MobiNet: A Scalable Emulation Infrastructure for Ad Hoc and Wireless Networks

Priya Mahadevan
UC, San Diego
pmahadevan@cs.ucsd.edu

Adolfo Rodriguez
IBM, RTP and Duke University
razor@cs.duke.edu

David Becker
Duke University
becker@cs.duke.edu

Amin Vahdat [*]
UC, San Diego
vahdat@cs.ucsd.edu

Abstract

The current state of the art in evaluating applications and communication protocols for ad hoc wireless networks involves either simulation or small-scale live deployment. While larger-scale deployment has been performed, it is typically costly and difficult to run under controlled circumstances. Simulation, on the other hand, allows experimenters to quickly vary system configuration such as the MAC layer, routing protocol, and movement patterns. However, it typically cannot capture many of the important characteristics of real-world environments. Simulation requires the duplication of application, operating system, and network behavior within the simulator, not only decreasing accuracy, but also increasing development effort. While simulation and live deployment will clearly continue to play important roles in the design and evaluation of mobile systems, we present *MobiNet*, a third point in this design space, where the communication of unmodified applications running on stock operating systems is subject to the real-time emulation of a user-specified network environment. We believe that each of simulation, emulation, and live code deployment will play important roles in the life cycle of mobile system construction and experimentation.

MobiNet utilizes a cluster of *emulator* nodes to appropriately delay and drop packets based on MAC-layer protocols, congestion, queuing, and available bandwidth. MobiNet also emulates the characteristics of ad hoc routing protocols to build per-node routing tables. We describe our implementation of an 802.11-based MAC layer, an implementation of the DSR ad hoc routing protocol and evaluate MobiNet's accuracy using the random waypoint mobility model. The MobiNet infrastructure is extensible, thereby facilitating the development and evaluation of new MAC layers, routing protocols, and mobility models. Our evaluations show that MobiNet emulation is scalable and accurate while executing real code (such as video playback).

1 Introduction

Modern wireless mobile systems have become an increasingly popular technology in the past few years. New application classes vary from warehouse inventory to medical applications. At the same time, the number of mobile devices is exploding at exponential rates. Of particular interest has been the proliferation of *ad-hoc* wireless networking where mobiles nodes form peer relationships with one another to relay information through the network. Nodes communicate with other nodes in their range and act as forwarders of data from nodes communicating with out-of-range nodes.

One key challenge in this area has been evaluating the protocols and applications that function in this environment in a scalable and accurate manner. It is difficult and costly to deploy and coordinate development software on a

[*]This research is supported in part by the National Science Foundation (EIA-9972879), Hewlett Packard, IBM, Intel, and Microsoft. Vahdat is also supported by an NSF CAREER award (CCR-9984328).

large number of real mobile nodes and to control the operating conditions of such an experiment to obtain reproducible results. Live deployment also makes it difficult to reason about the behavior of a wireless system under varying system assumptions, such as different MAC layers, communication protocols, and wireless communication ranges. To overcome the cost, scale, and experimental limitations of live evaluations, researchers have developed simulation engines that attempt to mimic the behavior of mobile systems by modeling packet loss, queuing delays, MAC-layer behavior, and congestion. While simulation is a useful approach, it cannot accurately model key details of real implementations. Application code is typically re-written to conform to the simulation environment. For example, applications in the popular ns2 network simulator [11, 21] must be ported to TCL, conforming to the ns2 interfaces. While requiring increased development, this approach also leads to loss in accuracy as the behavior of a real application running over a real operating system, a real network stack all running on real hardware is lost. Finally, while increasing evaluation scalability, network simulators that attempt to faithfully capture detailed network characteristics cannot typically scale to simulations beyond tens of nodes.

MobiNet is an emulation environment designed to overcome some of the accuracy and scalability challenges in mobile evaluation. The goal of our work is to allow system developers and researchers to evaluate the behavior of their mobile and wireless systems under a range of conditions in a controlled, reproducible experimental environment leveraging a commodity workstation cluster. Thus, we wish to use MobiNet to answer the following types of questions:

- How scalable is a new ad hoc routing protocol for a target application deployment, MAC layer, and node movement pattern?
- What effect does a variant of TCP or a new reliable MAC layer have on end-to-end battery consumption?
- How resilient is the routing infrastructure to the failure of a varying percentage of wireless sensor nodes assuming different topologies, communication patterns, and routing protocols?
- Will a given network topology consisting of both fixed and mobile wireless infrastructure support adequate quality of service for a voice over IP application?

To support the above types of experiments in a controlled environment, we designed MobiNet to emulate a target mobile network on a scalable LAN cluster (with gigabit interconnect), enabling researchers to deploy unmodified IP-based software and subject it to faults, varying network conditions, different routing protocols, and MAC layer implementations. Edge nodes running user specified OS and application software are configured at the IP-layer to route packets through one or more MobiNet *core* nodes that cooperate to subject the traffic to the bandwidth, congestion, and loss profile of the target network topology. Because emulation occurs in the core nodes, client nodes can have arbitrary hardware and software configurations and can remain unmodified in this environment. In our experiments, we use Linux-based PCs as clients, though our emulation environment is general to a variety of operating systems.

We built MobiNet as an extension to the publicly available ModelNet wide-area network emulation software [17]. We leverage the observation that packets operating in target wireless network environments will have significantly less available bandwidth and will incur more delay than available in today's commodity local area network technology. Thus, we are able to appropriately delay packets as they travel through an emulated multi-hop network. Relative to wide-area emulation available through ModelNet, we must address three key challenges to successfully emulate large-scale multi-hop wireless networks. First, for wide-area networks, it is not necessary to emulate the characteristics of the MAC layer on a hop-by-hop basis, as the behavior of the MAC layer (e.g., Ethernet or 802.3) does not typically impact end-to-end packet behavior because of switched LAN technologies and over-provisioning. However, the behavior of the MAC layer (e.g., various flavors of 802.11) significantly impacts the behavior of multi-hop wireless networks. Next, node mobility and position plays a significant role in mobile and wireless environments, especially relative to typical wide-area networking scenarios where we assume that there is no node mobility. Finally, the behavior of the routing protocol plays a critical role in the performance of multi-hop and ad hoc wireless

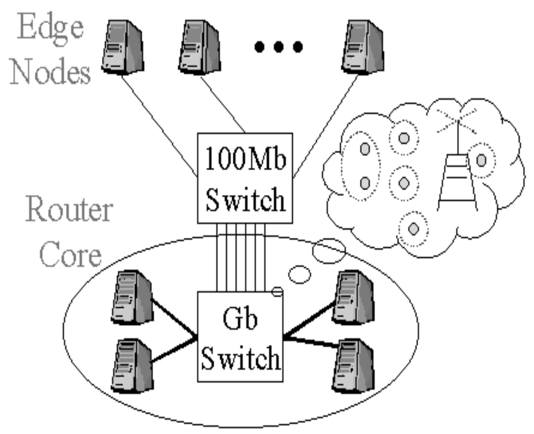

Figure 1: ModelNet Architecture

networks. The base ModelNet implementation precomputes all-pairs shortest path routes among all hosts. Clearly, this approach is inadequate for evaluating, for example, ad hoc routing protocols under a range of mobility models.

In this paper, we describe our experiences in accurate and scalable emulation of large-scale wireless networks, with a particular focus on: i) MAC layer emulation, ii) node mobility, and iii) routing protocol emulation. We present experimental evaluation of the working MobiNet system, including support for IEEE 802.11 MAC layer emulation, various node mobility models, and the DSR [5] ad hoc routing protocol. We demonstrate MobiNet's accuracy by comparing execution of our system running unmodified application binaries to ns2 simulation with similar communication patterns. We further quantify the scalability of MobiNet and find that a single MobiNet core can forward up to 89,000 packets per second. Additional capacity is available with additional MobiNet core nodes. Using just one MobiNet core and 2 physical edge nodes, we have been able to emulate a 200-node topology, forwarding application packets in real time.

