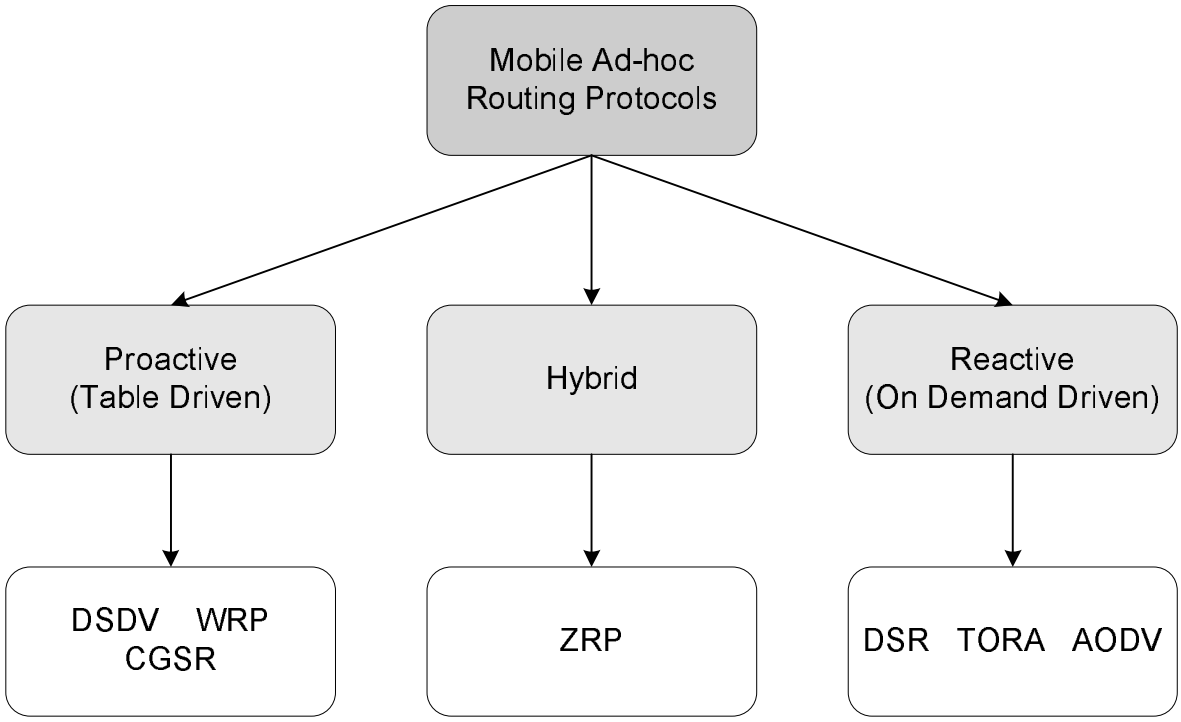


Figure 2.1: Ad Hoc Routing Protocols

researchers are studying TCP performance issues related to packet corruption over wireless link.

Security Exposure Wireless communication is vulnerable to security threats due to the characteristics of its transmitting medium. This makes securing ad hoc routing protocols difficult. Besides, key distribution in encrypted ad hoc routing protocols is also challenging.

Generally speaking ad hoc routing protocols can be classified into three groups, as illustrated in figure 2.1: *proactive*, *reactive* and *hybrid* of the two. In the following paragraphs, we explain the three groups of ad hoc routing protocols one by one.

Proactive Protocols

Proactive protocols are also called *table-driven* routing protocols. Using proactive routing protocols, the routing table of each mobile node contains the routing information for all nodes in the network. Since routes to all possible destinations are already in the routing table, applications usually experience low initial delay. However, proactive protocols introduce extra overhead to keep the routing table of each mobile node up-to-date. Well known proactive protocols includes *Destination Sequenced Distance Vector* (DSDV) [3], *Cluster Switch Gateway Routing* (CSGR) [13] and *Wireless Routing Protocol* (WRP) [14]. The following paragraphs give an overview of these protocols.

- *Destination Sequenced Distance Vector* (DSDV) offers loop-free routing in an ad hoc network. This feature makes DSDV stand out among other distance-vector algorithms, like Bellman-Ford routing algorithms. Each node in the ad hoc network maintains routes to all possible destinations in the network. Connectivity change between any two mobile nodes will trigger routing table updates of all nodes. These updates propagates through the entire network with the help of sequence numbers. Sequence numbers are used in DSDV to reflect the "freshness" of routes. Special techniques are introduced to reduce the amount of control information exchanged in the network. Sequence numbers together with route metrics (hop count) are used as route selection criteria.

- *Cluster Switch Gateway Routing* (CSGR) partitions the network into a group of interconnected clusters. Each cluster has a cluster head. Through clustering,

a framework for code separation (among clusters), channel access, routing, and bandwidth allocation can be realized. Figure 2.2 provides a good illustration on CSGR. *Least Cluster Change* (LCC) is introduced to reduce the frequency of cluster head changes, such that more bandwidth are available to relay packets. When transmitting packet over mobile network via CSGR, the packet is sent to the cluster head of source node firstly. Then, the packet is forwarded from the cluster head to another cluster head via a gateway. The forwarding procedure continues until the packet reaches the cluster head of the destination node. The packet is then delivered to the destination by the last cluster head.

- *Wireless Routing Protocol* (WRP) is a loop-free table-driven routing protocol. In WRP, update messages are sent to neighboring nodes to reflect connectivity changes. What makes WRP different from other protocols is that distance to destination (hop count) and second-to-last hop information are exchanged as part of the routing procedure.

Reactive Protocols

Reactive protocols are sometimes referred as *source-initiated on-demand* routing protocols. Using proactive routing protocols, the routing table of each mobile node contains only the routing information for part of nodes in the network. When a node needs to communicate with another node, whose routing information is not kept in the routing table, it will send *Routing Discovery Packets* to find a route to the destination. Thus, less bandwidth is used to maintain the routing table at each node. However, the latency experienced by applications increases. This is because packets

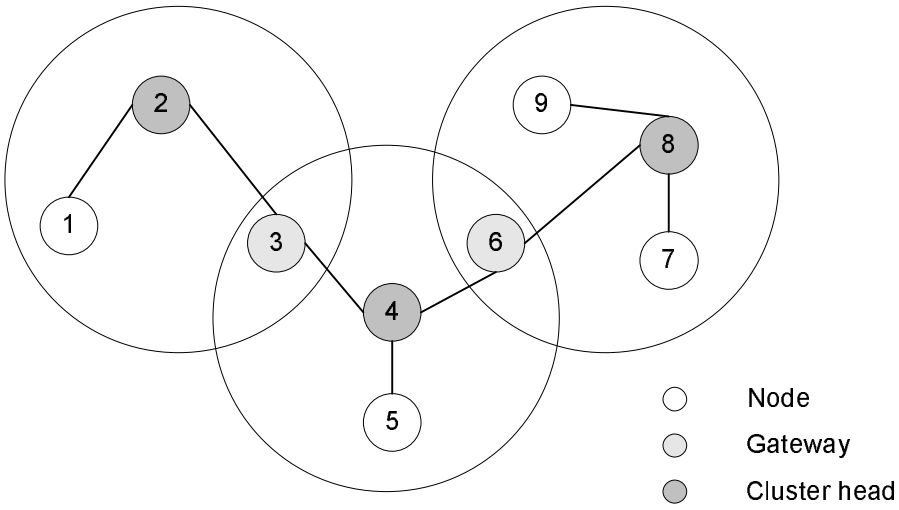


Figure 2.2: Routing in CSGR

cannot be transmitted until all the necessary routing information has been collected. Well studied source-initiated on-demand protocols include: *Ad Hoc On-Demand Distance Vector* (AODV) [15], *Dynamic Source Routing* (DSR) [16], *Temporally Ordered Routing Algorithm* (TORA) [17]. All of the three protocols will be explained briefly in the following paragraphs.

- *Ad Hoc On-Demand Distance Vector* (AODV) is the on-demand version DSDV. Using AODV, only routes that are in used are kept in the routing table at each mobile node. If a route is not used for some time, it will be discarded from the routing table. Three types of messages are exchanged in route discovery and maintenance. They are *Route Request* (RREQ), *Route Reply* (RREP), and *Route Error* (RERR). AODV is capable of supporting unicast, multicast, and broadcast. This feature reduces the overall routing overhead of AODV and simplifies the coding complexity of its implementation. By default, AODV

assumes that communication links are symmetric. AODV does not have special requirement on the physical communication channel.

- *Dynamic Source Routing* (DSR) is a simple loop-free routing protocol for mobile ad hoc networks. It is based on two procedures: *route discovery* and *route maintenance*. In route discovery phrase, a node handle a route request by adding its own "footprint" in the request and forwarding it to its neighbors. A route reply is generated when the route request reaches the destination or an intermediate node knows how to reach the destination. Route maintenance is carried out by route error packets and acknowledges. One interesting feature of DSR is that the data packets includes a ordered list of all nodes that the packet must travel through. The feature enable the nodes, which forward the packet or overhear the packet, to remember the routing information.

- *Temporally Ordered Routing Algorithm* (TORA) is a link reversal routing algorithm. Mobile nodes use relative "height" to form a *Directed Acyclic Graph* (DAG) rooted at the destination. When the DAG route is broken, TORA is able to reconstruct a DAG rooted at the same destination in a single pass. The pass accomplishes three functions: deleting the routes caused by broken link, initializing the search for new routes, and reconstructing the new routes.

Hybrid Protocols

As the name suggested, hybrid protocols are combination of proactive protocols and reactive protocols. Hybrid protocols take the advantages of both proactive and reactive approaches. *Zone Routing Protocol* (ZRP)[18] is one of hybrid protocols.

- *Zone Routing Protocol* (ZRP) utilizes the concept of *zones*. Zones are similar to clusters. Special border nodes are selected dynamically to connect adjacent zones, which is different from hierarchical routing. Zone radius adjusts the size of the zone. A proactive protocol is used inside the zone, and a reactive protocol is applied outside the zone. By adjusting the zone radius dynamically, we can find a value, which gives the best overall utilization of proactive and reactive routing protocols.

2.2.3 Testing Environments

To analyze and evaluate the design of various ad hoc routing protocols, we need testing environments, where real world ad hoc networks can be represented. Over years of investigation, researchers have developed numerous approaches to test ad hoc networks, which can be classified into three categories:

- Test-bed Approach
- Simulation Approach
- Emulation Approach

In the remaining of this chapter, we discuss the advantages and disadvantages of each category in detail.

Test-bed Approach

In test-bed approach, real hardwares and softwares are used in tests. Therefore, the experimental results collected by test-bed approach are convincing. Arbitrary

user traffic can be injected into the test-bed. At the same time, the operational environment used for test is also real. Every possible factors, such as EMI (Electric Magnetic Interference), energy issues, etc., are actively affecting the communication channels. Further more, gathering various statistical information of the test results is not difficult for test-bed approach.

However, time, money and man power involved in this approach is a negative factor. In practice, large and complex mobility scenarios are too difficult to test with test-bed approach. But test-bed approach is still indispensable in research community, as it offers convincing testing results.

David A. Maltz, Josh Broch and David B. Johnson [19] together with other researchers from Carnegie Mellon University managed to build a multi-hop wireless ad hoc network of 8 nodes moving around in a 700m by 300m area. Each node runs Dynamic Source Routing (DSR) protocol. The test-bed is designed to fulfil two objectives. One is to build a platform to investigate the impact of real implementations running on physical machines in a outdoor environment. The other one is to study the behavior of ad hoc protocols under highly mobile scenario. Within 7 months, researchers collects huge amount of valuable data regarding to many aspects of ad hoc networking. However, taking the time and cost into consideration, such test-bed approach is limited to testing the behavior of small scale ad hoc networks, in terms of population and geographic size.

Another example of this approach is the *Ad hoc Protocol Evaluation test-bed* (APE) [20]. APE is able to create verifiable and reproducible experiment environments, using strict "choreography" and "virtual mobility metric" (derived from measured signal

quality). Researchers in [20] reported that up to 37 nodes were involved in the ad hoc network experiments with APE.

Simulation Approach

One distinctive advantage of simulation approach is that it can usually support large, and complex mobility scenario due to the use of parallel computing technology. It reasonably models both the operational environment and the energy condition. Another advantage of simulation is that the cost involved in simulation approach is not high, even when simulating a large scale network.

However, the variety of user traffic that can used in this approach is limited, since simulators often require that ad hoc applications are re-implemented using its own specific APIs. This is an obvious disadvantage. Another drawback of simulation is that designs proved useful by the simulator are often difficult to be ported the real target platforms. According to the research work of David Cavin, Yoav Sasson, and Andre Schiper [21], simulated behaviors have to match the reality closely enough so as to generate meaningful results. Their research work has another two important findings:

- It is vital for simulator to correctly model components like radio propagation, collision and MAC protocols;
- Meaningful and credible results come from realistic simulation parameter settings, e.g. initial conditions. If the values of simulation parameters are gathered from real data, the simulation results should be more credible.

There are many simulators available for ad hoc networking, such as *NS-2* [7],

GloMoSim [8], *OPNET Modeler* [22]. Some of the popular MANET simulators are well explained in [21].

We summarize several popular MANET simulators as following:

- *NS-2* is a discrete event network simulator. It is initially used to simulate wired networks. CMU extended NS-2 to support wireless networks such as MANET and wireless LAN. It simulates layers from physical layer and above. This property makes it a good candidate for testing end-host applications. Powered by C++ for core engine and OTcl for configuration and simulation scripts, it is easy to use without losing much performance;

- *GloMoSim* is a *library-based* discrete event simulator for MANET. It is capable of carry out both sequential and parallel simulations. Because it is developed on top of PARSEC, a C-based parallel simulations language, it is extensible.

