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

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

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

David Becker
Duke University
becker@cs.duke.edu

Amin Vahdat
UC San Diego
vahdat@cs.ucsd.edu

The current state of the art in evaluating applications and communication protocols for ad hoc wireless networks usually involves either simulation or small-scale live deployment. Larger-scale live deployment is typically costly and difficult to run under controlled circumstances. Simulation allows more flexibility in varying system configurations, but requires the duplication of application and network behavior within the simulator. While simulation and live deployment will clearly continue to play important roles in the evaluation of mobile systems, we present *MobiNet*, a third point in this space. In MobiNet, the communication of unmodified applications running on stock operating systems is subject to the real-time emulation of a user-specified wireless network environment. MobiNet utilizes a cluster of *emulator* nodes to appropriately delay, drop or deliver packets in a hop by hop fashion based on MAC-layer protocols, ad hoc routing protocols, congestion, queuing, and available bandwidth in the network. Our evaluations show that MobiNet emulation is scalable and accurate while executing real code, including video playback.

I. Introduction

Modern wireless mobile systems have become an increasingly popular technology in the past few years. Of particular interest has been the proliferation of *ad-hoc* wireless networking where mobile nodes form peer relationships with one another to relay information through the network.

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 large number of real mobile nodes. It is also difficult to control the operating conditions of such an experiment for obtaining 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 these 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. Application code is typically re-written to conform to the simulation environment. While requiring increased development, this approach also leads to loss in accuracy as the behavior of an unmodified application running over a real operating system, network stack and hardware is lost. Finally, accurate simulation environments face significant scalability limitations, often topping out at a few tens of mobile wireless hosts.

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.

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 nodes assuming different topologies, communication patterns, and routing protocols?

[*] An initial version of this paper appeared as a short paper in the Wireless Traffic Measurements and Modeling Workshop, (WiTMeMo 2005). The current MC2R submission has validation experiments including comparison of our infrastructure with the popular ns2 simulator, which were not addressed in the WiTMeMo version. We have also described in detail each component of our emulation environment and our experiences in using it.

To support the above types of experiments, 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. We configure edge nodes running user specified OS and application software 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 wireless 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 [15]. 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 several key challenges to successfully emulate large-scale multi-hop wireless networks. First, for wide-area networks, it is, typically, not necessary to emulate MAC layer characteristics on a hop-by-hop basis, as the behavior of the MAC layer (e.g., Ethernet or 802.3) does not usually impact end-to-end packet behavior. 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, while nodes are mostly stationary in the wide-area. Finally, the behavior of the routing protocol plays a critical role in the performance of multi-hop and ad hoc wireless 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 [4] ad hoc routing protocol.

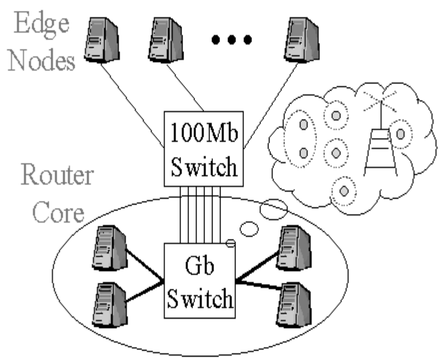

Figure 1: ModelNet Architecture

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. Using just one MobiNet core and 2 physical edge nodes, we have been able to emulate a 200-node topology, forwarding application packets in realtime. We also present results from running unmodified binaries (video playback) that demonstrate the power and flexibility of our system.

The remainder of this paper is organized as follows: Section II provides an overview of the baseline hop-by-hop emulation environment, ModelNet. Section III describes the details of the MobiNet framework. We evaluate MobiNet's accuracy and scalability in section IV. Section V describes our experiences in deploying real unmodified applications over MobiNet. We discuss related work in section VI and present our conclusions in section VII.

II. ModelNet Overview

MobiNet extends the wired network emulation provided by ModelNet [15]. While the details of the ModelNet infrastructure are beyond the scope of this paper, we provide a brief overview here for completeness, particularly as it pertains to MobiNet. ModelNet was initially developed for testing large-scale distributed services for wired wide-area network environments. 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 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, with each instance getting its own unique IP address. ModelNet clients use internal IP addresses (10.*), thus 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. ModelNet configures all VNs to route their traffic through a particular ModelNet core.