Overall, we believe that MobiNet can qualitatively improve the realism of mobile system experimentation by allowing researchers and developers to subject their unmodified applications running on unmodified operating systems (ranging from perhaps TinyOS [12] for wireless sensor networks to Linux for more traditional productivity applications) to a wide variety of wireless network technologies, MAC layers, routing protocols, and usage scenarios. Relative to simulation, experimenters can gain confidence in the robustness of their code and can scale to much larger experimental environments by leveraging multiple nodes communicating in real time. The remainder of this paper is organized as follows. Section 2 provides an overview of the baseline hop-by-hop emulation environment. Section 3 describes the details of the MobiNet framework. We evaluate MobiNet's accuracy and scalability in section 4. Section 5 describes our experiences in deploying real unmodified applications over MobiNet. We discuss related work in section 6, present our discussions and future work in section 7 and conclude in section 8.

2 ModelNet Overview

MobiNet extends the wired network emulation provided by ModelNet [17]. While the details of the ModelNet infrastructure are beyond the scope of this paper, we provide a brief overview of it, particularly as it pertains to MobiNet. ModelNet was initially developed for testing large-scale distributed services for wired wide-area networks such as the Internet. It is capable of emulating large topologies with large numbers of clients. The ModelNet architecture is composed of *Edge Nodes* and *Core Nodes* as shown in figure 1. Edge nodes in ModelNet can run arbitrary architectures and operating systems. They run native IP stacks and function as they would in real environments with the exception that they are configured to route IP traffic through ModelNet cores. Modelnet core nodes run a modified version of FreeBSD to emulate topology-specific and hop-by-hop network characteristics.

Target applications run on edge nodes as they would in a real setting. However, to decrease the number of client (edge) machines required for large-scale evaluations, the ModelNet architecture allows for *Virtual Edge Nodes* (VNs). VNs enable the multiplexing of multiple application instances on a single client machine each with its own unique IP address. Because ModelNet clients use internal IP addresses (10.*), the number of clients that can be multiplexed onto an edge node is not limited by IP address space limitations, but rather by the amount of computational resources (e.g. threads, memory) that the target application uses. All VNs are configured to route their traffic through one of the ModelNet cores. Upon receiving a packet from a client, a core node routes the traffic through an emulated network of pipes, each of which represents a real link in the target topology. Each pipe is associated with emulation values for packet queue size and discipline, bandwidth, latency and loss-rate. The emulation executes in real time. Cores delay, shape, or drop input packets according to the emulation characteristics for each pipe. Hence, packets traverse the emulated network with the same rates, delays, and losses as they would in a real network. When a packet exits the chain of pipes, the core transmits the packet to the edge node hosting the destination VN.

ModelNet cores store pipes through which packets traverse for every source-destination pair in an n^2 matrix. Pipes are distributed among the cores to spread the emulation load. Packets are passed between the cores when the next hop on their path is handled by a different core than the current one. A flow diagram of the ModelNet packet processing in FreeBSD is shown in Figure 1. The grayed boxes are not part of the ModelNet kernel module. The routing module (in light gray) is responsible for all routing decisions made by the core. Upon receiving a packet from a VN, the core performs a lookup in the n^2 route matrix to determine the chain of pipes through which the packet should traverse. It schedules emulation for each hop in the pipe chain, forwarding for non-local pipe hops to the appropriate peer core. Once the emulation is complete, ip-output in FreeBSD forwards the packet to the edge node supporting the destination VN.

3 The MobiNet Framework

This section describes the extensions that MobiNet makes to the ModelNet framework to support the evaluation of wireless and ad hoc networks. MobiNet, like ModelNet, has edge nodes capable of supporting many platforms and operating systems. While our current experiments have been performed on edge nodes running Linux, our edge nodes could be a combination of different devices like laptops, PDAs, etc running different operating systems. As in ModelNet, edge nodes in MobiNet host multiple virtual nodes (VNs) to allow for large-scale emulations. MobiNet cores emulate wireless network behavior at multiple layers while eventually routing packets to the edge node hosting the destination VN. In addition, a core may route a packet to a different core if a next-hop pipe is emulated on that other core. MobiNet cores have modules that emulate MAC effects, routing behavior, and node mobility. We choose to incorporate this functionality into the core so that edge nodes can be left unmodified to support many platforms and operating systems. MobiNet emulation is a three step operation: topology creation, assignment of VNs and pipes to hosts and cores respectively, and application execution. A user creates a desired topology, MobiNet distributes pipes associated with the topology across the cores to distribute emulation load, assigns VNs in our emulated topology to edge nodes, and configures and executes the applications in the MobiNet emulation framework.

Relative to wide-area emulation, a mobile system emulation must perform three critical tasks. First, MobiNet must emulate mobility behavior. For mobile wireless networks, it is important to provide different movement patterns to nodes in the topology. Second, MobiNet routing must be dynamic. MobiNet implements a routing module that tracks the position of nodes and maintains a list of nodes within transmission range for each node. The routing module is responsible for finding routes to destination nodes as nodes in the topology follow different movement patterns. Third, MobiNet accounts for MAC layer collisions. Effects of packet losses due to collisions in the MAC layer play an important role in wireless networks, thus requiring MobiNet to emulate MAC layer behavior. The physical layer also plays an important role in wireless networks, hence we have a signal propagation model that combines a free space propagation model and a two-ray ground reflection model.

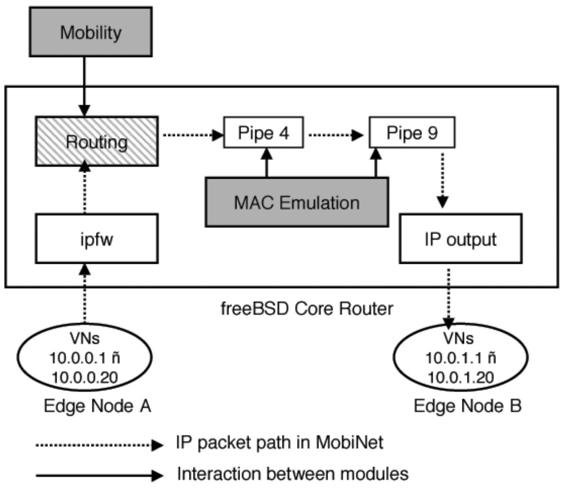

Figure 2: MobiNet Modules

By dividing mobile behavior under functional lines, MobiNet's modules are more easily developed and replaced. This allows experiments to use different combinations of modules, leading to a more flexible and powerful emulation framework. For example, one can compare different ad hoc routing protocols by plugging in different routing modules while keeping the mobility and MAC emulation modules constant. This provides a fair and consistent evaluation of the routing protocols in question. Alternatively, one can compare how a particular ad-hoc routing protocol performs or how much energy is consumed for a target communication pattern when using different MAC modules. This structure enables direct performance evaluation of mobile and wireless systems in a variety of scenarios. We believe this modularity to be critical to the ultimate utility of the MobiNet emulation infrastructure.

Figure 2 depicts the interactions between the different modules in MobiNet. The mobility model is implemented as a user level application that downloads new movement files into the MobiNet core's kernel at user specified time intervals. These files contain information that includes each node's current coordinates along with a list of its neighbors (nodes that are within a node's transmission range). The routing module uses this information to find new routes when existing routes become stale. Once a packet enters the system, it is handed up by the ipfw module in the FreeBSD kernel to the MobiNet module. The routing module within MobiNet is now responsible for finding a path in order to send this packet to its destination. The path is basically a list of nodes through which the packet has to traverse before reaching its destination. Once the path has been obtained, the MAC-layer module emulates the packet according to the specified attributes of each pipe in the path. Thus pipes in MobiNet correspond to the transmission capacity of their associated nodes. The packet traverses through every intermediate node's pipe, thereby being subjected to queuing delays and congestion at every node. Once the packet successfully reaches the last hop in the path, the packet is then sent to the virtual node hosting the packet's destination. Thus, transmitting a packet from source A to destination B via nodes C, D, and E will involve sending the packet through pipes A, C, D and E before finally relaying it to destination B.