- *OPNET Modeler* is able to simulate both wired and wireless networks. It has many built-in node types, such as a wireless node, a firewall, a router, etc. A network is constructed by connecting different nodes. Existing node types can be extended. OPNET Modeler uses state machine to describe a process, such as an algorithm.

MANET simulators will continue to evolve in the future due to its high scalability and low costs.

Emulation Approach

Emulation approach is able to support reasonable large scale mobility scenario. The emulation layer allows most applications to be run directly without any modification on emulators. Some emulators support emulation of operational environment. This approach is also cost-effective. Emulators are usually easy to maintain and extend, since applications can be fit into both real platforms and emulators easily. This outstanding feature makes emulator easy to learn and use. Logically, emulation approach lies between test-bed approach and simulation approach.

Generally speaking, there are two ways towards emulation of ad hoc networks. They are *trace emulation* and *direct emulation*.

In trace emulation, a trace of a network dynamics is obtained through either simulation or real-world experiments. The trace is then used to manage the behaviors of real hosts' networking stack, such as error, out-of-order, delay, etc.

Paper [23] uses the trace-based method to reconstruct the dynamics of a real wireless network. This reconstruction procedure is reproducible and adjustable. The emulation is complete transparent to applications. Therefore, real applications can run on emulators without any modification. The emulation consists of three phases: trace collection, trace analysis, network dynamics reproduction.

In direct emulation as illustrated in figure 2.3, packets generated by real machines are sent to a centralized system, which emulates the behavior of an ad hoc network setting. The system then forwards the packets to their destinations at appropriate times. Emulator mentioned in [24] uses this technique. Its authors claims that the emulator is able to evaluate realistic and repeatable ad hoc network scenarios on real

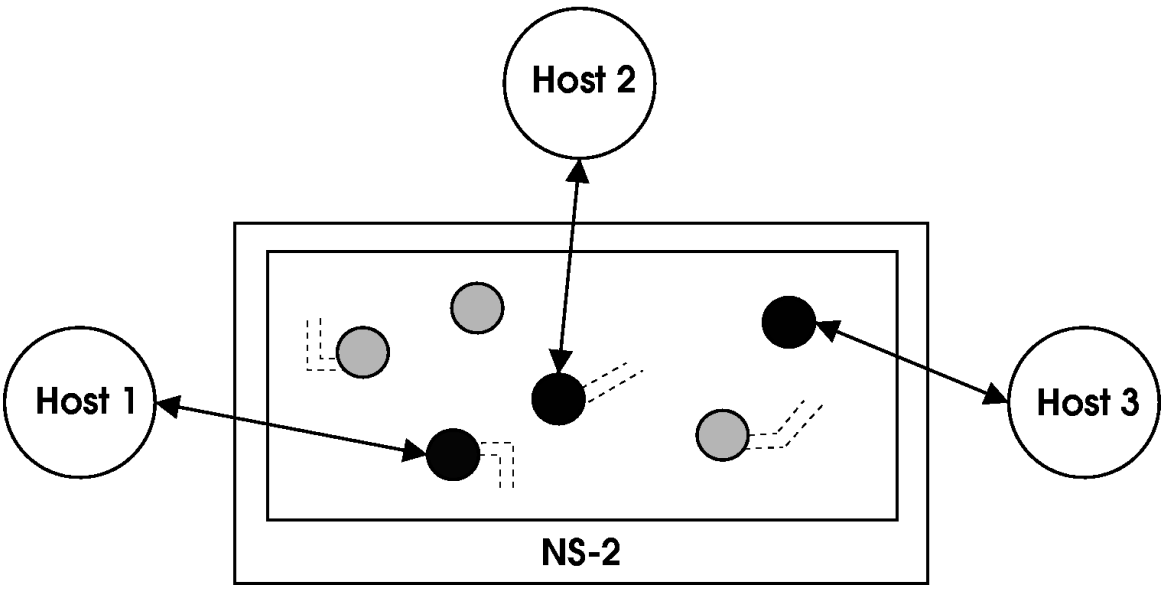


Figure 2.3: Direct emulation

systems.

MobiEmu[2] developed by Yongguang Zhang and Wei Li applies direct emulation differently. The centralized system behaves like a normal switch, which can seen in figure 2.4. Packets are either dropped or forwarded by the kernel of the source node (a real Linux machine) according to predefined rules.

Emulators usually have limited support for operational environment. It is not trivial to emulate energy models in emulators. Carefully designed emulators offers a competitive overall usability over the other two approaches. It is likely that, in the near future, emulation approach will become more and more popular in the MANET research community.

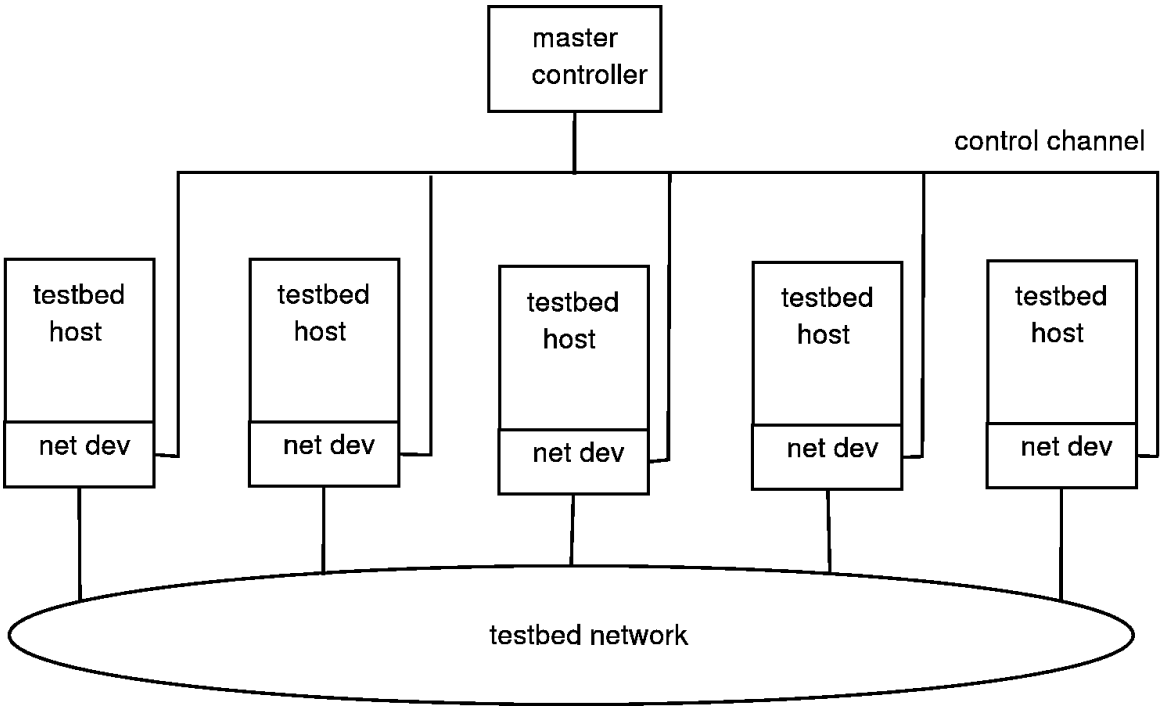


Figure 2.4: Architecture of MobiEmu

Chapter 3

Emulator Design

Most current Mobile Ad-Hoc Network (MANET) testing environments do not scale very well, due to the fact that each physical node only represents one ad-hoc node. With the help of virtual machine technology UML, we have built an extensible MANET emulator to carry out large-scale experiments with high fidelity. This chapter explores the design of our emulator.

3.1 Overall Architecture

We have developed an ad hoc network emulator for testing real ad hoc protocols and applications (*test subject*). Our emulator is able to emulate n mobile ad hoc nodes on a physical Linux machine. In a real ad hoc network environment, connectivity changes are caused by mobile nodes moving in or out of communication range of one another. Among various characteristics of a mobile ad hoc network, we only focus on connectivity changes in this emulator. In our emulator, mobile nodes are connected to a mobility-aware virtual switch. By dynamically enabling/disabling packets forwarding between two ports, the virtual switch emulates the connectivity

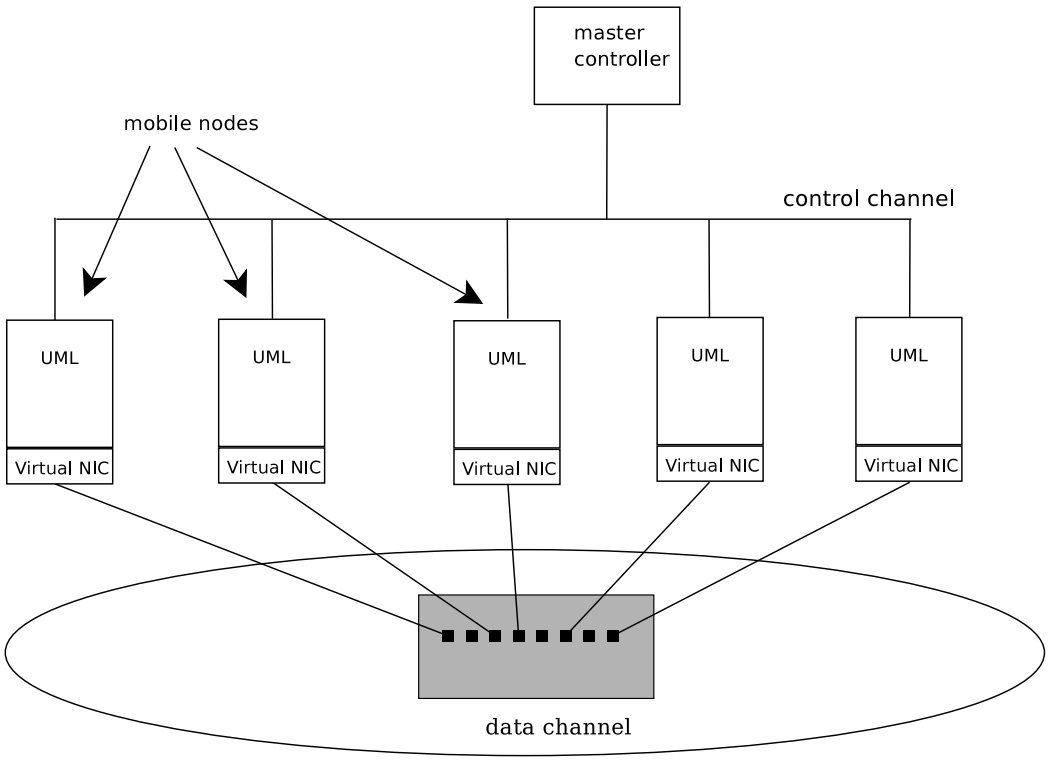


Figure 3.1: Overall Architecture of Our Emulator

changes in a real ad hoc network. In other words, the role of the virtual switch is the wireless communication medium. Extensions, such as physical medium emulation, can be easily added to the simulator.

Our emulator uses the "direct emulation" technique mentioned in chapter 2. The design of our emulator is motivated by *MobilEmu* [2] and [24]. The overall architecture of our emulator and *MobiEmu* are similar. However, they are different in various ways, such as techniques used to control the connectivity, utilization of *virtual OS* (or *virtual machine*) technology, emulation synchronization, etc. We will explain the details of the emulator design step by step.

Figure 3.1 illustrates the overall architecture of our emulator. Each mobile node is

emulated by an *User Mode Linux* (UML). UML is a virtual machine, which can run most software that the host Linux can. There are two virtual NICs in each mobile node: one connected to the control channel, the other connected to the data channel. Therefore, there are two virtual switches. The virtual switch in control channel behaves almost the same as a real switch. It can be viewed as a "software" switch. To manage the mobile connectivity changes, virtual switch in the data channel operates in ad hoc mode. Its basic function is dynamically enabling/disabling packet forwarding between a pair of nodes according to predefined rules. For mobile nodes in this emulator, these connectivity rules mean that they move in or out of the communication range of one another.

Like MobiEmu, our emulator operates in a master/slave control architecture. The master controller runs either on the host machine or another dedicated UML. The slave controller runs inside each UML, which emulates one mobile node. The master controller has three main functions:

- It initializes the emulation environment;
- It controls all slaves and coordinates their actions;
- It sends out connectivity rules to the virtual switch, which operates in ad hoc mode.