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. ModelNet cores store pipes through which packets traverse for every source-destination pair in an n^2 matrix. Cores delay, shape, or drop input packets according to the emulation characteristics for each pipe, with the emulation taking place in real time. 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.

A flow diagram of the ModelNet packet processing in FreeBSD is shown in Figure 2. The grayed boxes are not part of the ModelNet kernel module and represent some of the MobiNet extensions, described further below. 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 route matrix to determine the chain of pipes through which the packet should traverse. Once the emulation through the pipes is complete, ip-output in FreeBSD forwards the packet to the edge node supporting the destination VN.

III. The MobiNet Framework

This section describes our MobiNet extensions to the ModelNet framework to support the evaluation of wireless and ad hoc networks. Like ModelNet, the MobiNet architecture is composed of *edge nodes* and *core nodes*. The edge nodes support a variety of platforms and operating systems. While we perform our current experiments 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.

Relative to wide-area emulation, a mobile system emulation must perform the following additional critical tasks. First, MobiNet must emulate mobility behavior, i.e., different movement patterns of 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 support free space propagation and two-ray ground reflection models [1]. Supporting other propagation models should be straightforward as they become available.

While we implement the above modules in the core, an alternate design choice would involve implementing them in the edge nodes. We chose to incorporate the above modules in the core so that edge nodes can be left unmodified to support many platforms and operating systems. Another important consideration is that users have the option of selectively turning off MAC, physical and routing layer emulations at the core. One can directly plug in wireless devices such as PDAs or laptops with commercial off the shelf MAC cards directly to the core.

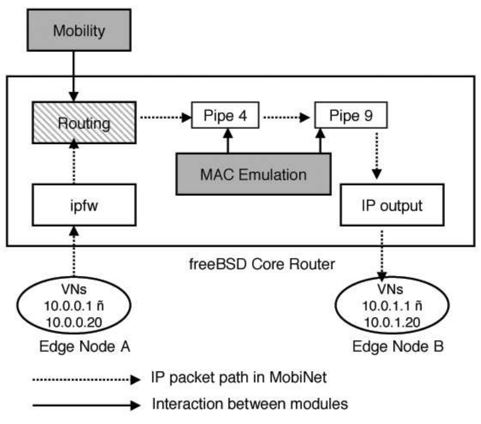

Figure 2: MobiNet Modules

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 packet forwarding functionality. However, we believe that while this approach will be more accurate due to the fact that it uses a real MAC layer, it will not offer the scalability that one would get by emulating physical, MAC, and routing protocols at the cores. Another disadvantage of this approach is its inability to model node mobility. Since the wireless devices are directly plugged into the core, they cannot be mobile.

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. We implement the mobility model through a user level application that downloads new node movement files into the MobiNet core's kernel at user specified time intervals. The routing module uses this information to find new routes as 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 is now responsible for finding a path 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 is obtained, the MAC-layer module emulates the packet transmission 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 every intermediate node's pipe, subject 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.

Running experiments in MobiNet is a three step operation: topology creation, assignment of VNs and pipes to hosts and cores respectively, and application execution. First, a user creates a desired wireless topology. MobiNet associates pipes with each node in the wireless topology in order to perform the emulation and evenly distributes these pipes across the cores to distribute emulation load. Next, MobiNet assigns VNs in our emulated topology to the edge nodes. It binds each application instance to the respective VN and executes the applications in the MobiNet emulation framework. We now describe each of MobiNet's modules in greater detail.

III.A. Mobility

The mobility module is a user-level application that generates various node positions and neighbor lists consisting of all nodes within a specified node's transmission range. The module downloads this information into the kernel of the MobiNet cores at regular user-specified intervals. Alternatively, we could calculate these positions and neighbor lists in realtime within the core's kernel. Doing so, however, would cause significant overhead since floating point operations would be required in the kernel (which are not natively supported in FreeBSD). Instead, MobiNet's mobility application pre-computes this information, stores the values in time indexed files and downloads into the kernel in realtime, 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 downloading new coordinates, 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 node 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 higher.

Our current mobility application supports the random waypoint mobility model [1], though MobiNet can support 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 (this allows us to recreate exact mobility files and average results over various seeds). The mobility 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 [10] allowing us to directly compare MobiNet emulations with ns2 simulations for identical node movement patterns. By default (and for our emulations) we use a transmission range of 250 meters, though this value is configurable.