Each pipe maintains a queue for storing packets that need to be transmitted from that particular node. Queues are implemented in drop-tail fashion and hold up to 50 packets for the experiments that we describe in this paper, though this value is configurable. All attributes of pipes such as bandwidth, queue size and loss rate are user-configurable. The attribute values for each node can be downloaded into the core's kernel using the *sysctl* function call in FreeBSD.

One important thing to note is that users have the option of turning off MAC, physical and routing layer emulations at the core. One can directly plug in wireless devices such as PDAs or laptops having real MAC cards directly to the core. This obviates the need for the core to perform MAC layer emulation. For ad hoc wireless networks, the wireless devices could also run an instance of an ad hoc routing protocol on the edge host. This feature allows users to evaluate real physical and MAC layers with the cores concentrating on traffic shaping functionality. However, we believe that while this approach will be more accurate due to the fact that it uses a real MAC layer, it won't offer the scalability that one would get by emulating physical, MAC, and routing protocols at the cores.

3.1 Mobility

The mobility module is a user-level application that generates various node positions and neighbor lists consisting of nodes within a node's transmission range at certain times. This information is downloaded into the kernel of the MobiNet cores at regular user-specified intervals. Alternatively, we could calculate these positions and neighbor lists in real time within the core's kernel. Doing so, however, would cause significant overhead since floating point operations would be required in the kernel. Instead, MobiNet's mobility application pre-computes this information, thus avoiding critical-path computations. These values are stored in time indexed files and downloaded into the kernel in real time, enabling various movement patterns during emulation.

One interesting parameter in MobiNet's emulation is that of the interval used to refresh node positions within the core's kernel. If the interval is too high, valuable kernel processing is wasted in reading new node coordinates for values that have changed little. If this interval is too low, node coordinates quickly become stale. When new coordinates are downloaded, many routes suddenly are invalid, leading to a storm of routing updates. Both of these anomalies ultimately lead to inaccurate results. MobiNet attempts to bridge the gap between kernel performance and accuracy by choosing an interval value that provides good performance and accurate results under a wide variety of emulations. We found that setting the node position refresh rates to 0.5 seconds provides good results for our test scenarios, with velocities up to 20 m/s. While we can download new positions into the kernel at a much lower granularity, we found that the results obtained at 0.5 second refresh rate compared favorably with lower values of refresh interval, and without incurring too much overhead. However, the refresh interval is user configurable and for high node mobility, it is recommended that users specify a low refresh interval, while in scenarios where the nodes move at relatively low rates, the refresh interval can be much higher.

Our current mobility application supports the random waypoint mobility model described in [1], though MobiNet can use arbitrary movement models. In our applications, users specify the topology size, the duration of the experiment, the maximum speed of nodes, the movement *pause time*, the interval of desired output, and the seed for the random number generator we use (this allows us to recreate exact mobility files and average results over various seeds). The application creates time-indexed movement files that include the current positions of each node and the neighbor lists for each node. These movements files can be read by the MobiNet core during the execution of the experiment. We also provide an interface to export the same movement files to ns2 [11] allowing us to directly compare MobiNet emulations with ns2 simulations for identical node movement patterns.

In our random waypoint model, nodes are spread over a rectangle so that the resulting density is 9000 m^2 per node. For example, for 50 nodes, the resulting rectangle is of size 1500 meters by 300 meters for a total of 450000 m^2. We also support a square with the same density. In this case, the sides of the square are 635 meters each. In our experiments, we obtained similar results for the rectangle and square models. At the start of the experiment, each node is assigned a random start position in the rectangle and waits for *pause time* seconds at its initial position. Each node subsequently picks a random destination point in the rectangle and begins moving toward that point with a random speed uniformly selected from 0 m/s to the maximum speed specified. Once the node reaches its destination, it pauses at that point again for *pause time* seconds. After the pause, it again picks another random destination and continues the process described above. In our emulations, we used different values of pause time: 0, 30, 60, 90, 120, and 300 seconds. A pause time of 0 seconds indicates that the nodes are moving constantly, while a 300 second pause time implies no motion at all for our default 300-second experiments. We used a number of speed values, though report results only for runs using 1 m/s and 20 m/s. The specified interval values determines when node positions and computed neighbor lists of each node are written to a movement file. The routing module in MobiNet uses the node locations and neighbor lists to determine the set of nodes that each node may communicate with each other in the kernel (basically the set of edges to insert between nodes in the topology). These nodes are those within a specified transmission range. By default (and for our emulations) we use a transmission range of 250 meters (though this value is configurable).

3.2 MAC Layer Emulation

Our modular emulation approach is amenable to a wide range of models for the MAC layer. In this section, we describe our implementation of an emulated 802.11 channel based on IEEE's 802.11 standard specification. However, we stress that there is no perfectly accurate MAC layer model. For instance, it may be more appropriate to emulate bit errors on a per-packet basis based on measurements of a live wireless network deployment as done in TOSSIM [9]. This approach would certainly reduce the overhead of our current implementation and would have some basis in measurement. However, it would come with the downside of not capturing potential radio interference from simultaneous packet transmission.

We leave evaluation of different MAC modeling techniques to future and describe here our 802.11 bit-level emulation model. We implemented the MAC model along 802.11's specifications for RTS-CTS-Data-ACK pattern. The Distributed Coordination Function(DCF) [2] in our 802.11 implementation models data contention in the wireless environment. We also implemented the physical link model, that computes the power level at which packets are received before handing the packet to the MAC layer.

In order to send a packet, a node first performs a carrier sense to check if the transmission medium is idle. Nodes defer transmission to nodes that are currently transmitting and employ an exponential backoff algorithm to avoid congestion. On detecting that the medium is idle, a node first sends a request-to-send (RTS) packet to the receiver. Upon successfully receiving the RTS packet, the receiver transmits a clear-to-send (CTS) packet to the sender. The sender then prepares to send the actual data packet to the receiver. If it does not receive the CTS from the receiver, it assumes the RTS packet was lost and performs the exponential random backoff algorithm before attempting to send the packet again. On successful receipt of CTS, the sending node sends the data packet. Once the node succeeds in sending the packet, it waits for an acknowledgment (ACK) from the receiver. If the sender does not receive an ACK within its ACK timeout period, it assumes the packet is lost, increments its contention window as well as the packet TTL value. If the packet has reached its maximum TTL value, the packet is discarded. If the packet has not reached its TTL value, the sender increments its congestion window by a power of 2, performs the random backoff, and prepares to send the packet again when the medium becomes idle.

In our current implementation, we maintain a list for all the nodes in our system called the *in-flight* list. Corresponding to each node entry in the list, we maintain the current packet being transmitted by that node along with the start and calculated end transmission times (determined by the bandwidth of the medium and the size of the packet). Before a node starts the transmission process, it checks this list to ensure that none of its neighbor nodes are transmitting at that time. If none of its neighbors are transmitting and the medium has been idle for *DIFS* (DSC Interframe Space) seconds, the node begins its transmission and updates its status in the *in-flight* list. If two entries in this list are destined for the same receiver at overlapping time intervals or if the receiver cannot receive the packets correctly due to collision with other ongoing transmissions at that time, we mark the corresponding packets as captured depending on the power level at which they were received.

The physical layer plays an important role in the performance and energy consumption of mobile and wireless systems. The free space model and the two-ray model predict the received power as a deterministic function of distance [1]. Our physical link model combines both the free space propagation and two-ray ground reflection model. When the receiving node is within the reference distance (100 meters) of the sending node, we employ the free space model where the signal degrades as $1/r^2$. Beyond this reference distance, the signals degrades as $1/r^4$. When a packet is received, before being processed by the MAC layer, we compute the power level at which the packet was received. We compare this value to the carrier-sense threshold and the receive threshold. If the received power level is below the carrier-sense threshold, we discard the packet as noise, while if the power level is above the carrier-sense threshold, but below the receive threshold, it is marked as error. In case of overlap of two packets at the receiver, we check the power levels at which both the packets were received. If the power level of one of the packets is at least 10 dB greater than the power level of the other packet, we assume capture and the weaker packet is dropped. Otherwise, both the packets are assumed to have interfered with each other and both are dropped.