The control channel is dedicated to the master/slave control communication. We don't want message exchanges between master controller and slave controller affected by ad hoc topology. That is why we setup the control channel and an associated virtual switch. The data channel is purely for transmitting network traffic in an

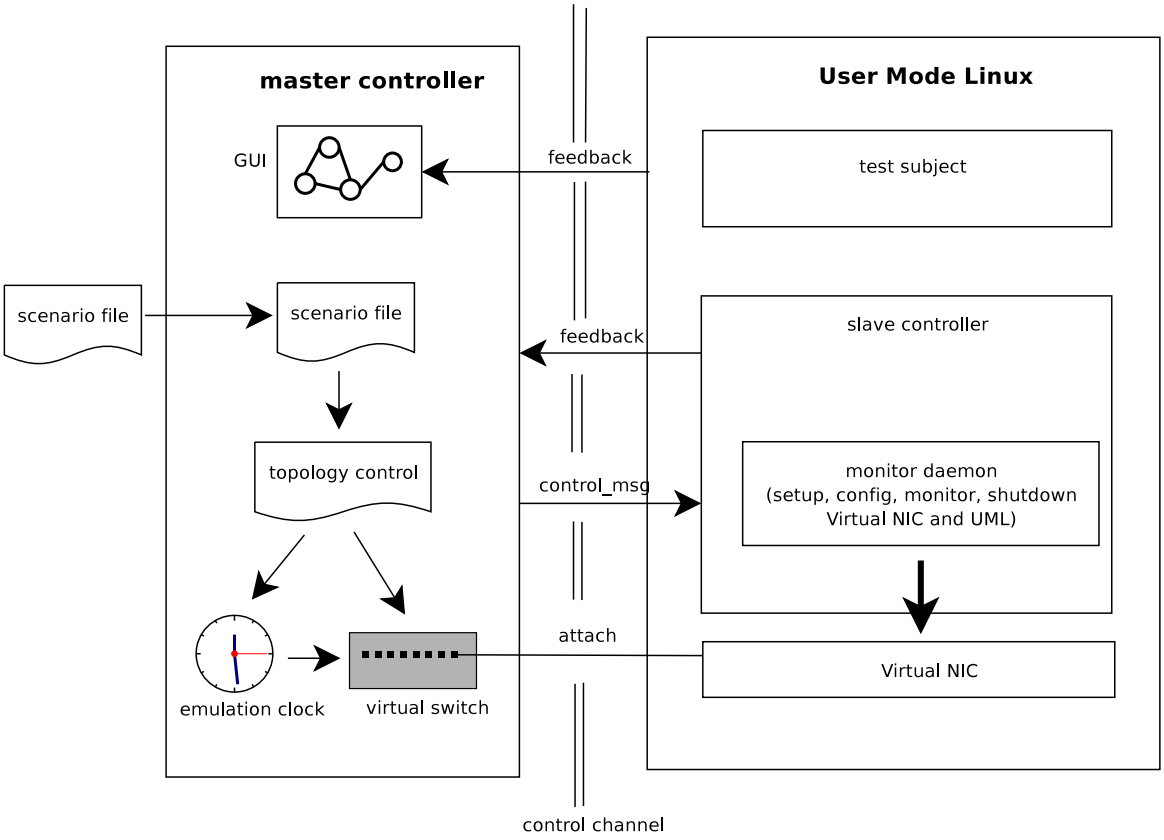


Figure 3.2: Internal Structure of our Emulator

ad hoc network. Figure 3.2 illustrates the internal structure of our ad hoc network emulator.

3.2 Master Controller

The master controller is responsible for controlling and coordinating of various components of the emulator. In current implementation, the master controller is a command line tool without graphical user interface (GUI). In the future, we may implement a GUI to provide better visualization and convenience of control.

Column	1	2	3	4	5	6	7
Values	10.18	6	100.31	95.66	200.78	400.21	2.00

Table 3.1: An example entry in a scenario file

3.2.1 Mobile Scenario

Every emulation needs a scenario file, which describes the mobile scenario of an ad hoc network. The scenario file is loaded and parsed by master controller before the start of emulation. The scenario file is an ordered list of movement and location information sorted according to time. The scenario file format recognized by our emulator contains the following fields:

- Time
- Node_ID
- Current location represented by x and y coordinates
- Destination location represented by x and y coordinates
- Speed of moving

The arrangement of fields in each movement and location information is very similar to MobiEmu design. An example entry of a scenario file is shown in table 3.1.

The meaning of the entry shown in the table is that at time 10.18 (seconds), mobile node 6 is at location (100.31, 95.66) and is moving towards (200.78, 400.21) at the speed of 2.00 meter/second.

To generate such scenario file, we take the output of *setdest* in *NS-2* [7] with CMU wireless extension and convert it to the format recognized by our emulator.

Time	Node_1	Node_2	Connectivity
10.18	6	7	0
10.18	6	8	1

Table 3.2: Connectivity Rules

Mobile scenario file cannot be used directly in our emulator. Our emulator needs the connectivity "rules", which are timed connectivity between two mobile nodes, to run. These connectivity "rules" are extracted from mobile scenario file. Table 3.2 shows us two simple rules.

The meaning of the table is that at time 10.18 (seconds) node 6 and node 7 are too far away for communication. At the same time, node 6 and node 8 are within the communication range of each other.

Unlike MobiEmu, slave controllers do not know any rules. Only master controller maintains all the connectivity rules. When emulation starts, the master controller sends out connectivity rules to the virtual switch at appropriate times. Note that theses rules are actually the changes of connectivities in an ad hoc network. For instance, at time t_0, node 1 and node 2 are "connected". Later on, at time t_1 these two nodes are still able to communicate with each other. Therefore, no rules are sent regarding the connectivity of these two nodes from t_0 till t_1. After a while, at time t_2, these two nodes are too far to "talk" to each other. Between t_1 and t_2, still no rules are sent regarding the connectivity of two nodes. However, at time t_2, a rule indicating that node 1 and node 2 are not within communication range, is sent from master controller to virtual switch.

3.2.2 Emulation Clock and its Synchronization

Each virtual node in our emulator is an UML (User Mode Linux). The "clock" inside UML runs as fast as the host's "clock" [25]. So if a program inside UML "sleeps" for 5 seconds. The 5 seconds is equivalent to 5 seconds in the host machine. The clock of UML is synchronized with the clock of host machine. Because of this feature of UML, we need not worry about time synchronization problem of the emulator. The master controller can operate on both the host machine or on an UML. Consequently, master controller, UML, and slave controller of our emulator virtually share the same "clock", which runs on the host machine.

There is still one issue left, that is the clock of the virtual switch. As we mentioned before, master controller sends out rules to virtual switch. Virtual switch applies rules during packet forwarding. So if master controller sends out rules faster, virtual switch will apply them faster. Therefore, it is the master controller that has the control over how fast rules are applied. Emulation functions, such as "speed up/slow down" and "pause/resume", can be easily added to the master controller. Unlike MobiEmu, emulation clock and its synchronization are easily handled in our emulator. MobiEmu broadcasts/multicasts time-stamp messages to achieve global time synchronization. This approach wastes bandwidth in the "control channel". This drawback of MobiEmu is avoided in our emulator.

3.3 Virtual Switch

The interaction between *virtual switch* and *master controller* is the heart of our emulator. Virtual switch in our emulator has two operating modes: *normal mode* and *ad*

hoc mode. The meanings of these two modes are explained as the following:

Normal Mode In this mode, virtual switch behaves like normal network switches/hubs. It forwards packets from source mobile nodes to destination nodes. If *-hub* option is passed to the virtual switch, every node connected to the virtual switch except the source node will get a copy of the packet, witch is being transmitted.

Ad Hoc Mode In this mode, the virtual switch initially disables all the possible communications among its nodes. As the emulation begins, the master controller will send out rules (connectivity changes) to the virtual switch. Consequently, packets forwarding among mobile nodes are dynamically enabled/disabled. This is the emulation of connectivity changes caused by topology changes in an ad hoc network.

In our emulation, two virtual switches are used. One is configured to normal mode. This virtual switch is used in control channel, since we want this channel to behave like normal Ethernet. The other virtual switch is configured to ad hoc mode. It is placed in data channel to reflect connectivity changes of mobile nodes.

3.4 Virtual Node

Each virtual mobile node in our emulator is an *User Mode Linux* (UML) [1] [25]. UML is a port of Linux kernel to Linux. By using virtualized system call interface, UML is a fully functional Linux kernel. Normally, applications in Linux run directly on the host as shown in figure 3.3. UML provides a wrap for applications as shown in figure 3.4. The wrap is transparent for applications running in it. From the applications' point

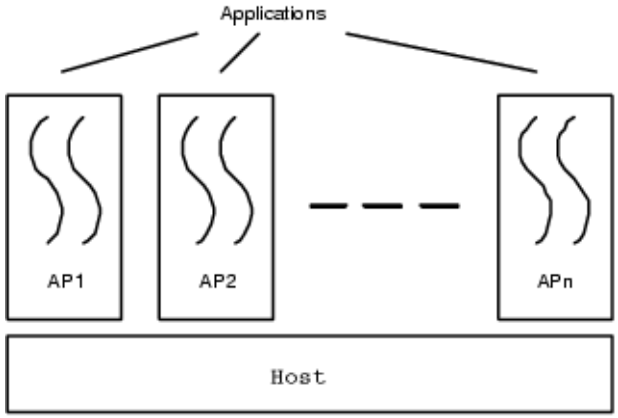


Figure 3.3: Applications running on Linux

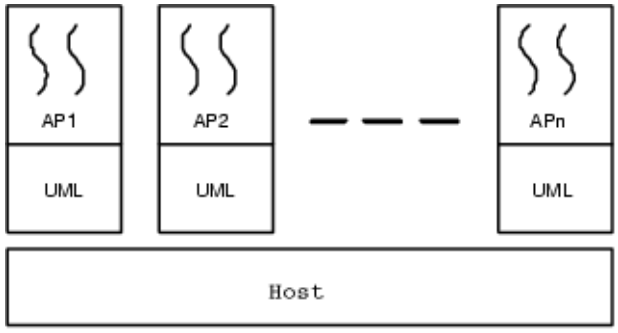


Figure 3.4: Applications running on UML

of view, they are running in Linux. Therefore, applications that run on Linux can run on UML normally without modification. On the other hand, if an application fails, at most this will lead to the failure of its corresponding UML. There will be no damage in underling host. This makes UML ideal for testing new applications safely.

3.4.1 Advantages of UML

As a new virtual machine technology, UML has a number of features. The important UML features that we concern about include:

- UML kernel is a complete Linux system with its own virtual machine (VM), scheduler, file systems, and various I/O devices. UML supports three categories of devices: storage devices, terminal devices, and network devices. UML's support for network devices is very flexible and powerful;
- UML kernel runs as non-privileged user process on the host. But inside UML, normal user on the host can be privileged root;
- UML and its process do not have direct access to host hardware resources. This is a security advantage of UML;
- Most binaries that run in normal Linux kernel can directly run in UML kernel without modification. This makes application porting between Linux and UML easy in most cases;
- UML's clock ticks at the same speed as the host clock. This provides time synchronization between UML and the host;

There are other virtual machine technologies, such as VMWare [26] and Plex86 [27]. But UML suits our needs best due to the features mentioned just now.

3.4.2 Virtual Networking with UML

UML provides a number of virtual networking mechanisms, such as ethertap, multicast, TUN/TAP, a switch daemon, etc. Some of these mechanisms enable a pure UML virtual network; some of them connect UML with the host to form a hybrid virtual network; some mechanisms can support both pure UML virtual network and hybrid virtual network, such as the switch daemon. Our virtual switch may be viewed

as an extension to the switch daemon, which only supports normal switch-like packet forwarding behaviors. New features of switch daemon approach are proposed to solve problems like performance drop in heavy-load network, switching among UMLs across different host, etc. These new features will also benefits our virtual switch.

3.5 Slave Controller

The responsibility of our slave controller is not as heavy as its counterpart in MobiEmu. Generally speaking, its duty includes:

- Initialization
- Providing Services to Master Controller

All master/slave communications take place in control channel.

3.5.1 Initialization

Before UML is booted, the two communication channels have already been set up by the master controller. By parsing the boot arguments of UML, *slave controller* collects enough information to connect UML to both control channel and ad hoc data channel. After the *virtual NICs* have been hooked to the two networks, the slave controller needs to play an additional trick with *virtual switch* in data channel. Even after the virtual NIC has "connected" to a port in the virtual switch, the virtual switch cannot figure out the MAC address of the NIC by itself. Therefore, slave controller needs to send out dummy packet to inform the virtual switch its MAC address. After that, the virtual switch is able to apply MAC filtering, since it has known the identity of all the nodes connecting to its ports.

After the two networks are configured, slave controller sends "ready" signal to the master controller. Only after the master controller learns that the current boot is successful, subsequent boots can take place.

3.5.2 Providing Services to Master Controller

Slave controller is able to respond to simple queries like "ping", which are initiated by master controller to check slave status.

Slave controller also collects statistical information, such as routing table changes, ad hoc routing daemon status, etc. These information can be passively polled by master controller. They also can be sent by slave controller as feedback periodically.

Sometimes slave controller send outs urgent information, like crash of routing daemon. These urgent information is another form of feedback. When sending urgent information, the slave controller is the communication initiator.

As requested by master controller, slave controller needs to shut down the UML after emulation has finished.

3.6 Emulation - The Complete Picture

The process of a complete emulation consists of three phases:

- generating mobile scenarios,as shown in figure 3.5;
- preparing UML root file systems, which contains slave controller, ad hoc routing daemon and all other applications, if necessary, as shown in figure 3.6;
- emulating the mobile scenario, as shown in figure 3.7.

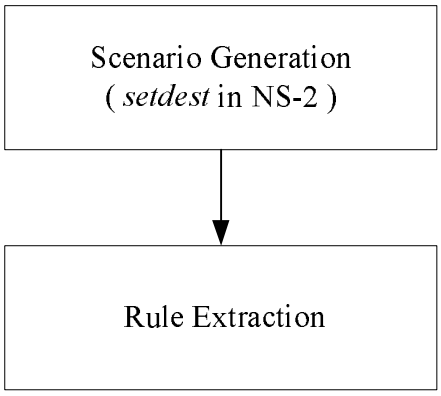


Figure 3.5: Generate mobile scenario

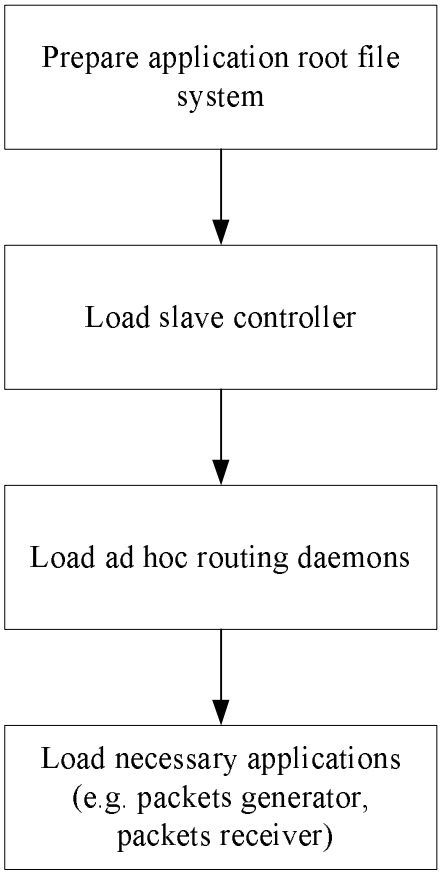


Figure 3.6: Prepare root file systems, slave controller, ad hoc routing daemon, etc.