III.B. MAC Layer Emulation

Our modular emulation approach is amenable to a wide range of models for the MAC layer. In this section, we describe the emulated 802.11 channel based on IEEE's 802.11 standard specification that we implemented in MobiNet. For brevity, we leave out details of 802.11b specifications in this paper. 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 [7]. 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 implemented the MAC model along 802.11b's specifications for RTS-CTS-Data-ACK. As in the specification, we can turn off RTS-CTS in our module as not all operational networks use this option. In our current implementation, we maintain a global 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 begins transmitting a new packet, 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. More details of our implementation are described in [8]. If a packet was successfully transmitted, it is then moved to the queue of the next pipe.

Moving packets from one pipe to another in ModelNet operates as follows. We maintain a heap of pipes sorted by earliest deadline, where each pipe's deadline is defined to be the time that the first packet in its queue would complete its traversal of that pipe. The MobiNet scheduler executes once every clock tick (currently 10Khz) and runs at the kernel's highest priority level. The scheduler traverses the heap for all pipe deadlines later than the current time. The descriptor for the packets at the head of the queue for each of these pipes is removed and either: i) moved to the tail of the queue for the next pipe on the packet's path from source to destination, or ii) the packet itself is scheduled for delivery (using the kernels ip output routine) in the case where the packet has reached its destination.

The physical layer plays an important role in the performance and energy consumption of mobile and wireless systems. We implemented the free space model and the two-ray ground reflection model that predict the received power as a deterministic function of distance [1]. 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 in 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, we assume that the packets interfere with one another and drop both.

III.C. Dynamic Routing

As with all other MobiNet modules, we implement the routing layer as a pluggable module in the FreeBSD kernel. 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) [3, 4] 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 [13], DSDV [14], or TORA [11, 12]. 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 modules in the kernel with relative ease.

When a packet enters the MobiNet core, MobiNet queries its per-node route cache to check if a route exists for that source-destination pair. If so, 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 as described in [3, 4]. In our experiments, we allow nodes to buffer up to 50 packets while awaiting routes, though the buffer size is configurable. Once the sender receives a ROUTE REPLY to its ROUTE REQUEST, 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. To ensure that routes stored in the cache are valid, nodes also periodically perform ROUTE MAINTENANCE as specified in the DSR protocol.

Our DSR implementation does not implement all of the optimizations in the DSR specifications and incorporated in ns2. Specifically, in *salvage-with-cache*, a node consults its cache for a route on a transmit failure and salvages the packet using the route from its own cache if possible. The node also marks the packet as salvaged to prevent a single packet being salvaged multiple times. Next, in *use-TAP*, a node can use promiscuous mode listening to overhear packets not intended for its MAC address and thus update its cache with routes from overheard packets. 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. The above optimizations do not impact the correctness of the protocol, although they improve performance, for instance, by lowering the overhead due to control packets.

IV. Evaluation

In this section, we describe our experiences using MobiNet for evaluating ad hoc wireless applications. Our evaluation focuses on testing MobiNet for scalability as well as accuracy.

We have written and tested a simple application using native UDP and in the ns2 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 the native and ns2 versions of the application function identically, its simplicity leads to the test application exhibiting very similar behavior in both environments. Using this application, we execute 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 are to reproduce the experiments described in [1].

In all of our experiments, MobiNet edge nodes consist 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. We conduct our ns2 experiments on machines 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 simulate the random waypoint mobility model using various seeds and pausetime values, resulting in different movement patterns. For most of our experiments, we specify a neighbor-refresh interval of 0.5 seconds.

IV.A. Core Performance

In our first experiment, we set out to quantify the number of packets per second the MobiNet core router could emulate without saturating the core. We ran our experiments on a topology comprising 200 nodes.

We emulate 200 VNs spread across 2 edge nodes. We disable the mobility module to decrease the overhead due to DSR. Thus DSR is invoked only once for a source-destination combination. Once a route to a particular destination is found by the routing module, the route does not change. We associate a sender application with every VN on one edge machine, and a listener application with every VN on the other edge machine. Senders transmit 64-byte UDP packets at a constant bit rate to a specific listener, thereby accounting for 100 flows from the sender edge machine to the lister edge machine. Each source sends packets in exactly 1 hop to exactly one destination which is also the node's sole neighbor. Thus, there are no packet collisions. We also set the DIFS and SIFS values in the 802.11b specifications to zero as the goal is to gauge the maximum number of packets that could be sent through a single MobiNet core. The 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.