3.3 Dynamic Routing

As with all other MobiNet modules, the routing layer is implemented as a pluggable module in the FreeBSD kernel. The general idea is that the user will be able to test any desired routing protocol without having to change any other module in MobiNet. The MobiNet core makes a call to this routing module to retrieve paths for the packets that it receives.

We have implemented the Dynamic Source Route (DSR) [4, 5] protocol in the MobiNet core. DSR uses source routing rather than hop-by-hop routing. While we chose DSR in our current implementation, DSR can be replaced with any other ad-hoc routing protocol such as AODV [15], DSDV [16], or TORA [13, 14]. Our generic design and the fact that each component in MobiNet is pluggable and not dependent on other components enable us to implement a broad range of routing module in the kernel with relative ease. None of the other components (e.g. packet scheduling, emulation, underlying MAC layer) would have to be modified.

When a packet enters the MobiNet core, MobiNet queries its route cache to check if a route exists for that source-destination pair. If a route exists, it appends the source route to the packet and the packet is transmitted hop-by-hop as dictated by the route and the occupancy of send queues at intermediate nodes along the path. If a route is not present, MobiNet buffers packets received for that destination while it performs the DSR ROUTE DISCOVERY mechanism. In our experiments, we allow nodes to buffer up to 50 packets while awaiting routes, though the buffer size is configurable. The source node creates a ROUTE REQUEST packet, adds itself to the route list, and broadcasts it to all of its current neighbors. The neighbors check their route cache to see if they have a route for that destination. If they do, they send back a reply to the sender. If they do not, they broadcast the request further to all of their neighbors, appending their own node-id to the route list in the packet. Assuming a connected network and that a sufficient number of packets are not lost, the destination node receives the ROUTE REQUEST. At this point, it reverses the route list, stores the route to the sender in its route cache and forwards a ROUTE REPLY to the sender. All nodes present in the route list update their route caches with routes for the sender and destination before forwarding the reply back along the path to the sender.

Once the sender receives a ROUTE REPLY, it stores the route in its cache, appends the source route to all the packets stored in this buffer for that destination and begins transmitting the packets. Before passing the packet to the next-hop, MobiNet checks to see if the next hop node is still within transmission range of the current node. If the next-hop node has moved out of transmission range, it invalidates the current route, drops the packet and sends an ICMP ERROR message back to the sender. The ICMP ERROR message also has the id of the node that moved out of range. The sender then transmits a ROUTE REPAIR message for the destination. The ROUTE REPAIR is similar to ROUTE REQUEST, except that ROUTE REPAIR also specifies the broken link in the previous cached route. If the neighboring nodes do not have routes for that destination, they broadcast the ROUTE REPAIR to their neighbors. Nodes that do have routes for that destination check the routes present in their caches and if the broken link is part of their stored route, they invalidate it and propagate the ROUTE REPAIR message further. Otherwise, they send back a ROUTE REPLY to the sender with their stored route.

Our DSR implementation does not implement all of the optimizations in DSR specifications and incorporated in ns. Specifically, in *salvage-with-cache*, a node consults its cache for a route on a transmit failure and salvages the packet using the route if possible. Next, in *use-tap*, a node listens to a promiscuous tap from the MAC layer. Finally, with *ring-zero-search*, a node first sends a non-propagating ROUTE REQUEST to its neighbors and waits to hear back from them. The neighbors do not forward this request to their neighbors, only replying back if they have the route in their cache. If the node has not heard back from any neighbor within a timeout, it transmits a standard propagating ROUTE REQUEST. We are currently in the process of implementing these three optimizations in MobiNet.

4 Evaluation

We have written and tested a simple application in native TCP/IP and in the ns-2 network simulator to enable comparisons between MobiNet emulation and ns2 simulation. The application establishes simple constant bit rate (CBR) streams between senders and receivers using UDP. Each sender sends data to exactly one receiver. Our CBR communications consists of 64-byte packets sent from each node (sender) at the rate of 4 packets per second. While it is impossible to guarantee that both versions function identically, the simplicity of our test application leads to it exhibiting very similar behavior in both environments. Using this application, we have executed a number of experiments to evaluate the performance, scalability, and accuracy of the different modules in MobiNet. The goal of our accuracy and routing overhead tests were to reproduce the experiments described in [1].

In all of our experiments, MobiNet edge nodes consisted of Pentium 4 2.0 GHz PCs with 512 MB memory running linux version 2.4.2. We use a single Pentium 3 dual processor with 2 GB memory supporting FreeBSD version 4.5 as our MobiNet core. Our experiments on ns2 were conducted on a machine similar to our edge nodes. MobiNet provides various packet statistics that enable us to determine the number of packets sent, packets dropped due to MAC collision, and other useful metrics. Likewise, we make use of ns2 trace files to extract these metrics.

With our mobility application, we simulated random waypoint mobility using various seeds, resulting in different movement patterns. We have tested a number of *maximum speed* values, but present results only for 1 m/s and 20 m/s speeds. Our application creates movement files that describe the positions and neighbor lists of nodes at the specified interval. This information is used by MobiNet and ns2 during emulation/simulation, allowing each to update node coordinates and lists of nodes within communication range. For most of our experiments, we specified an neighbor-refresh interval of 0.5 seconds.

In addition to creating interval-based movement files, our mobility application also creates a *continuous* description of node movement. Here, we specify node destinations and the speed at which nodes move toward those destinations in our continuous movement file. ns2 has native support for mobility and automatically computes positions and neighbor list for each node at every time instant. (though ns2 discretizes events, these effects are insigficant). We found that our interval of 0.5 seconds gives us comparable results with lower intervals such as 0.2 seconds and also with the continuous movement pattern in ns2.

4.1 Core Performance

One of the experiments we have executed, tested the ability of the MobiNet core to process packets. The goal was to find the number of packets per second the MobiNet core router could emulate without saturating the core. To this end, we disabled DSR in the MobiNet core and stored pre-computed paths from sources to destinations. Later, we enabled DSR to compare the performance of the core with DSR enabled, and thereby compute the overhead due to DSR. We ran similar experiments with two topologies, one with 200 nodes and the other with 500 nodes. For brevity, we only describe the results for the 200 node topology here.

The 200 VNs in the topology were distributed across 2 edge nodes (100 virtual IP addresses mapped to each edge node). Our simple application sent 64-byte UDP packets at a constant bit rate from every virtual node (VN) hosted by one of these edge machines to specified listeners running on the other edge machine. Thus, there were 100 flows

CPU utilization at core	Pkts/sec forwarded for 1 hop	Pkts/sec forwarded for 3 hops	Pkts/sec forwarded for 5 hops
40%	35K	20K	10K
50%	45K	27K	17.5K
70%	65K	40K	25K
90%	80K	48K	32K
100%	89K	52K	37K

Table 1: Forwarding capacity at the Core with DSR disabled

CPU utilization at core	Pkts/sec forwarded for 1 hop	Pkts/sec forwarded for 3 hops	Pkts/sec forwarded for 5 hops
40%	33.5K	18K	8K
50%	43.5K	25K	16K
70%	63.5K	38K	23K
90%	78K	47K	30K
100%	86K	50K	35K

Table 2: Forwarding capacity at the Core with DSR enabled

from the sender edge machine. Each VN sent packets to exactly one destination VN which was also the node's sole neighbor. Each packet could be sent from the sender to the destination in exactly one hop and there were no packet collisions as each sender had merely the destination VN as its neighbor. We also set the DIFS and SIFS values in the 802.11 specifications to zero as the goal was to gauge the maximum number of packets that could be sent through a single MobiNet core.