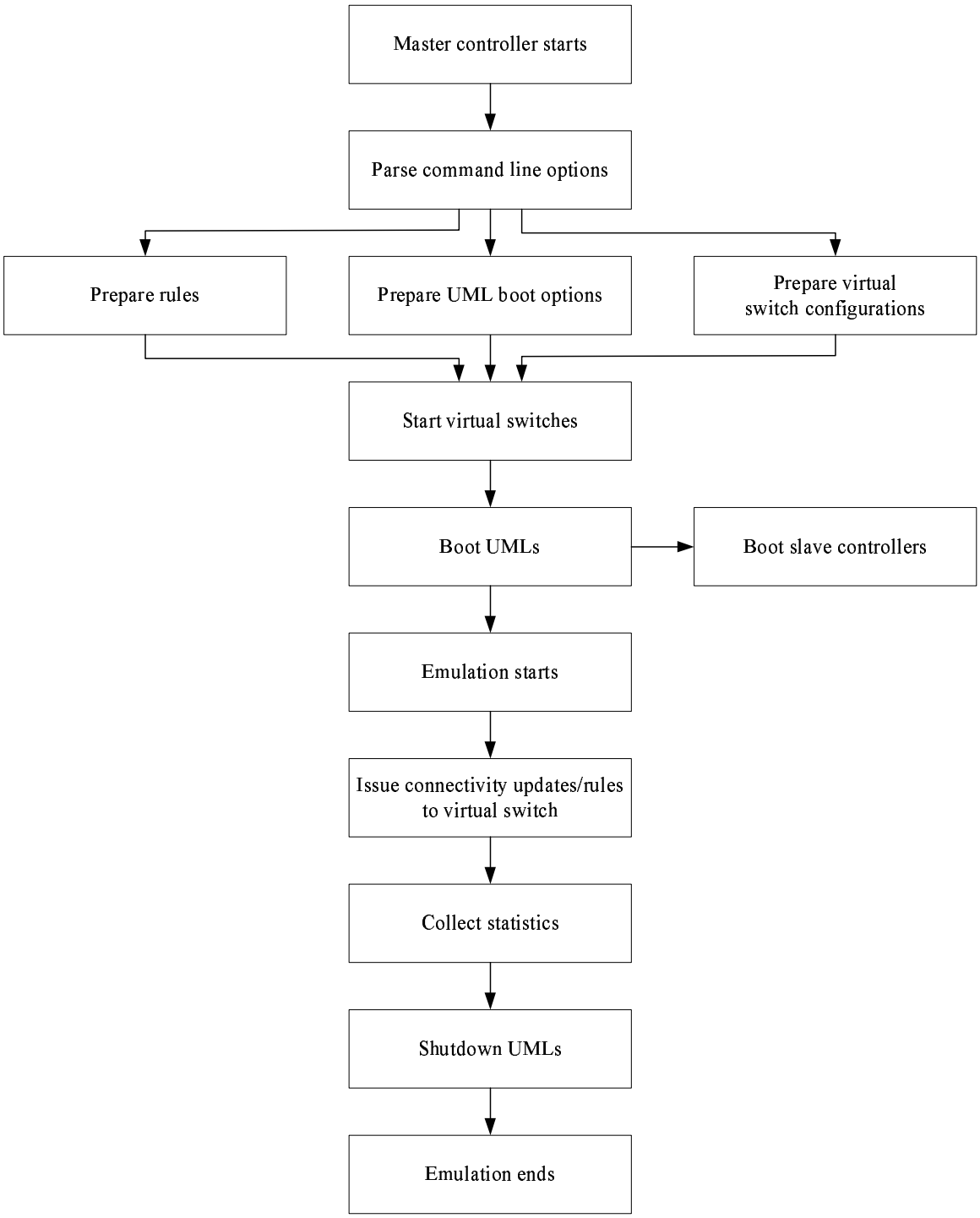


Figure 3.7: The emulation

Chapter 4

Emulator Implementation

In this chapter we discuss the implementation details of our ad hoc network emulator.

4.1 Implementation Languages

We want to develop a mobile network emulator that is easy to use and has good performance. Among all possible choices of programming languages, we select *Perl* as the implementation language of the emulator because of its distinctive features. Firstly, programs written in Perl usually are very short. Consequently, they are easier to debug compared with programs written in other programming languages. Given the large amount of modules and extensions available in Perl, we can concentrate on realizing high level ideas instead of going to details on writing low level supporting libraries. Last but not least, Perl has good performance.

The implementation of ad hoc network emulator involves many considerations, such as writing individual emulator components, creating installers, collecting test results, etc. It is not surprising that Perl is the best choice.

4.2 Virtual Switch

There are many kinds of networking supports in UML. Some networking supports are designed for purely virtual networking, such as *multicast* transport; some are used only for UMLs to communicate with the host, such as *TUN/TAP*; and others can support both types of communication, such as the *daemon* transport. The following are the issues that we take into consideration during networking support selection:

- Multicast transport is not a good candidate for virtual switch because of two reasons. First, the performance of multicast transport degrades fast as the number of participating UMLs increases. Second, there is generally no easy way to emulate link changes using multicast transport.

- TUN/TAP together with the bridging may be able to emulate connectivity changes. However, managing TUN/TAP devices in Linux requires special privilege. This is not desirable in the host, since the privilege granted to TUN/TAP may lead to security problems.

- The daemon transport requires only normal user privilege. It can also support large number of UMLs. These two advantages make daemon transport an ideal solution for virtual switches. This transport has two parts: one resides as virtual NIC in UML, the other is in the switch daemon.

Among all components of our emulator, virtual switch is the most difficult to implement. *switch daemon* is written in *C* in the UML utilities release. But it is

difficult to add extensions to it. Therefore, we need to implement switch daemon in Perl firstly and then add our own extensions, i.e. the ad hoc mode.

When we implemented the TUN/TAP support feature of the virtual switch, we realized that some system call interfaces are not well supported in Perl. To solve this problem, we wrote a Perl module in *C* to support the missing system calls.

During the implementation of the virtual switch, we started with normal mode support. Later on, ad hoc mode was built on top of it. As a result, we saves lots of development time and efforts because of the modular design of the virtual switch.

4.3 Root file system

The root file system in UML is like a hard-disk in a computer. UML cannot boot if the root file system is missing. We decided to select a root file system with a Linux distribution, such as Red Hat, Mandrake or Suse, Debian, etc. This is because that a distribution usually bundled with a lot of useful general purpose utilities and handy system maintenance scripts. Experiments showed that Debian is a suitable Linux distribution, which is very flexible and easy to set up. By connecting UML to the Internet, we can tailor the Debian packages to meet our requirements easily.

There is *copy-on-write* (COW) support in UML. The COW mechanism consists of a read-only shared device and a groups of private read-write devices layering on top of the read-only shared device. The read-only device stores data that are common to all UMLs. The private device stores it's own updates/adjustments with respect to the shared read-only device. Through the use of COW devices, large amount of disk space can be saved, since usually private device is small compared with the shared

device. Another reason to use COW is that the host is able to cache the shared data (in the shared device) in memory. In this way, UMLs spend less time accessing their virtual hard-disks. Our emulator fully utilize this feature of UML to save hard-disk space, which is used to store root file systems, and to boost emulator performance.

4.4 Master Controller

Implementation of master controller is straight forward except one part, which issues connectivity updates to the virtual switch. We require the updates to be sent at specific times. When large number of updates need to be sent out in a short interval, delay or failure may occur. These delay and failure will reduce the accuracy of our emulator. Our solution is forking one dedicated process to deal with connectivity updates. These updates can be very near to one another in term of timing. In this way, the master controller is able to support frequent topology changes, which may happen in real ad hoc networks.

4.5 Slave Controller

Slave controller behaves like a normal daemon, such as an ssh server. A helper script is developed to install slave controller to mobile nodes and to configure it to run in appropriate run level.

The basic role of the slave controller in our emulator is quite different from its counterpart in MobiEmu. In MobiEmu, a slave controller is in charge of dynamically enabling/disabling connections. In our emulator, this function is moved to the virtual switch. We believe that having the virtual switch control the connectivity is intuitive.

On the other hand, the overhead that presents in the control channel of MobiEmu does not present in our emulator. This is because, in our emulator, there is no need to pass time-stamp information in the control channel.

Chapter 5

Experimentations and Results

The characteristics of our emulator (testing environment) are best demonstrated by well designed experiments. In this chapter, we introduce the systematic design of a set of experiments firstly. Then, we present and analyze the experimental results. Possible enhancements are covered at the end of this chapter.

5.1 Experiment Design

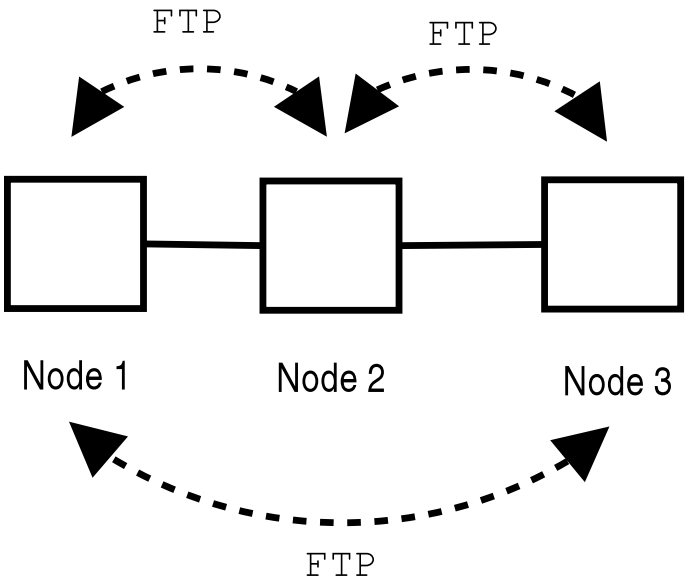
Before going to experiment design details, we would like to highlight the objectives of our experiments. Experiments are carried out to test the *reachability*, *routing correctness* and *throughput* of our emulator. The explanations of these objectives are given in the following paragraphs.

- **Reachability:** Reachability is the ability that packets can reach their intended destinations in our testing environment, if a valid route is set up. Reachability may mean that we can PING mobile node B from mobile node A if there is a valid route in a mobile scenario. Reachability may also mean that we can telnet to mobile node B from mobile node A. Reachability is the foundation of our ad

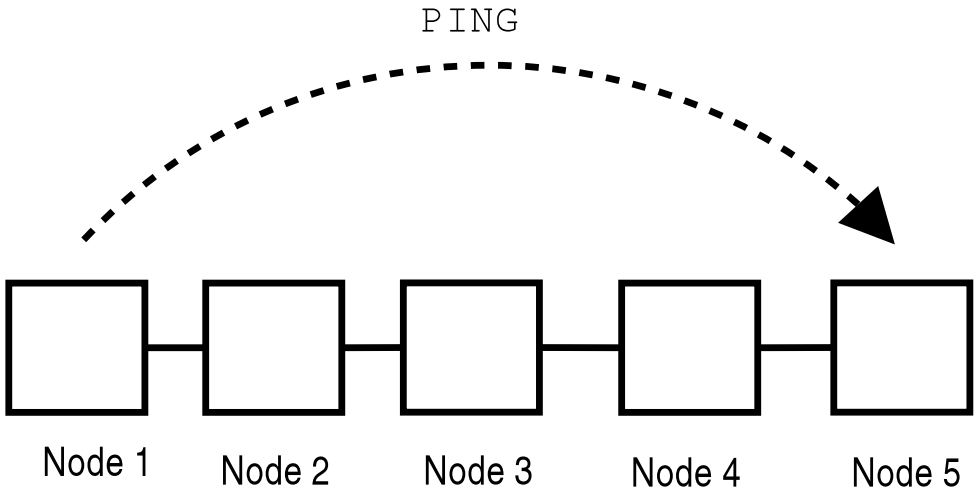
hoc emulation environment.

- **Routing Correctness:** Reachability itself is not sufficient to prove the effectiveness of our emulator. We also need to demonstrate that ad hoc routing protocols can work on our emulator correctly. In other words, routing correctness refers to whether our emulator can help ad hoc routing protocols to establish and maintain correct routing tables. Together with reachability, routing correctness determines the effectiveness of our ad hoc emulator(testing environment).