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%	89K	50K	35K

Table 1: Forwarding capacity at the Core

We measure throughput in terms of packets per second and CPU utilization at the core for varying packet transmission rates. 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 ensure that there were no collisions and nodes have only their communication partners as their neighbors. In the case where each packet has to traverse only 1 hop before reaching the destination VN, we find that the core is able to process up to 89,000 packets per second. As the number of hops that

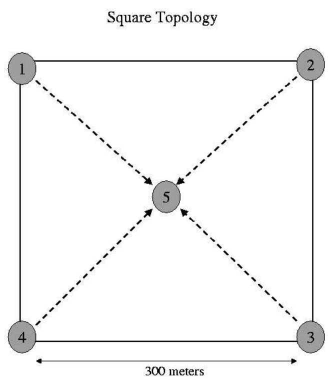

Figure 3: Square topology used for MAC validation. Nodes 1, 2 3 and 4 send packets to node 5

each packet has to traverse increases, we find that the total number of packets that the core can forward per second decreases as the core now has to perform more work per packet. At maximum capacity, where each packet has to traverse 5 hops, the packet processing rate at the core drops to 35,000. We summarize our results in Table 1.

IV.B. 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 802.11b MAC model, we conduct micro-benchmarks to compare MobiNet's MAC layer behavior with that of ns2's MAC implementation 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 choose several topologies and configure nodes to send packets to each other at different rates. Since we are comparing the performance of our MAC model, we turn off DSR and supply MobiNet with precomputed routing tables. Nodes are stationary, hence the routes are valid for the entire duration of the experiment.

Figure 3 depicts one such topology. We place four nodes at the four corners of a square of side

	40 pkts/sec		400 pkts/sec	
Sender	ns	MobiNet	ns	MobiNet
node 1	10000	10000	4573	4634
node 2	10000	10000	4603	4527
node 3	10000	10000	4697	4594
node 4	10000	10000	4635	4674

Table 2: Packets received by central node (node 5) in square topology for ns2 and MobiNet for various sending rates of 40 and 400 pkts/sec

300 meters. We place a fifth node at the center of the square. All the four nodes at the corners of the square send packets to the central node. The transmission range of the nodes is 250 meters. Each corner node sends a total of 10,000 UDP packets to the central node, where each packet is 64 bytes long. We report the total number of packets that the central node receives from each of the senders in both MobiNet and ns2. We present results for packet transmission rates of 400 pkts/sec and 40 pkts/sec. As the packet transmission rate increases, the contention at the central node increases, leading to more packet drops. For lower packet sending rates, contention is not an issue at the receiving node and hence it receives all the packets from all sending nodes. The values shown in Tables 2 are averaged over 3 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.

We run similar experiments for different topologies and verify the accuracy of our MAC implementation. In all cases we find that MobiNet and ns2 had similar packet delivery ratios [8].

IV.C. Routing Accuracy

We validate the emulation accuracy of MobiNet by comparing experimental results obtained from MobiNet to that from ns2 for our simple CBR communication. We use the 802.11b MAC protocol and DSR implementations available in ns2. Using our mobility model, we generate identical movement files for both ns2 and MobiNet.

In our first accuracy experiment, 50 nodes move 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. We choose 10 nodes to be the designated senders in a random fashion. Each sender transmits all of its packets to a fixed randomly chosen destination for the duration

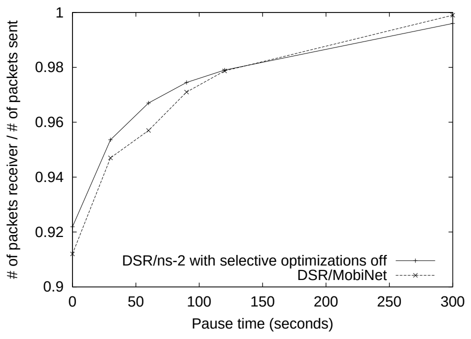

Figure 4: Packet delivery ratio as a function of pause time in ns2 and MobiNet with identical sets of DSR optimizations and maximum speed of 20 m/s.