We measured throughput in terms of packets per second and CPU utilization at the core for different packet sending rates. Our results are shown in Table 1. By varying the number of packets sent by our CBRs, we measured the number of packets that the core could forward. We found that a single core could forward up to 89,000 packets/second for the baseline case of 1 hop at which point the CPU is completely utilized. Our MobiNet core runs with a clock resolution of 10Khz, meaning that we are able to accurately emulate each packet hop to within 0.1 ms accuracy. Even for end-to-end path lengths of 10 hops, packet transmission delays are accurate to within 1 ms, sufficient for our target wireless scenarios, especially when considering end-to-end transmission, propagation, and queuing delays. This accuracy holds up to and including the peak emulation rate because MobiNet's emulation runs at the kernel's highest priority level. As long as a packet is able to enter the kernel, its per-hop emulation time will be accurate to with 0.1 ms [17].

We ran similar tests but with different topologies, so that each packet from the sender must traverse 3 hops and 5 hops respectively before reaching the destination. Again, we ensured that there were no collisions and nodes just had their communication partners as their neighbors. As the number of hops increased, we found that the total number of packets that the core could forward per second decreased as it now had to perform more work per packet. We summarize our results in Table 1.

While maintaining the same topology, we then enabled DSR instead of providing precomputed routes to MobiNet. The core forwards approximately the same number of packets per second as with DSR disabled. This is because the nodes in our topology are stationary and once a route has been found, it does not change for the duration of the experiment. The slightly lower packets per second forwarded compared to the no DSR case is due to the fact that

	40 pkts/sec		400 pkts/sec		4000 pkts/sec	
Sender	ns	MobiNet	ns	MobiNet	ns	MobiNet
node 1	10000	10000	4573	4634	502	521
node 2	10000	10000	4603	4527	483	477
node 3	10000	10000	4697	4594	519	526
node 4	10000	10000	4635	4674	509	501

Table 3: Packets received by central node (node 5) in square topology for ns2 and MobiNet for various sending rates: 40, 400, and 4000 pkts/sec

the core has to do a little extra work in DSR such as ensuring the next hop is still a valid neighbor, etc. Table 2 summarizes these results.

4.2 MAC layer accuracy

Validating the behavior of our MAC layer implementation is difficult as no known emulation or simulation technique can accurately predict the bit error rates or radio interference under arbitrary deployment scenarios. We believe our architecture to be general to a wide variety of MAC layer models. However, to gain some baseline confidence in the accuracy of our 802.11 MAC model, we conduct micro-benchmarks to compare MobiNet's MAC layer performance with that of ns2 for a variety of topologies and packet transmission rates. Since the packet transmission rate is dependent upon the timing and rate of collisions, we hypothesize that if MobiNet and ns2 deliver the same packet throughput under a range of conditions, the packet collision and backoff behavior is likely to be similar. We chose several topologies and had nodes sending packets to each other at different rates. Since we are comparing the performance of our MAC model, we turned off DSR in MobiNet and supplied MobiNet with precomputed routing tables. Nodes are stationary, hence the routes are valid for the entire duration of the experiment. Our topology is shown in Figure 3.

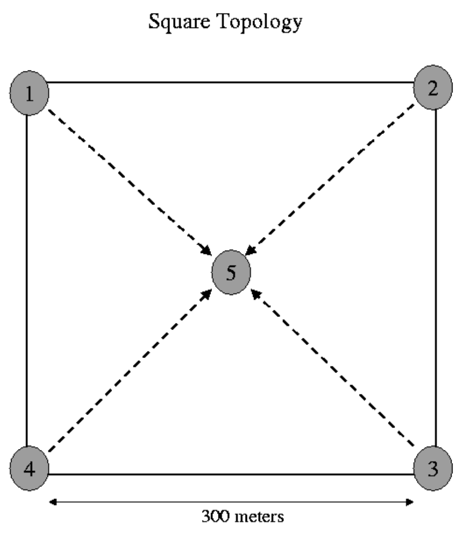

Figure 3: Square topology used for MAC validation. Nodes 1, 2 3 and 4 send packets to node 5

	50 pkts/sec		500 pkts/sec		1000 pkts/sec	
Receiver	ns	MobiNet	ns	MobiNet	ns	MobiNet
node 1	10000	10000	1108	1050	443	480
node 2	10000	10000	2370	2405	991	920
node 3	10000	10000	3201	3166	1387	1365
node 4	10000	10000	959	1004	561	606
node 5	10000	10000	2334	2402	1616	1678
node 6	10000	10000	3284	3347	1320	1369

Table 4: Packets received each receiver in ring topology for ns2 and MobiNet for various sending rates: 50, 500, and 1000 pkts/sec

Four nodes were placed at the four corners of a square of side 300 meters. A fifth node was placed at the center of the square. All the four nodes at the corners of the square sent packets to the central node. The transmission range of the nodes was 250 meters. Each corner node sent a total of 10,000 UDP packets to the central node, where each packet was 64 bytes long. We report the total number of packets that the central node received from each of the senders in both MobiNet and ns2. We present results for 3 different packet transmission rates: 4000 pkts/sec, 400 pkts/sec and 40pkts/sec. As the packet transmission rate increases, the contention at the central node increases, leading to more packet drops. At a lower packet sending rate, contention is not a problem at the receiving node and hence it receives all the packets sent to it by the various sending nodes. The values shown in Tables 3 are averaged over 3 different runs of the experiment. We see that the average number of packets received by the central node from the 4 senders in both ns2 and MobiNet is approximately the same.

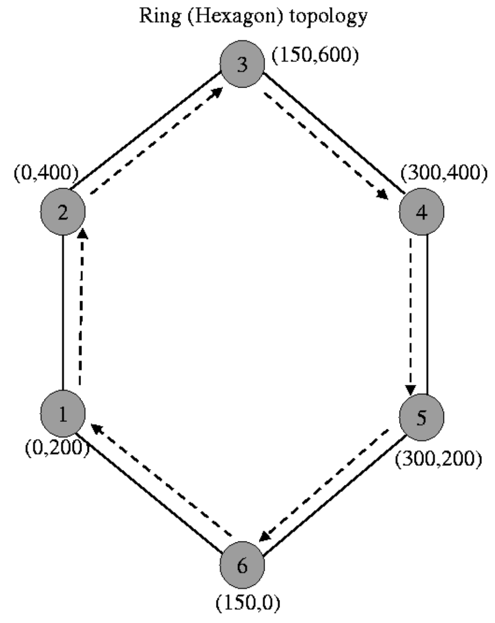

Figure 4: Ring topology used for MAC validation

We ran similar experiments for a different topologies. We present the results from a ring (hexagon) topology. The six nodes in our topology form a ring, with each node sending packets to the node on its right as shown in Figure 4. Node 1 and node 2 are 200 meters apart, node 2 and node 3 are 250 meters apart, node 3 and node 4 are 250 meters apart, node 4 and node 5 are 200 meters apart, node 5 and node 6 are 250 meters apart and node 6 and node 1 are 250 meters apart. As before, each node establishes CBR communication with the node on its left, sending 64 byte sized packets for a total of 10000 packets each. Our results are shown in Table 4. We experimented with different

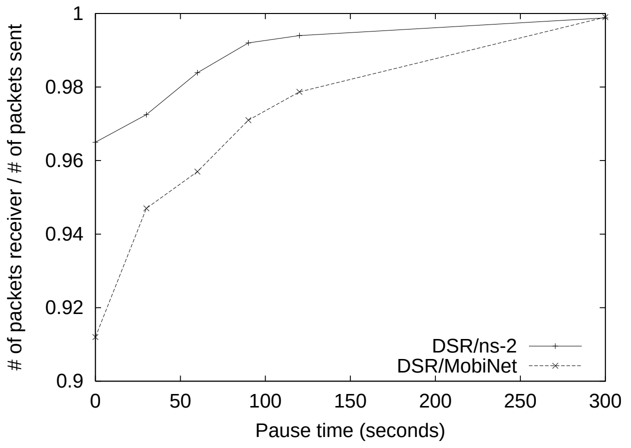

Figure 5: Packet delivery ratio as a function of pause time in ns2 and MobiNet for maximum speed of 20m/s.