- **Capability:** Capability of our emulator is determined by the throughput of virtual switch. The throughput indicates how fast virtual switch can forward packets from the source node to the destination node. The throughput is measured in MB/Sec. We do not use the total number of nodes supported by the emulator as the metric of capability, because capability is determined by the density of traffic passing through it for a given period of time, not by the number of participating mobile nodes. The following scenarios explain why we select throughput as capability measurement metric. In scenario (a), which is shown in Figure 5.1, there are three mobile nodes, and all of them are transferring large files through FTP with one another. Respective FTP sessions are represented by dotted arrows. In scenario (b), which is also shown in Figure 5.1, the only network traffic is the "PING" traffic between *node 1* and *node 5*. The "PING" traffic is represented by dotted arrow. As we mentioned in Chapter 3, a packet is forwarded by virtual switch on each hop. Therefore each packet needs to be handled by virtual switch at least once. It is clear that the amount of traffic



(a) an ad hoc network with heavy load



(b) an ad hoc network with light load

Figure 5.1: Throughput Vs Total number of mobile nodes

handled by virtual switch in scenario (a) is larger than in scenario (b), even though there are more mobile nodes in scenario (b). As a result, we choose the throughput of virtual switch as capability measurement metric instead of the total number of nodes supported by the emulator.

5.1.1 Design Considerations

Mobile Node

As we mentioned in Chapter 3, each mobile node is emulated by a virtual node (UML) in our emulator. Typically, a Linux system must have *processor*, *root file system* (normally on hard-disk partition), *memory*, *operating system*, and *applications* to operate properly. A virtual mobile node needs these resources as well.

The processor of a virtual node is abstracted as slices of CPU time of the *host*. To make the complexity of the experiments manageable, the host uses the default Linux CPU scheduling algorithm. In other words, each process in the virtual node consumes CPU resources as a normal unprivileged process on the host.

The memory of a virtual node is mapped to a special file on the host's file system. Each virtual node is allocated 32MB memory, because we want to minimize the memory used by one virtual node in order to support large number of virtual nodes on a single host.

The size of the *root file system* is not minimized, as we make use of the *copy-on-write*(COW) mechanism provided by UML (see Chapter 4 for details) to save disk space. The size of each root file system is 150MB. Suppose that there are n virtual mobile nodes. The total amount of disk space required, which is measured in MB,

can be calculated as in Equation 5.1:

$$total\_size = 150 \times (1 + n \times k) \tag{5.1}$$

where k is the average amount of difference of the n root files systems with respect to the read-only reference 150MB root file system. Normally k is around 0.01. If COW is not used, the total amount of disk space required can be calculated by Equation 5.2:

$$total\_size = 150 \times n \tag{5.2}$$

Comparing Equation 5.1 with Equation 5.2, we can notice the advantage of COW mechanism in terms of disk space usage.

As mentioned in Chapter 4, COW mechanism can help to boost the overall performance of these virtual nodes.

Mobile Scenario

Mobility is one of the fundamental characteristics of MANET. By mobility we are referring to two concepts:

- Speed of a Mobile Node
- Changes in Relative Positions among Mobile Nodes

Suppose we have two mobile nodes: node A and node B. As shown in Figure 5.2, both node A and node B are moving very fast during certain period of time. We sometimes refer these two nodes as high-mobility nodes. High-mobility does not always lead to frequent routing table changes. In Figure 5.2, although both node A and node B are moving fast, they are always (or most of the time) within the "reach"

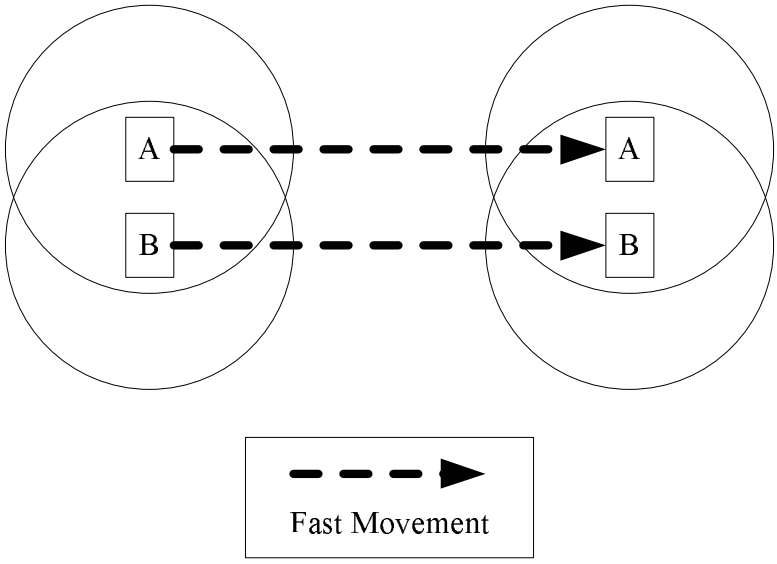


Figure 5.2: Fast moving nodes - infrequent link changes

of each other. Therefore the connection between them seldom changes. So does the route between node A and node B. Situation shown in Figure 5.3 is different. In Figure 5.3, even if both node A and node B move slowly, the connection between them may change frequently.

Network Traffic

Both "connectionless" (UDP) and "connection oriented" (TCP) traffic are tested in our emulator. These two kinds of traffic are dominant traffic in today's Internet.

There are growing research interests in studying TCP performance on wireless networks in general. TCP is originally designed for wired network. Many assumptions made on wired networks no longer stand in wireless networks, especially in ad hoc networks. For instance, in wired networks, repeated retransmissions are treated as a sign of network congestion. If repeated retransmissions occur, the sender's TCP stack

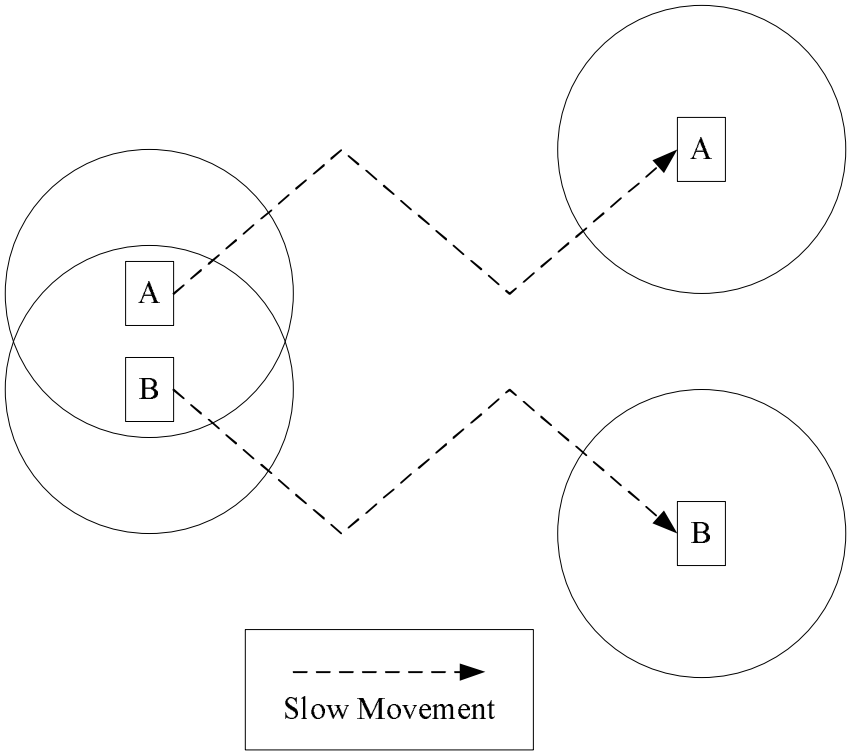


Figure 5.3: Slow moving nodes - frequent link changes

will slow down its transmission to avoid further network congestion. TCP recovers from slow transmission rate when ACKs are received. However, in wireless networks, repeated retransmissions may be due to packet corruption in the transmission media instead of network congestion. In this case, there is no need for the sender to slow down its TCP transmission rate. Many researchers are seeking methods to improve TCP performance in wireless network.

Unlike TCP, UDP is a light weight protocol, which does not has sophisticated methods to adapt to changing network environment. It introduces small amount of overhead in the packets and packets' processing time. Therefore, it is widely used by real-time applications, such as IP telephony and video conferencing. By studying the

behavior of UDP in wireless networks, especially in ad hoc networks, researchers can make their real-time applications work more efficiently.

We make use of *Iperf* [28], a powerful tool to generate TCP and UDP traffic and measure their performance respectively. Iperf operates in *client-server* mode. To measure network characteristics, such as bandwidth and packet lost ratio, between two hosts, one host is configured as Iperf server, and the other is configured as Iperf client. One can specify the transmission time, amount of data to be transited, or a file to be transited in Iperf. With the help from Iperf, we can spend more time on experiment's design and deployment, rather than on utilities, which are used to collect experiment's data and to analyze them.

Ad Hoc Routing Protocols

Routing Protocols are integrated part of MANET. As we mentioned earlier, there are three categories of ad hoc routing protocols: proactive, reactive and hybrid. Even under the same category, protocols behave differently under same network scenario. It would be both challenging and interesting to test real Linux implementations of ad hoc routing protocols on our emulator. Porting ad hoc routing protocols to UML is not as easy as it seems. As we mentioned earlier in chapter 3, UML virtualizes the Linux kernel system call interface. UML kernel is not an exact copy of real Linux kernel. Modifications of the kernel source introduce challenges on porting ad hoc routing protocol implementations, which relay heavily on certain kernel support, from real Linux kernel to UML.

MADHOC is an early attempt of *AODV* implementation. Since it is implemented

completely in user space without any special kernel support (like a routing utility kernel module), we choose it as the AODV implementation in all our experiments. *MADHOC* runs on our emulator without any modification. All participating mobile nodes start an user-space MADHOC routing daemon before the start of test. As an early attempt, the functionalities/features of *MADHOC* are limited. More information is given in experiments details.

There are other *AODV* implementations available, like *KERNEL AODV* [29] and *AODV-UU* [30]. They are more mature, thus offer improved performance and standard compliance (RFC 3561) compared with *MADHOC*. However, both of them need special kernel support. *KERNEL AODV* runs completely in Linux kernel. *AODV-UU* runs in user-space. But it uses a helper program, which runs as a kernel module. We managed to compile both of them. However, currently we are still investigating some run time errors generated by these two AODV implementations.

5.1.2 Measurement Metrics

In order to meet the objectives of the experiments, we make use of the following measurement metrics:

- **Connectivity:** Connectivity is the metric of *reachability*. We use *PING* to test the reachability among mobile nodes in our emulator. Connectivity is measured when there is a valid route between two mobile nodes in our emulator. Here, connectivity has two meanings. The first one is that whether the two nodes can *PING* each other. The other one is the round trip time.

- **Routing Table:** Routing table is the metric of *Routing Correctness*. The most

effective way to measure routing correctness is observing the routing table of participating mobile nodes. Suppose there is a route between mobile node A and mobile node B, and node A is sending packets to node B. Node A's routing table should contain an entry that indicates the valid route to node B.

- **Throughput:** Throughput is the metric of *capability*. There is a module inside virtual switch to measure the throughput of virtual switch. Capability of the emulator is measured in MBits per second.

5.2 Experiment Results and Analysis

Three sets of experiments were carried out to meet our objectives. Each of the following subsections will focus on one set of the experiments.

5.2.1 Reachability

Static Scenario

As shown in Figure 5.4, we developed a series of straight-line-topology scenarios to test reachability of our emulator. In each of the scenarios, there is only one sender node and one receiver node. These two nodes are edge nodes in the topology. In other words, they have only one neighboring node. The rest of the nodes are able to communicate with two neighboring nodes. This series of straight-line-topology scenarios are foundations of all other scenarios. For example, as shown shown in Figure 5.5, at time t, there is a sender (node 1) and three receivers (node 3, node 7 and node 8) in an ad hoc network. This scenario can be treated as the summation of three straight-line-topology scenarios: node 1 to node 3, node 1 to node 7 and node

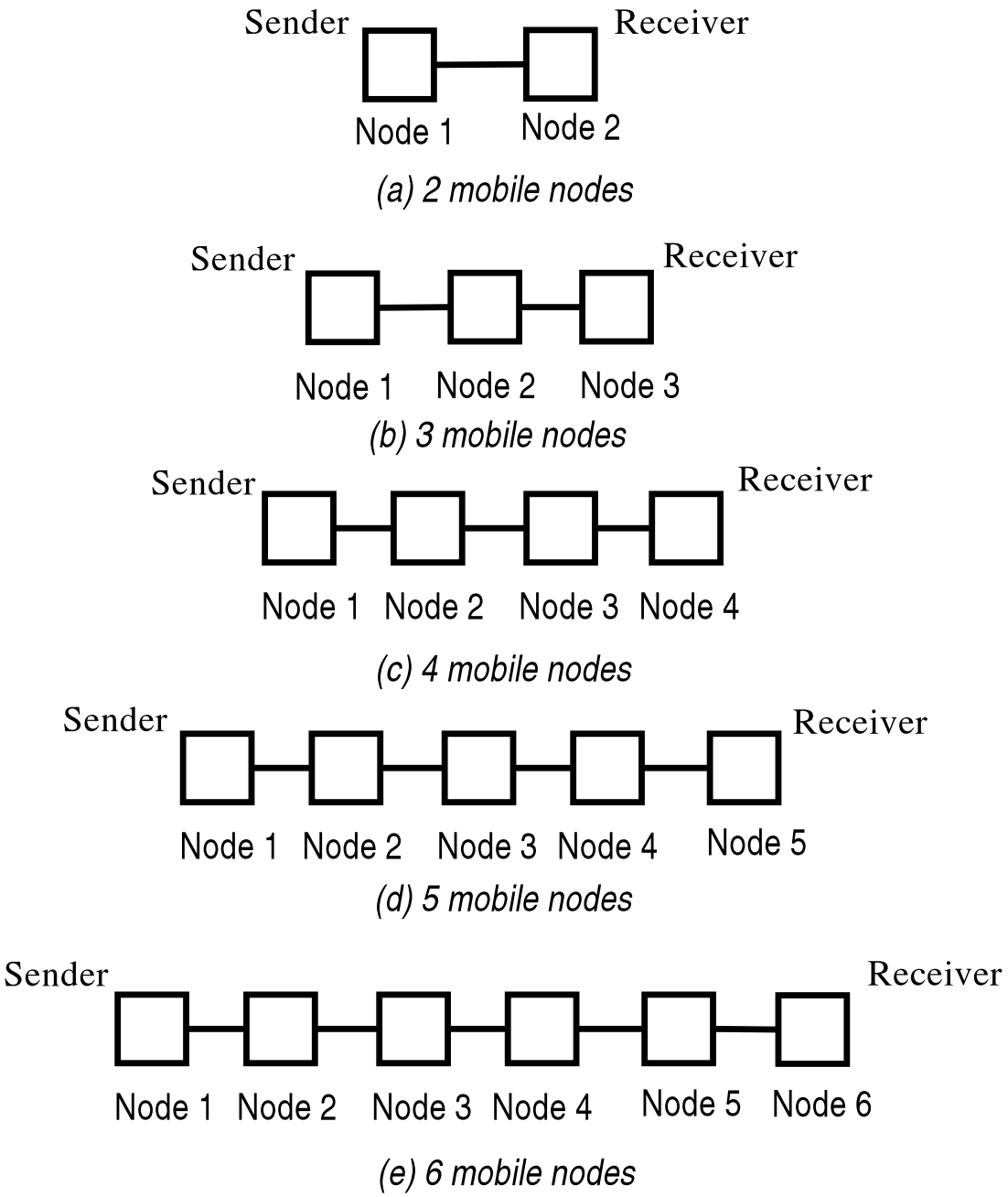


Figure 5.4: Straight-line-topology scenarios

1 to node 8. Once the reachability of the fundamental scenarios are maintained, the reachability of those scenarios, which are derived from the fundamental ones, can also be maintained.