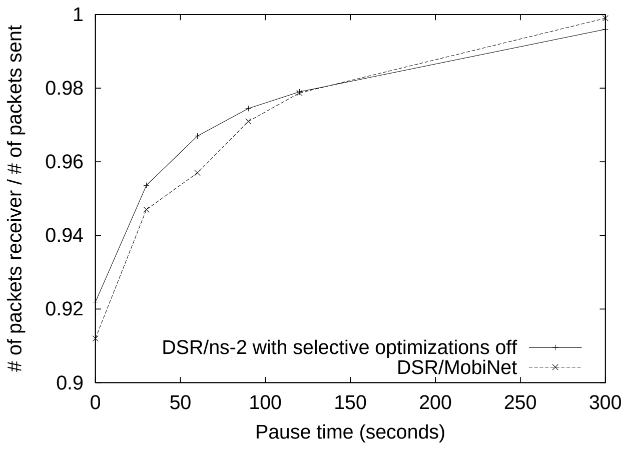

Figure 6: Packet delivery ratio as a function of pause time in ns2 and MobiNet with identical sets of DSR optimizations and maximum speed of 20 m/s.

packet transmissions rates and present the ones that are most interesting in terms of contention at different nodes and packet drops.

4.3 Routing Accuracy

We validated the emulation accuracy of MobiNet by comparing experimental results obtained from MobiNet to that from ns2 for our simple CBR communication. We used the 802.11 MAC protocol and DSR implementations available in ns2. Using our mobility model, we generated movement files that were used by ns2 and MobiNet.

In our first accuracy experiment, 50 nodes moved according to the random waypoint model in a rectangular strip of size 1500 meters by 300 meters with a maximum speed of 20 m/s. 10 nodes were chosen randomly to be the designated senders. Each sender sent all of its packets to a fixed randomly chosen destination for the duration of the experiment. Because our experiment ran for 300 seconds, each CBR sent 1200 packets for a total of 12000 packets sent in aggregate. We measured the packet delivery percentage for the above experiment with ns2 and MobiNet. We varied the pause times from 0 seconds (high movement) to 300 seconds (no movement). Figure 5 shows our results averaged over four different random seeds.

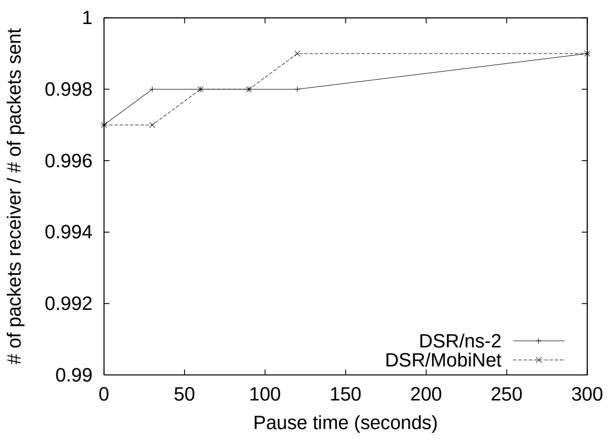

Figure 7: Packet delivery ratio as a function of pause time in ns2 and MobiNet. maximum speed is 1 m/s

We found that MobiNet performs as well as the ns2 for low and no movement. The lower delivery rate for higher mobility results from the fact that we have not implemented all features of DSR in MobiNet as noted in the previous section. To validate our claim that the lower accuracy for high mobility scenarios, we compared the performance of MobiNet with the appropriate optimizations turned off in ns2. We find that MobiNet's packet delivery ratio is very similar to the packet delivery ratio in ns2 without the optimizations. The results are summarized in Figure 6.

To further validate our emulator, we repeated the previous experiments with a reduced maximum speed in our random waypoint model of 1 m/s, resulting in lower node mobility. Figure 7 shows our results. We found that in this case, MobiNet's packet delivery ratio is similar to the values obtained from ns2 for all values of pause time. As in the earlier experiments, we chose 10 CBRs, each generating 64 byte sized packets at the rate of 4 packets per second. The results for 1 m/s maximum speed closely match those of the ns2 model.

4.4 Routing Overhead

We ran further tests to determine the number of control packets transmitted by our implementation of the DSR routing protocol relative to the number transmitted by the ns2 implementation. Figure 8 shows the number of routing packets sent by ns2 and MobiNet to achieve the packet delivery ratio in Figure 5. Nodes move with a maximum possible speed of 20m/s. Since we are yet to complete the 3 optimizations of the DSR protocol in MobiNet, the number of control packets sent by MobiNet is almost 2 times that sent by ns2 for lower values of pause time. For higher values of pause time, the number of routing packets sent by MobiNet either closely matches or is equal to that of ns2.

Again, to validate our claim that the routing overhead in MobiNet is due to incomplete DSR optimizations, we measured routing overhead in ns2 with the optimizations turned off. In this case, the routing overhead in MobiNet is similar to the overhead in ns2 without the optimizations, as shown in Figure 9.

We again repeated our experiments with a reduced maximum speed of 1 m/s. In this lower mobility evaluation, we found that MobiNet compares favorably with ns2 in terms of number of routing packets. These results are shown in Figure 10.

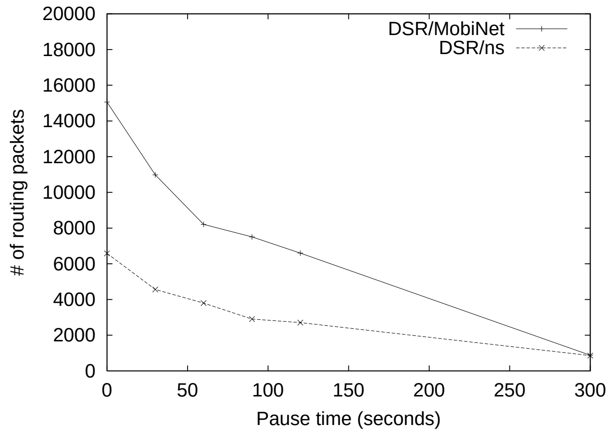

Figure 8: Routing overhead in ns2 and MobiNet for maximum speed of 20 m/s

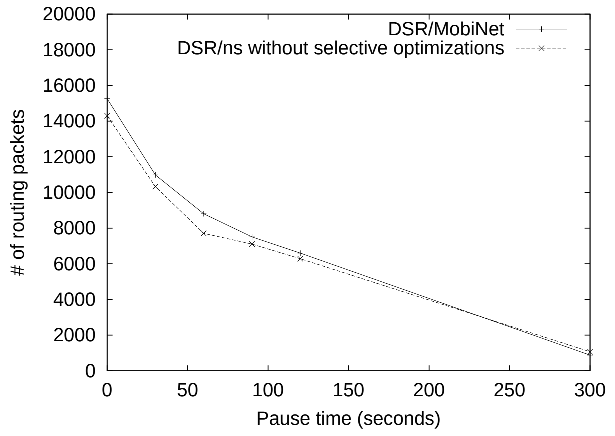

Figure 9: Routing overhead in ns2 with selective optimizations and MobiNet for maximum speed of 20 m/s

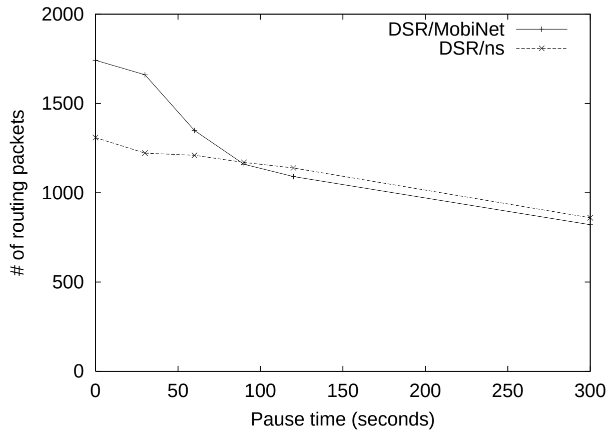

Figure 10: Routing overhead in ns2 and MobiNet. Maximum speed is 1 m/s

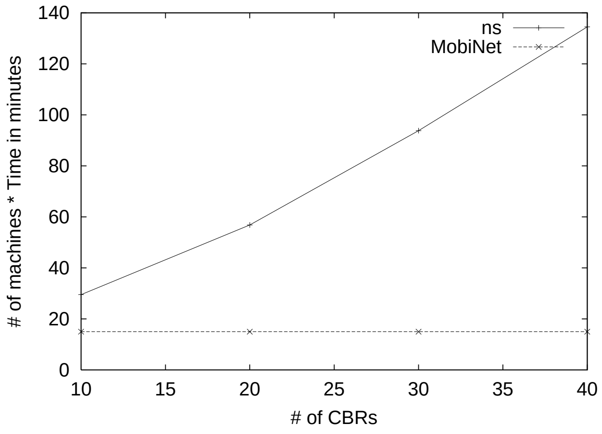

Figure 11: Scalability in MobiNet vs. ns2 as a function of time

4.5 Scalability

One of the main benefits of MobiNet over using a simulator such as ns2 is that experiments can be run in real time. We have already shown that excluding DSR and RTS/CTS exchange, a single MobiNet core is capable of forwarding up to 89,000 packets per second. To further show the usefulness of our emulator, we compared the time required to run experiments with varying numbers of senders. We used a 200 node topology with nodes distributed randomly in a 3000 meter by 600 meter rectangle (resulting in the same node density as our previous experiments). For MobiNet, the 200 nodes were distributed across 2 MobiNet edge nodes. The ns2 experiments were run on a single machine with the same configuration as the MobiNet edge node. We disabled node mobility in this case to reduce the overhead due to finding routes with DSR. Here, DSR only needs to find routes to destinations once (at the start of the experiment).

We varied the number of CBR sources from 10 to 40, with each sender once again transmitting 64-byte packets at the rate of 4 packets per second. Each node sent a total of 1200 packets. Figure 11 shows the computation time necessary to execute the experiment for MobiNet emulation and ns2 simulation. This is taken as the time it takes for the experiment to complete multiplied by the number of machines used in the experiment. In real time, this experiment takes 5 minutes, as it takes each CBR 300 seconds to transmit its share of packets. As a result, MobiNet using 3 machines (2 edges and one core) emulates the experiment in 15 minutes. In contrast, ns2 simulation time of the experiment increases linearly with the number of CBR nodes. In the case of 40 nodes transmitting, the ns2 simulation lasted 134.5 minutes, compared to MobiNet's 15-minute emulation.

5 Deploying Real Applications

In this section we demostrate the utility and generality of our infrasturcture by deploying and evaluating real unmodified code, a video player over MobiNet. We used XAnim as our sample application. XAnim is a program that plays a wide variety of animation, audio and video formats on unix X11 machines. Running the same application on ns2 would be difficult to impossible. Our goal was to study the performance of the the video player in a ad hoc wireless network as a function of node movement.

We started with a wireless topology consisting of 50 nodes moving according to the random waypoint movement model, where the maximum random speed was set to 1 m/s. The nodes in our topology were hosted on two edge machines, thus each edge node was responsible for 25 VNs. We randomly chose two VNs from our topology. XAnim was deployed over one of the VN, while the display was set to the other VN. Communication between these

Pause time (s)	1 m/s	5 m/s	20 m/s
0	14500	13708	5490
30	15596	13728	13031
60	14927	14565	13207
120	15889	15611	14752
300	16200	16100	16086

Table 5: x11 packets exchanged between 2 VNs in a 2 minute interval for various maximum speeds: 1, 5, and 20 m/s

two nodes ran over the x11 protocol. The VN executing XAnim would send its packets to the MobiNet core, which would use DSR to find a route to the VN hosting the display. The packets were emulated according to our 802.11 implementation in the MobiNet core and then sent to the destination VN which would display the movie. Due to node movement, if existing routes went stale, DSR was used to find fresh routes to the destination VN.

The video was replayed in a continous fashion for 2 minutes. The video had a total of 44 frames (at the rate of 15 frames per second). For lower node mobility scenarios, packet drops due to broken routes was low and we observed that the video played in an almost continuous manner. In a highly mobile environment, we found that the video clip would stall for a while during packet drops. Once routes were found, the clip would start playing again.

We recorded the total number of XAnim packets exchanged between the two VNs for different values of pause time. We compared the results obtained from MobiNet with those obtained from tcpdump packet capture and found the results to be accurate. We averaged the results over several runs of the experiment and present them in Table 5.

We duplicated the above experiments but varied the maximum random speed in the random waypoint movement model. We found that for very high movement scenarios, the display at the VN stops after a while. Unlike CBR communication, in the x11 communication that takes place between the XAnim nodes, loss of vital packets due to node movement leads to the application stalling for a while. For a maximum speed value of 20 m/s and pause time of 0 secs, we found that the video clip stopped playing after a while due to the loss of a few vital packets. We present results for maximum speed of 5 m/s and 20 m/s in Table 5.

6 Related Work

Zhang and Li [22] have built an infrastructure similar to MobiNet for testing mobile ad hoc networks. However, their work does not support any routing protocol. Furthermore, their scheme does not restrict application bandwidth, making experimental results inaccurate for a range of important application characteristics.

Noble and Satyanarayanan [10] use trace-based network emulation to play back measured mobile network characteristics to real applications. Our approach generalizes this technique, allowing users to generate their own mobility scenarios. Further, we can modify parameters of the underlying network with respect to MAC protocol, routing protocol, transmission range, and other network functionality.

Netbed [19, 20] is a network testbed at the University of Utah. Netbed provides a testbed of real mobile nodes using real mobile hardware and software. Though a powerful and accurate evaluation tool, Netbed is costly to implement and requires the coordination of many people, machines, and systems. In contrast to our work, Netbed is a real testing environment, not an emulation or simulation infrastructure.

Emwin [23] is a network emulator similar to MobiNet. While Emwin handles MAC emulation, it does not implement ad hoc routing protocols. While they also map emulated topology as in MobiNet, they do not have separate edge nodes and cores. Each of their edge node is able to support 6 virtual nodes. MobiNet separates emulation nodes (cores) and edge nodes and thus achieves scalability with far fewer nodes. While Emwin can emulate a 48-node topology with 8 physical machines, MobiNet can easily emulate a 200 node topology with 3 physical machines with a similar configuration.

JEmu [3] is another emulation system for mobile and ad hoc networks. Each JEmu node runs a stack comprising of DSR, MAC and a JEmu Radio layer.The JEmu radio layer communicates with the JEmu emulator to handle collisions and to forward packets. Unlike MobiNet where the routing protocols, as well as MAC layer is implemented in the core, in JEmu, these modules are handles by a separate machine. To scale up to 200 mobile nodes, which required just 3 physical machines in MobiNet, JEmu will require 200 physical machines.

SeaWind [7] enables wireless emulation by changing various parameter sets. However, they do not provide a test-environment for ad hoc routing protocols. GloMoSim [8] is a network simulator for testing various protocols in a mobile wireless environment. GloMoSim enables the comparison of protocol performance at each network layer. It also provides scalability by allowing simulations to execute in parallel on a set of machines. While these features make GloMoSim attractive to researchers and offer additional scalability, it suffers from a number of the same problem as other simulators. For instance, it cannot support real applications, real traffic patterns, or real operating systems/hardware and therefore abstracts away key details of real implementations.