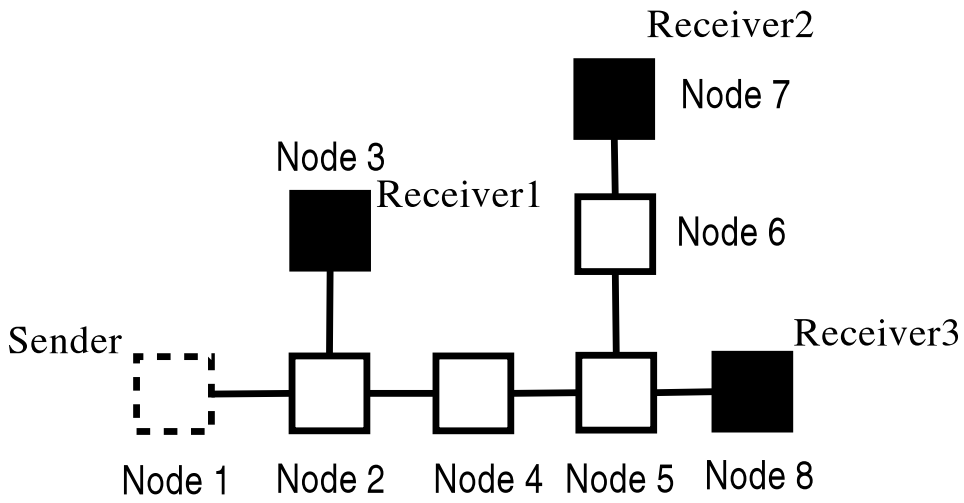


Figure 5.5: A complex ad hoc network scenario

We focus on static network scenarios first. The experiments start with scenario (a) and end at the scenario (e) as shown in Figure 5.4:

- The routing tables of all the mobile nodes are preconfigured to contain the correct routing entries to every other mobile node in the scenario;
- The routing table of each mobile node remains the same in all scenarios;
- There is no physical layer emulation. In other words, no packets are dropped by *virtual switch*;
- For each of the scenarios in Figure 5.4, all mobile nodes stay still for a period of 3 minutes;

- During each experiment period, the sender node keeps sending *PING* packets to the receiver node at the speed of 1 packet/second.

We measured the average *round trip time* (RTT) and the average *packet loss ratio* for each scenario. We modified the experiment setup, so that each mobile node uses *MADHOC* routing daemon to maintain its routing table. Then we reran the same set of experiments with the modified setup. The average *round trip time* and the average *packet loss ratio* were also measured this time. The details of the modified setup are listed below:

- There is no preconfigured routing table this time. *MADHOC* routing daemon is in charge of the maintenance of routing table;
- The routing tables of the mobile nodes are changed by *MADHOC* routing daemon if necessary;
- There is no physical layer emulation. In other words, no packets are dropped by *virtual switch*;
- For each of the scenarios in Figure 5.4, all mobile nodes stay still for a period of 3 minutes;
- During each experiment period, the sender node keeps sending *PING* packets to the receiver node at the speed of 1 packet/second.

We summarize the results obtained from the above two sets of experiments and present them in Figure 5.6 and Figure 5.7.

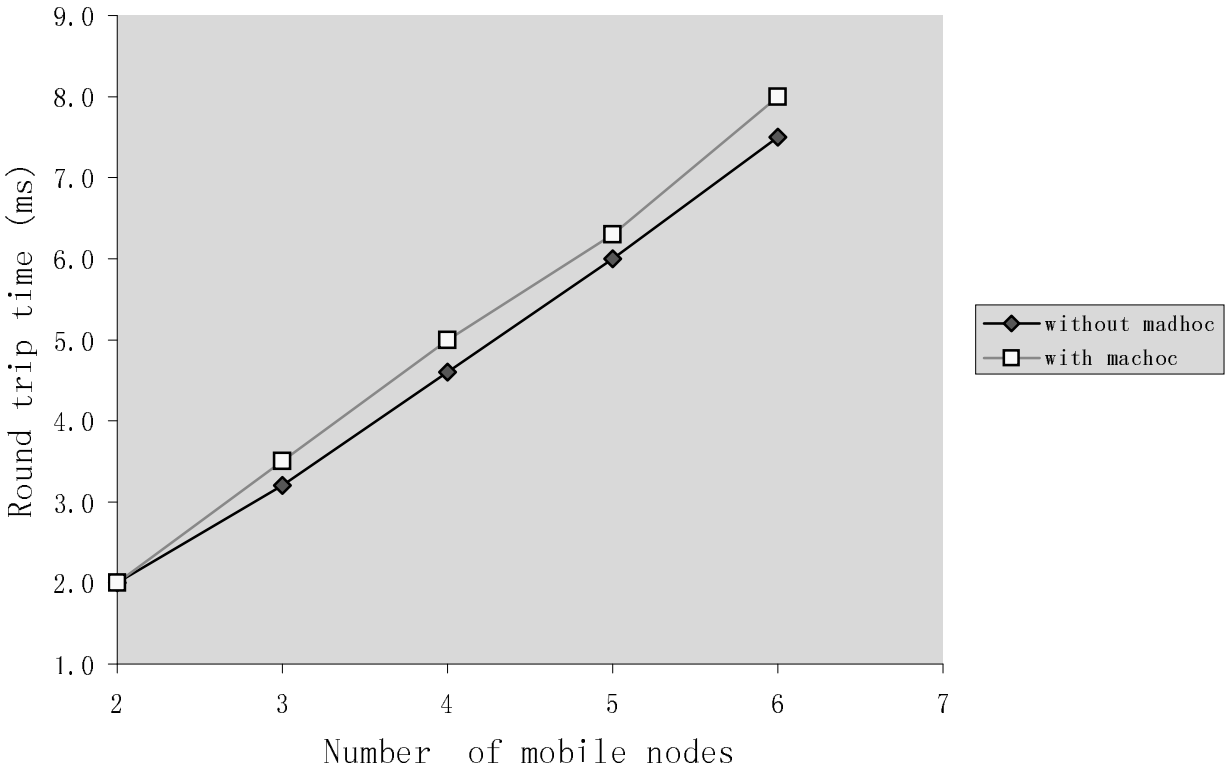


Figure 5.6: Round trip time Vs Number of mobile nodes (static scenario)

From Figure 5.6 we can find that as the number of mobile nodes increases, the *round trip time* also increases. The reason is straightforward. As *straight-line topology* is used in the experiments, more nodes lead to more forwarding of packets by virtual switch and mobile nodes along the routing path. Each "forwarding" introduces some delay. Therefore, adding mobile nodes in the scenario has the effect of increasing *round trip time*. Another interesting finding from Figure 5.6 is that the setup that uses preconfigured routing tables leads to smaller round trip time compared with the one that uses MADHOC routing daemons. According to AODV specification, each mobile node maintains a list of neighboring nodes. How this is achieved is implementation

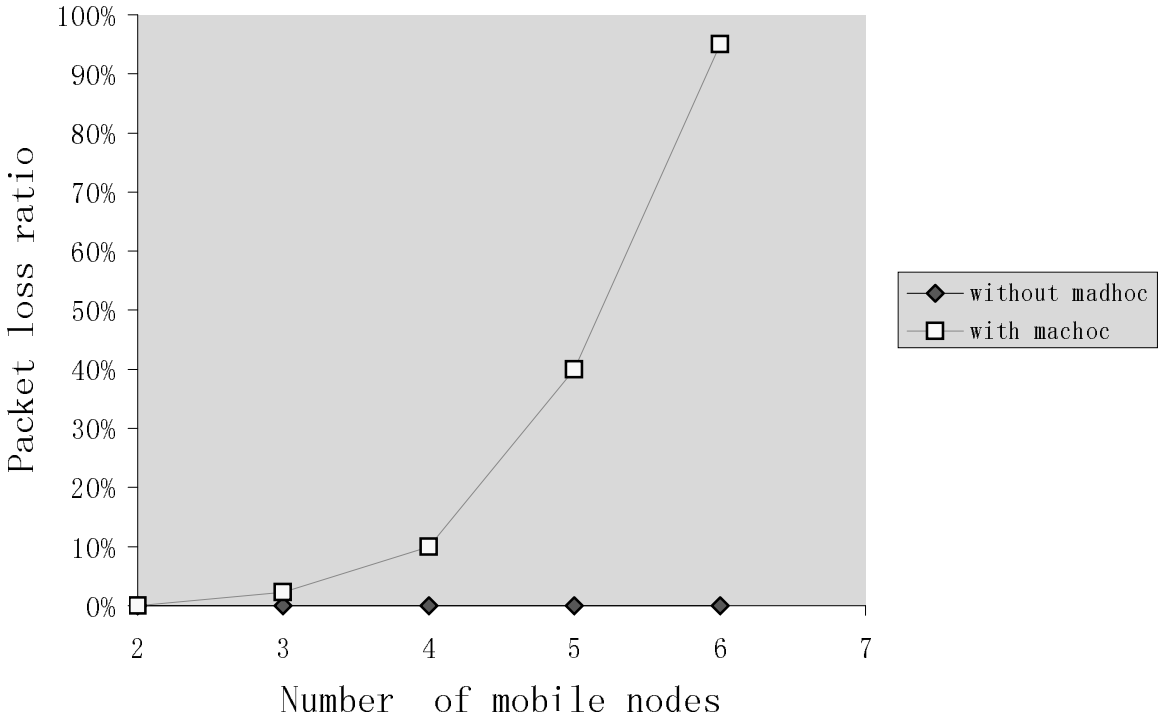


Figure 5.7: Packet loss ratio Vs Number of mobile nodes (static scenario)

dependent. What MADHOC does is that it keeps broadcasting *hello* messages to its neighboring nodes. Once a neighboring node receives the hello message, it will respond with a acknowledgment message to the sender of the hello message. When the sender of hello message receives the acknowledgment, it will add the sender of the acknowledge message to its neighbor list. As the *expire period* of the entries in the neighbor list is short, the *hello-ack* messages need to exchange frequently to keep the entries in the list. Both hello and acknowledgment packets are competing network bandwidth with *PING* packets. It is likely that *PING* packets will frequently wait in the transmission queue of a mobile node. This is the reason why there is a small gap between the lines of the two setups in Figure 5.6.

As shown in Figure 5.7, although the number of mobile nodes increases, *packet loss ratio* of the MADHOC-disabled setup remains zero. All routes in this setup are preconfigured. A mobile node does not perform any route discovery work, thus does not need to drop packets. However, packet loss ratio of the MADHOC-enabled setup increases very fast. This is due to two reasons:

- When there is no entry in the routing table to a particular destination, MADHOC daemon initiates the route discovery procedure. Usually the discovery procedure takes some time to complete. MADHOC daemon does not buffer the packets going to that destination during this period of time. These packets could be removed, as the space in the output queue of the machine is limited. A few AODV implementations have already included packet-buffering to reduce packet loss. The design of MADHOC focuses on simplicity, therefore, it does not implement support for buffering packets, whose destination route is not known in advance;

- The MADHOC implementation of AODV has a serious drawback, i.e. as the hop count between the sender mobile node and the receiver mobile node increases, it becomes harder and slower for the sender mobile node to find a valid path to the receiver mobile node. According to our experiments, when the hop count reaches 5, it is extremely hard for MADHOC to find a valid route. This drawback can cause the growth of packet loss ratio when the number of mobile nodes increases.