Walsh and Gun Sirer [18] have used staged simulation, a technique to improve the scalability and performance of network simulators. They have shown that ns2 tends to be slow and scales poorly with increasing number of nodes. By identifying and eliminating redundant computation using techniques like caching, they have shown that it is feasible to improve simulation time of large wireless networks.

TOSSIM [9] is a simulator for TinyOS wireless sensor networks. It captures network behavior by using a probabilistic bit error model. It also provides communication services for application interaction. While TOSSIM takes advantage of TinyOS's architecture and is specific to it, MobiNet is general to any hardware and operating system. Further, TOSSIM cannot capture the real time packet collisions that might take place for a given communication pattern or movement pattern, instead relying on the user to specify the bit error rate for a given deployment, and enforcing this error rate independent of communication time or location.

Judd and Steenkiste [6] describe an approach for wireless experimentation using a real MAC layer. While using a real MAC layer has advantages, scalability is limited as discussed above. Comparison between different MAC layers also becomes more difficult to perform.

7 Discussions and Future Work

In most of our experiments, we validated MobiNet against ns2 to increase our confidence in the accuracy of our results. We felt that this was an appropriate choice because ns2, with mobile/wireless extensions, has undergone significant development and validation and remains one of the most popular simulators available. We leave comparisons against real wireless networks and traces collected from wireless environments for future work. The results obtained from live deployment of the video player also help further validate our implementation. The fact that MobiNet allows deployment of real code over real edge machines while forwarding packets in real time, provides a significant advantage over most simulation environments.

While we support MAC and physical layer emulation at the cores for better scalability, MobiNet is flexible to alternate models where, for example, users directly plug in wireless devices to the core, in effect running whatever MAC layer is available on the target hardware platform and performing routing protocols at the edges (rather than within the core). Using this approach allows users to evaluate real MAC and physical layers, while sacrificing some scalability and making it more difficult to extrapolate to other physical characteristics or deployment scenarios. However, this would serve as a straightforward validation of MobiNet's emulation behavior for a certain fixed deployment scenario. Another of our avenues for future work is investigating the scalability versus accuracy tradeoffs of such an approach.

Implementing MAC and routing modules in the FreeBSD kernel can be time consuming and thus yet another direction for our future work is developing APIs specific to the ad hoc and wireless environment that can simplify such application development. We believe that this will greatly simplify the task of writing new MAC modules and routing protocols for new users. While our current evaluation ran on a single emulation core, we are investigating efficient methods for supporting multiple cores. One approach is to randomly divide the nodes in the topology (and thus the pipes) among available cores. While the actual emulation for a pipe will only be performed at the core owning that pipe, we would require that all cores maintain coordinate and neighbor list information for all the nodes in the topology. We are also investigating more intelligent partitioning approaches so as to minimize cross-core traffic. We believe this to be NP hard, especially given unpredictable node movement patterns.

8 Conclusions

The overall goal of our work is to support controlled experimentation of a variety of communication patterns, routing protocols, and MAC layers for emerging ad hoc and wireless scenarios, including laptops, PDAs, and wireless sensor networks. Current approaches to such experimentation include simulation and live deployment of hardware. While each clearly has its relative benefits and will continue to play an important role in mobile system design and evaluation, this paper argues for the power of modular, real-time emulation as another important point in this design space.

To this end, we present the design and evaluation of MobiNet, a scalable and accurate emulator for mobile, wireless and ad-hoc networks. MobiNet provides accurate mobile and wireless emulation, comparing favorably with existing network simulators while offering improved scalability. At the same time, MobiNet offers enhanced realism relative to simulation because it executes a real-time emulation of just the network. Relative to live deployment of real hardware and real mobility scenarios, MobiNet provides the similar benefit of unmodified application binaries running on stock hardware and operating system configurations. However, it enables the system designer to rapidly experiment with a variety of MAC, routing, and communication (layers 2-4) protocols that may not be easily available in live deployments. Similarly, MobiNet allows experiments of different scale, communication patterns, and node mobility than may be available through a particular hardware configuration.

This paper describes the real-time hop-by-hop packet emulation in MobiNet along with three inter-changeable modules supporting various mobility models, MAC layer protocols, and ad hoc routing protocols. In particular, we describe and evaluate our implementation of a waypoint mobility model, the 802.11 MAC layer, and the DSR routing protocol. Our comparisons against the popular ns2 simulator verify the relative accuracy of our work, while quantifying MobiNet's performance and scalability. Finally, we show the power of our emulation environment by running an unmodified video playback application communicating across an emulated large-scale multi-hop 802.11 network using DSR on stock hardware/software.

References

[1] 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 *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, October 1998.

[2] IEEE Computer Society LAN MAN Standards Committee. Wireless LAN Medium Access Control(MAC) and Physical Layer(PHY) Specifications, IEEE Std 802.11. 1997.

[3] Juan Flynn, Hitesh Tiwari, and Donal O'Mahony. A Real-Time Emulation System for Ad Hoc Networks. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference*, January 2002.

[4] David B. Johnson. Routing in ad hoc networks of mobile hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, December 1994.

[5] David B. Johnson and David A. Maltz. Dynamic Source Routing in ad hoc wireless networks, Mobile Computing, edited by Tomasz Imielinski and Hank Korth. pages 153–181, 1996.

[6] Glenn Judd and Peter Steenkiste. Repeatable and Realistic Wireless Experimentation through Physical Emulation. In *Proceedings of HotNets-II*, November 2003.

[7] M. Kojo, A. Gurtov, J. Manner, P. Sarolahti, T. Alanko, and K. Raatikainen. SeaWind: a Wireless Network Emulator. In *Proceedings of the 11th GT/ITG Conference on Measuring, Modeling and Evaluation of Computer and Communication Systems (MMB)*, 2001.

[8] UCLA Parallel Computing Laboratory and Wireless Adaptive Mobility Laboratory. GloMoSim: A Scalable Simulation Environment for Wireless and Wired Network Systems. `http://pcl.cs.ucla.edu/projects/glomosim/`.

[9] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *To appear in proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys)*, November 2003.

[10] Brian Noble, M. Satyananarayanan, Giao Nguyen, and Randy Katz. Trace-based Mobile Network Emulation. In *Proceedings of SIGCOMM*, September 1997.

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

[12] University of California Berkeley. TinyOS. `http://webs.cs.berkeley.edu/tos/`.

[13] V.D. Park and M.S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proc. of IEEE INFOCOM '97*, May 1997.

[14] Vincent D. Park and M. Scott Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of INFOCOM'97*, April 1997.

[15] Charles Perkins. Ad Hoc On Demand Distance Vector(AODV) routing, Internet-Draft, draft-ietf-manet-aodv-spec-00.txt. November 1997.

[16] Charles Perkins and Pravin Bhagwat. Highly dynamic Destination Sequenced Distance-Vector(DSDV) for mobile computers. In *Proceedings of SIGCOMM 94 Conference on Communications Architecture, Protocols and Applications*, August 1994.

[17] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[18] Kevin Walsh and Emin Gun Sirer. Staged Simulation for Improving the Scale and Performance of Wireless Network Simulations. In *Procedings of the Winter Simulation Conference*, December 2003.

[19] Brian White, Jay Lepreau, and Shashi Guruprasad. Lowering the Barrier to Wireless and Mobile Experimentation. In *Proceedings of HotNets-I*, October 2002.

[20] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[21] Ellen W. Zegura, Kenneth L. Calvert, and Samrat Bhattacharjee. How to model an internetwork. In *IEEE Infocom*, volume 2, pages 594–602, San Francisco, CA, March 1996. IEEE.

[22] Yongguang Zhang and Wei Li. An integrated environment for testing Mobile Ad-Hoc Networks. In *Proceedings of MobiHoc*, June 2002.

[23] Pei Zheng and Lionel Ni. EMWIN: Emulating a Mobile Wireless Network using a Wired Network. In *Proceedings of WOWMOM*, September 2002.