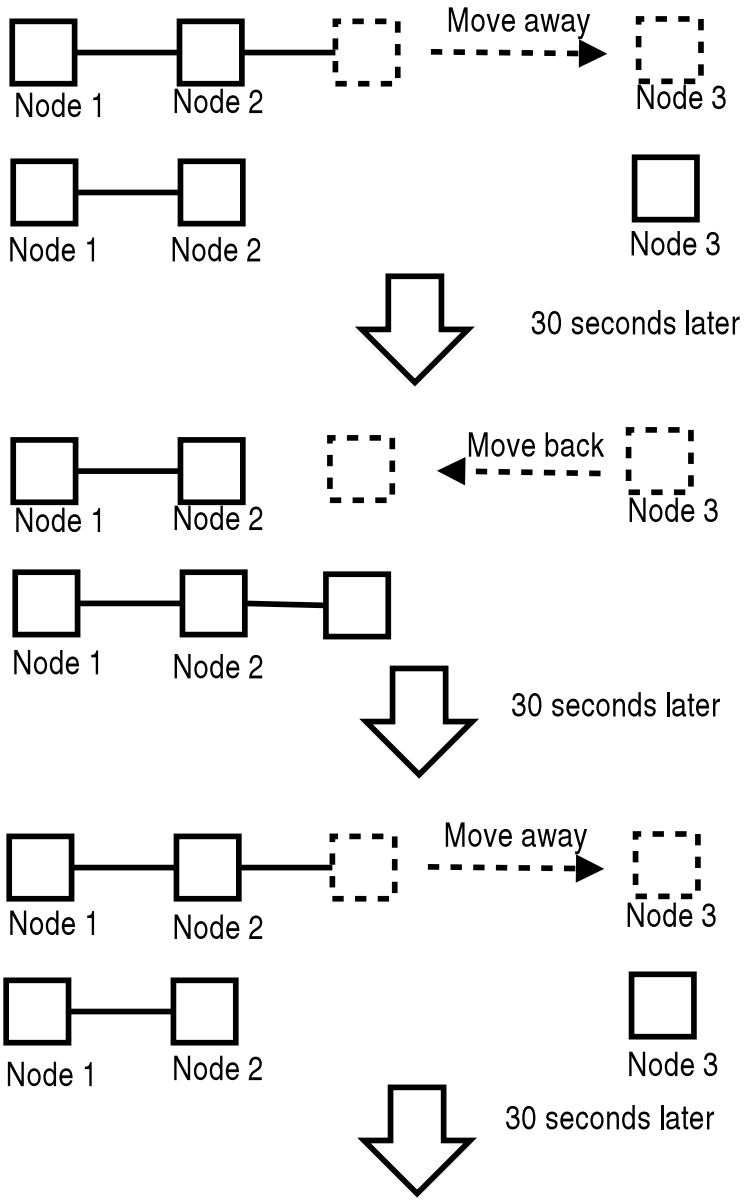


Figure 5.8: Dynamic scenario

Dynamic Scenario

Previous experiments all focus on *static* scenarios. We shall now move our attention to *dynamic* scenarios. In the following few paragraphs, we will discuss a set of

experiments carried out on *dynamic* scenarios.

The design of *dynamic scenarios* is based on *static* scenarios. By adding dynamic behaviors to the receiver mobile node, we can turn a static scenario to a dynamic scenario. Suppose we want to convert scenario b in Figure 5.4 to a dynamic scenario. The receiver mobile node moves away from it neighbor node's communication range for 30 seconds. Then it moves back to its neighbor's communication range for 30 seconds. The mobile node moves back and forth in this way until the end of experiment. Figure 5.8 illustrates this dynamic scenario.

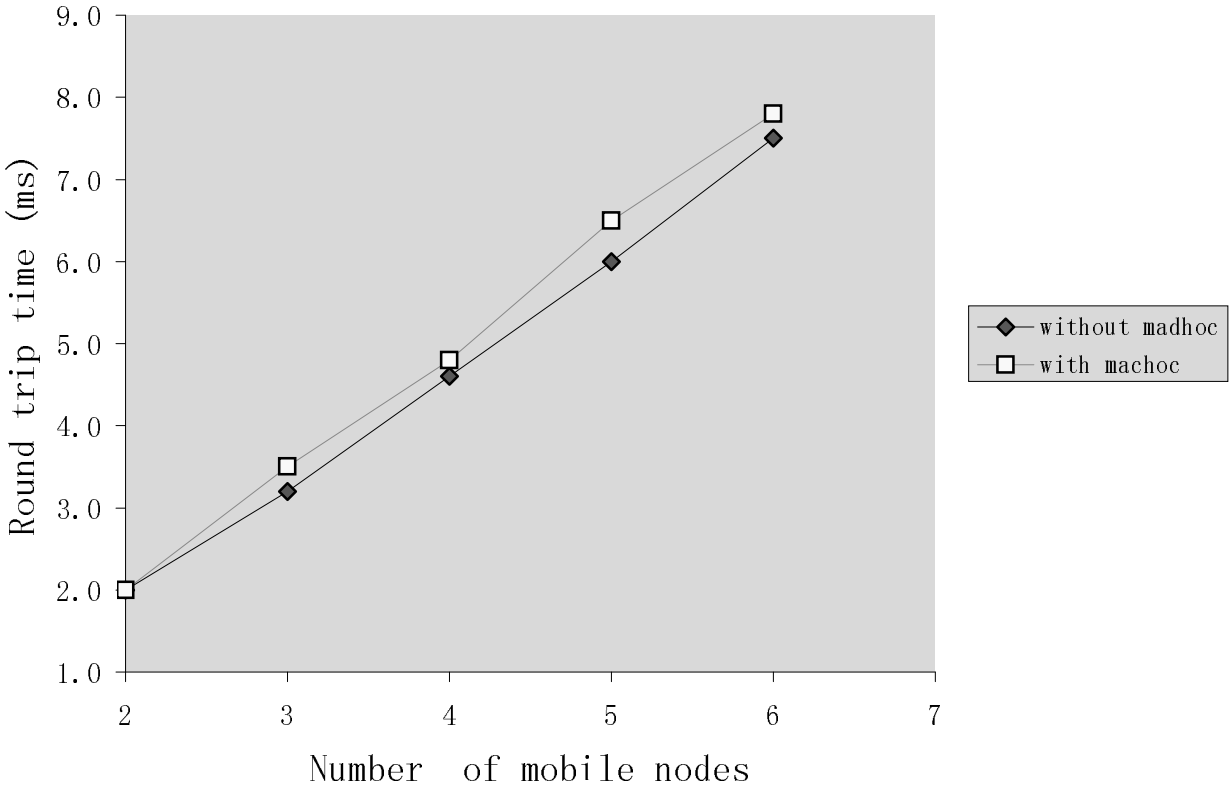


Figure 5.9: Round trip time Vs Number of mobile nodes (Dynamic scenario)

Figure 5.9 and Figure 5.10 summarized the results of experiments on dynamic

scenarios.

Figure 5.9 and Figure 5.6 are very similar. This is not a coincidence. Suppose the receiver mobile node is far away from its neighbor. Then most *PING* packets will be dropped by the receiver's neighbor, as the neighbor cannot find a valid route to the receiver mobile node. Most of these *PING* request will time-out. When we compute the average round trip time, we does not include the time-out values. Therefore, whether the receiver node is able to communicate with its neighbor has minimal impact on the average round trip time.

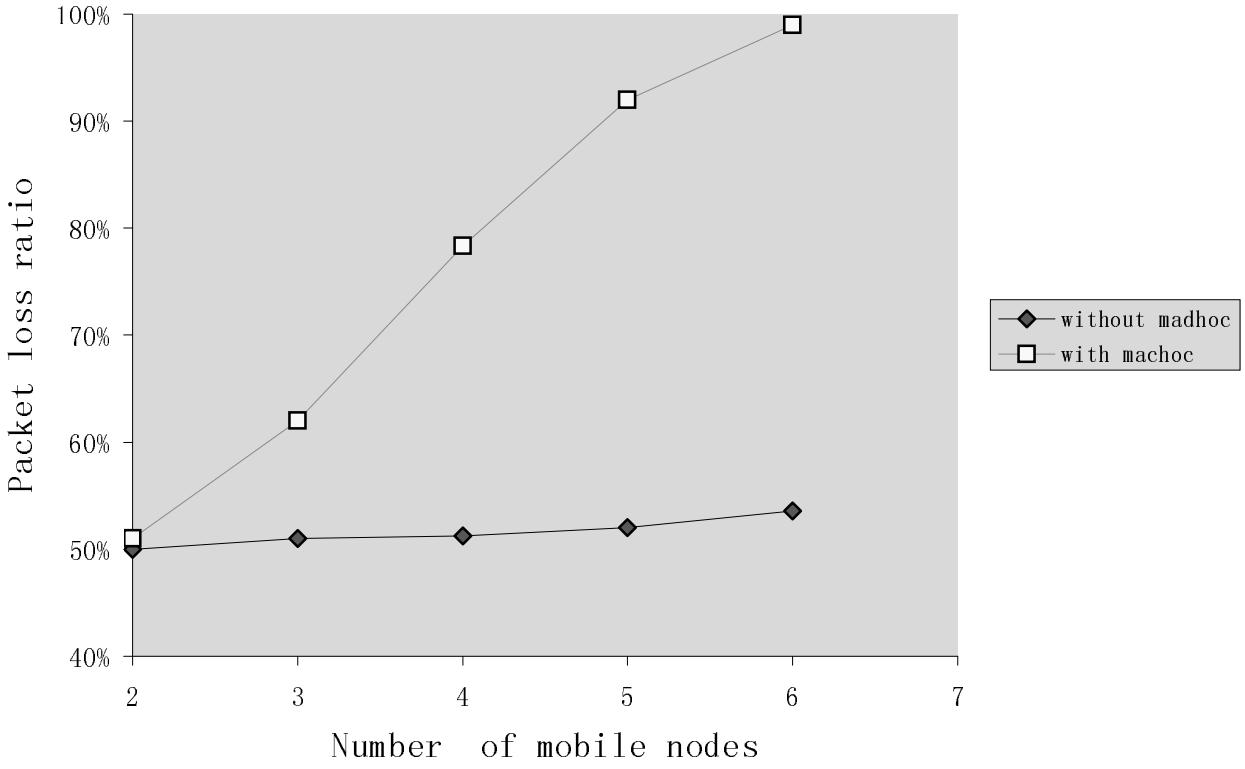


Figure 5.10: Packet loss ratio Vs Number of mobile nodes (Dynamic scenario)

We find that Figure 5.10 is similar to Figure 5.7. This is because that when the

receiver node is able to communicate with its neighbor, the dynamic scenario is the same as the static scenario. When the receiver moves away from its neighbor, all packets are lost. This has the effect of adding 50% to the packet loss ratio.

From Figure 5.10 We can also find that MADHOC implementation of AODV cannot find a valid route if the hop count between two mobile nodes is greater than or equal to five.

Emulation of Packet Corruption

We implement a *packet corruption* module inside *virtual switch* to make the emulation of our emulator more realistic. The packet corruption module has only one parameter, i.e. *packet corruption ratio*. This parameter represents the probability of a packet to be corrupted by virtual switch. We tested the packet corruption module using a dynamic scenario, which is derived from scenario b in Figure 5.4. There are 2 static nodes (node 1 and node 2) and 1 dynamic node (node 3) in this scenario. Node 1 is the sender node, while node 3 is the receiver node. The experiment results are summarized in Figure 5.11.

Summary of Experiments on Reachability

Given the experiments results on *static scenario* ,*dynamic scenario*, and *emulation of packet corruption* we are confident to claim that *reachability* of our emulator is satisfactory.

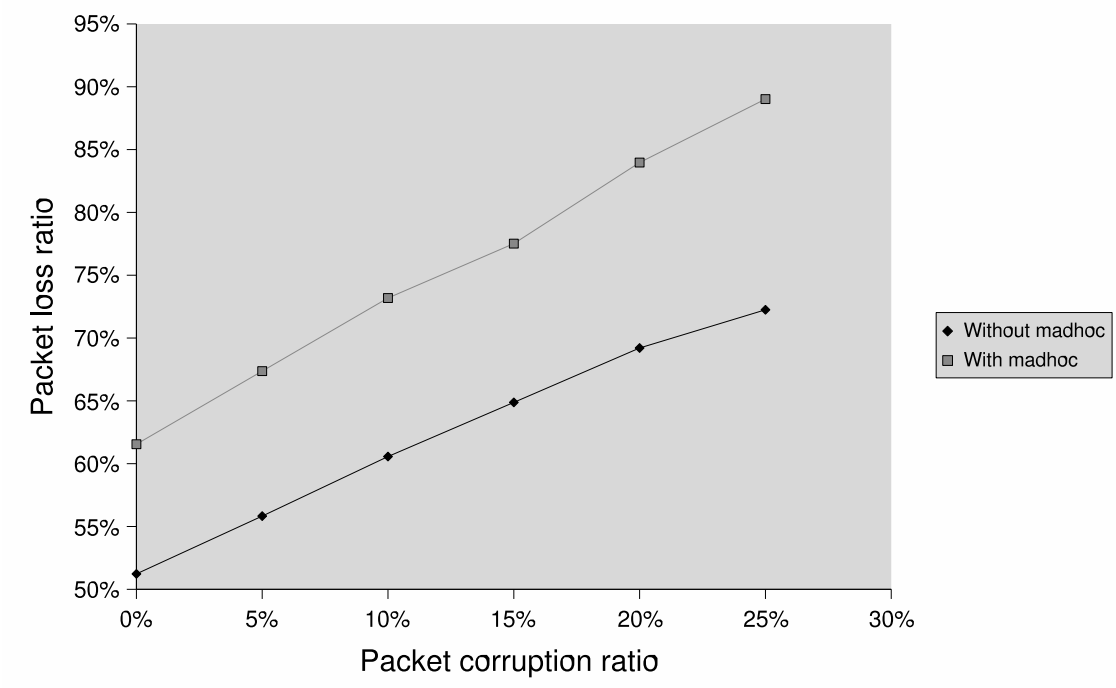


Figure 5.11: Packet loss ratio Vs Packet corruption ratio

5.2.2 Routing Correctness

Based on the experiences learnt from the experiments on reachability, we know that MADHOC cannot find a valid route if the cost of the route is beyond 5. MADHOC only works well with small scale scenarios.

We use scenario b from Figure 5.4 to carry out experiments on routing correctness. Each virtual mobile node in the scenario has two virtual NICs. Each virtual NIC is associated with a class B IP address. As mentioned in Chapter 3, a virtual machine needs to connect to two channels, i.e. control channel and data channel. In our experiment setup, "192.168.0.0" is the network address for control channel, and "192.168.1.0" is the network address for data channel. MADHOC modifies AODV

specification to use "255.255.255.255" as the broadcast address.

Table 5.1 is the routing table of the sender mobile node before there is route discovery. As shown in the table, the routing entry to mobile node 2 ("192.168.1.2") is added right after MADHOC is started. This is because that MADHOC keeps broadcasting *HELLO* messages to its neighbors in order to keep an up-to-date view of its neighbors.

Destination	Gateway	Genmask	Flags	Iface
192.168.1.2	192.168.1.2	255.255.255.255	UGH	eth1
255.255.255.255	*	255.255.255.255	UH	eth1
192.168.1.0	*	255.255.255.0	U	eth1
192.168.0.0	*	255.255.255.0	U	eth0

Table 5.1: Routing table of sender mobile node before route discovery

Route discovery is initiated by the sender mobile node when it tries to send packets to the receiver mobile node. When MADHOC receives *route reply*, it updates the routing table accordingly. Table 5.2 is the routing table of sender mobile node after a valid route to the receiver mobile node is discovered.

Destination	Gateway	Genmask	Flags	Iface
192.168.1.3	192.168.1.2	255.255.255.255	UGH	eth1
192.168.1.2	192.168.1.2	255.255.255.255	UGH	eth1
255.255.255.255	*	255.255.255.255	UH	eth1
192.168.1.0	*	255.255.255.0	U	eth1
192.168.0.0	*	255.255.255.0	U	eth0

Table 5.2: Routing table of sender mobile node after route discovery

Summary of Experiments on Routing Correctness

Besides observing routing table directly, we also use "traceroute" and "ping -R" to study routing behaviors. As expected, both methods produce the same results as Table 5.1 and Table 5.2. We also conduct experiments based on other scenarios from Figure 5.4. Again, correct routing tables are updated and maintained by MADHOC. Based on the experiments results, we are confident that our emulator can help ad hoc routing daemon to setup and maintain correct routing tables.

5.2.3 Capability

With the help of iperf [28], it is easy to measure TCP/UDP throughput on our emulator.

Using scenarios from Figure 5.4, we conducted a series of experiments to measure the capability of our emulator. *Iperf* generates both UDP and TCP traffic, and measures the throughput between the sender and the receiver.

The experiments' results are summarized in Figure 5.12. From the figure we find that throughput of both types of traffic drop as the number of the mobiles nodes increases. As more nodes are placed between the sender node and the receiver node, the contention on the virtual switch increases. This leads to the drop of throughput. TCP is "sensitive" to network congestion, while UDP is not. Therefore, throughput of TCP traffic is less than throughput of UDP traffic.

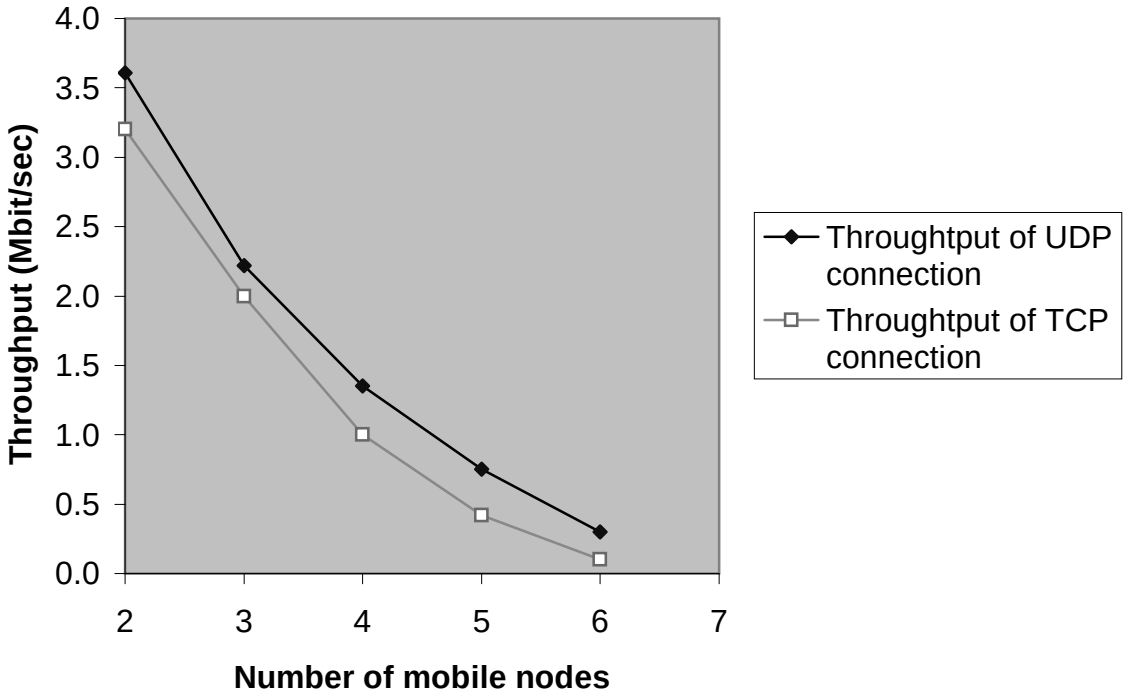


Figure 5.12: Emulator Throughput

5.3 Possible Enhancement

There is still a lot of space left to improve our emulator. We have short term plan and long term plan on the emulator's enhancement.

5.3.1 Short Term Plan

Considering the amount of work involved and resources available, we've planed the following enhancements:

- **GUI:** First of all, we need GUI to help us visualize the mobile topology and adjust various parts of the emulator conveniently and intuitively.
- **Faster Virtual Switch:** We also need to have our virtual switch run faster.

By making virtual switch forward packets faster, we can support heavy traffic effectively. The virtual switch is designed to be a single process polling packets from the ports. At any given point of time, only one packet can be processed. Therefore it is possible that a slow receiving virtual NIC may affect the overall performance of virtual switch. As the number of attached UMLs increases, the polling mechanism is not robust and efficient. Possible solution implementing the virtual switch using multiple processes or threads. In other words, each port is a process or thread. This will bring a significant performance boost for the virtual switch under heavy load.

- **Emulation Management:** Now our emulator does not support advance controls like pause, fast forward, jump, etc. These are useful features helping us control how fast emulation clock ticks. To facilitate debugging of ad hoc routing protocols, we sometimes need to pause the emulation and resume it; sometimes for the same scenario, we just want to have the topology change faster in order to study the behavior of ad hoc routing protocols under certain extreme conditions.

- **Trace Support:** In order to aid ad hoc routing protocols debugging, we can extend slave controller in our emulator to support packet tracing.

5.3.2 Long Term Plan

Based on workload and availability of appropriate hardware support, our long term plan is:

- **Cluster Version of the Emulator:** Given the limited computing power of a single machine, we are not able to emulate large population of mobile nodes. It would be a major enhancement to port the single machine version to a computer cluster, which can potentially support very large scale mobile network scenarios.

- **Physical Layer Emulation:** Now our emulator only emulates link connectivity changes. If we can add physical layer emulation, like radio propagation model, signal strength detection, etc, the accuracy of emulation can be greatly improved. To emulate physical characteristics of wireless medium in real-time, huge amount of data need to be processed in real-time. It is likely that the physical layer emulation is implemented on a high performance Linux cluster.

- **Beyond Master/Slave Mode:** There are situations where the scenario is not known before emulation. The slave controllers may keep updating their movements to the master controller. Consider the following situation. We want to emulate a set of "intelligent" vehicles moving around. These vehicles determine their movements by themselves. At the same time they want to exchange information. A new mechanism of control and collaboration among virtual mobile nodes is needed for this situation.

Chapter 6

Conclusion

We have successfully implemented a scalable, extensible, and high-fidelity mobile ad hoc network emulator. This emulator is built on top of scalable architecture, which can emulate large number of mobile nodes. Mobility-aware virtual switch is one of the key components in this architecture. By dynamically enabling/disabling the traffic flow between any pair of ports attached, the virtual switch can effectively emulate the connectivity changes in mobile ad hoc networks. Currently the virtual switch only emulate connectivity changes and packets corruption. Extensions like packets reordering, packets corruption can be implemented as modules in the virtual switch. Emulated mobile nodes are connected to the virtual switch via virtual NICs. We use UML (User Mode Linux) to emulate mobile node. This ensures high-fidelity of the emulation results. UML is a virtual machine technology, which ports Linux kernel to Linux. It is found that through the use of virtual machine technology and mobility-aware virtual switch, our emulator can effectively emulate the basic behavior of mobile ad hoc network. Master/slave control mechanism is built into our emulator in order to support various emulation management function.

The advantages of our emulator is unique among solutions of mobile ad hoc network testing environment. This research work is the first of its kind to emulate many mobile node in a single machine. Limited by the computing power of single machine, our emulator is not able to support very large population of mobile nodes. However, porting our emulator to a computing cluster is easy. Further research effort should concentrate on making our emulator work in a computing cluster, so that researchers could study implications of large and complex mobile ad hoc scenarios.

Bibliography

[1] User mode linux. http://user-mode-linux.sourceforge.net.

[2] Yongguang Zhang and Wei Li. An integrated environment for testing mobile ad-hoc networks. In *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 104–111, 2002.

[3] Charles Perkins and Pravin Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[4] C. Perkins. Ad-hoc on-demand distance vector routing. In *MILCOM '97 panel on Ad Hoc Networks*, November 1997.

[5] D. Johnson. Routing in ad hoc networks of mobile hosts. In *Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA, U.S., 1994.

[6] V. D. Park and M. Scott Corson. Temporally-ordered routing algorithm version 1: Functional specification, internet-draft, draft-ietfmanet-tora-spec-01.txt, 1998.

[7] The network simulator: Ns-2. http://www.isi.edu/nsnam/ns.

[8] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. Glomosim: A library for parallel simulation of large-scale wireless networks. In *Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.

[9] Charles E. Perkins. *Ad Hoc Networking*, pages 8–14,15–20,31–33,35. Addison-Wesley Pub Co, 1st edition, December 2000.

[10] R. Kahn et al. Advances in packet radio technology. In *Proceedings of the IEEE*, pages 1468–1496. IEEE, November 1978.

[11] C. K. Toh. *Ad Hoc Mobile Wireless Networks: Protocols and Systems*, chapter 15, pages 266–276. Prentice Hall, 1st edition, December 2001.

[12] S. Corson, J. Macker, and S. Batsell. Architectural considerations for mobile mesh networking. In *Proceedings of the IEEE Military Communications Conference (MILCOM '96)*, October 1996.

[13] C. Chiang, H. Wu, W. Liu, and M. Gerlaw. Routing in clustered multihop, mobile wireless networks with fading channel. In *The IEEE Singapore International Conference on Networks*, pages 197–211, 1997.

[14] S. Murthy and J.J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *ACM Mobile Networks and Applications Journal, Special issue on Routing in Mobile Communication Networks*, 1(2), 1996.

[15] C. Perkins. Ad hoc on demand distance vector (aodv) routing, November 1997. Internet Draft, draft-ietf-manet-aodv-00.txt.

[16] David B Johnson and David A Maltz. Dynamic source routing in ad hoc wireless networks. In Imielinski and Korth, editors, *Mobile Computing*, volume 353. Kluwer Academic Publishers, 1996.

[17] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM (3)*, pages 1405–1413, 1997.

[18] Z. J. Haas and M.R. Pearlman. The zone routing protocol (zrp) for ad-hoc networks, December 1997. IETF MANET, lnternet Draft.

[19] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Mobile Computing and Networking*, pages 85–97, 1998.

[20] Henrik Lundgren, David Lundberg, Johan Nielsen, Erik Nordström, and Christian Tschudin. A large-scale testbed for reproducible ad hoc protocol evaluations. In *3rd annual IEEE Wireless Communications and Networking Conference (WCNC 2002)*, pages 412–418. IEEE, March 2002.

[21] David Cavin, Yoav Sasson, and André Schiper. On the accuracy of manet simulators. In *Proceedings of the Workshop on Principles of Mobile Computing (POMC'02)*, pages 38–43. ACM, October 2002.

[22] Opnet technologies. opnet modeler. http://www.opnet.com/products/modeler/home.html.

[23] Brian Noble, M. Satyanarayanan, Giao Thanh Nguyen, and Randy H. Katz. Trace-based mobile network emulation. In *SIGCOMM*, pages 51–61, 1997.

[24] Qifa Ke, David Maltz, and David B. Johnson. Emulation of multi-hop wireless ad hoc networks. In *The 7th International Workshop on Mobile Multimedia Communications (MoMuC 2000)*, 2000.

[25] J. Dike. User-mode linux. In *5th Annual Linux Showcase and Conference*, October 2001. http://www.linuxshowcase.org/2001/dike.html.

[26] Vmware. http://www.vmware.com.

[27] Plex86. http://plex86.sourceforge.net.

[28] Iperf. http://dast.nlanr.net/Projects/Iperf/.

[29] Kernel aodv. http://w3.antd.nist.gov/wctg/.

[30] Aodv-uu. http://user.it.uu.se/ henrikl/aodv/.