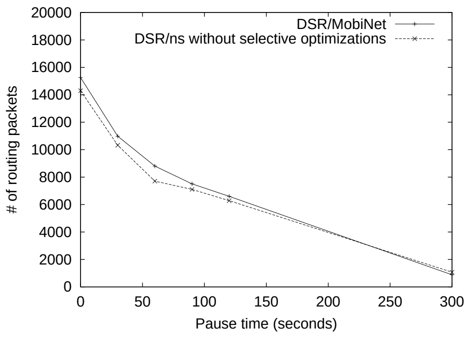

Figure 5: Routing overhead in ns2 and MobiNet with identical sets of DSR optimizations. Maximum speed is 20 m/s

of the experiment. Because our experiment ran for 300 seconds, each CBR sends 1200 packets for a total of 12000 packets sent in aggregate. We measure the packet delivery percentage for the above experiment in both ns2 and MobiNet. We vary the pause times from 0 seconds (high movement) to 300 seconds (no movement).

Since we have yet to implement 3 optimizations of DSR in Mobinet, we compare the performance of Mobinet to that of ns2 with the 3 optimization options turned off. 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 4. To further validate our emulator, we repeat the previous experiments with different maximum speed and find that MobiNet's packet delivery ratio matches that of ns2 [8].

IV.D. Routing Overhead

We conduct further tests to determine the number of control packets transmitted by our implementation of the DSR routing protocol relative to the number of control packets transmitted by the ns2 implementation. While we vary the maximum speed from 1 to 20 m/s, in the interest of space, we present results only for maximum speed of 20 m/s in Figure 5. As in previous experiments, we turn off the 3 DSR optimizations in ns2.

IV.E. Scalability

Given the accuracy of our emulation experiments, we next consider the scalability of our emulation environment. One of the main benefits of MobiNet over using a simulator such as ns2 is that experiments can be run in realtime. Simulators that do not run in realtime have the advantage that independent of experiment complexity, it eventually completes. On the other hand, emulators that run in realtime find the load too great at some stage. However, for typical experiments, a single MobiNet core is capable of forwarding up to 89,000 packets per second and thus has a distinct advantage over ns2 with respect to time taken to complete experiments up to this capacity. The structure of the simulation environment means that simulating a given configuration typically proceeds slower than realtime for the scales that we consider.

To quantify this benefit, we compare the time required to run experiments with varying numbers of senders. We use 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, we distribute 200 VNs across 2 MobiNet edge nodes. We execute ns2 experiments on a single machine with the same configuration as a MobiNet edge node. We disable node mobility in both MobiNet and ns2 to reduce the overhead of finding routes with DSR. Hence, DSR only needs to find routes to destinations once (at the start of the experiment).

We vary 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 sends a total of 1200 packets. Figure 6 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), thus takes 15 minutes of total machine time.

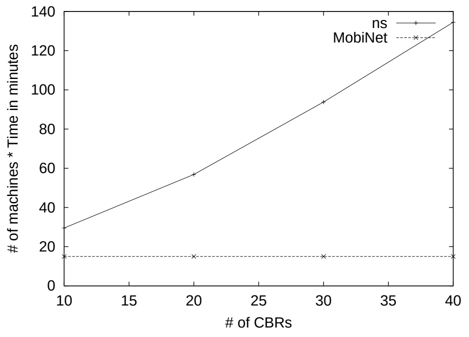

Figure 6: Scalability in MobiNet vs. ns2 as a function of time

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 lasts 134.5 minutes, compared to MobiNet's 15-minute emulation.

V. Deploying Real Applications

In this section we demonstrate the utility and generality of our infrastructure by deploying and evaluating real unmodified code, a video player over MobiNet. We used XAnim, a program that plays a wide variety of animation, audio and video formats. Running the same application on ns2 would be difficult to impossible because of the need to port the application to ns2's environment. Our goal was to study the performance of the the video player in an 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 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 deployed XAnim on two randomly chosen VNs, with one hosting the streaming player and the other hosting the display. Communication between these 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. Once the route was found, MobiNet emulated each packet in a hop-by-hop fashion, subjecting it to the contention, available bandwidth and delay imposed by the MAC layer. Due to node movement, if existing routes went stale, DSR was used to find fresh routes to the destination VN.

We replayed the video in a continuous fashion for

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

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 periodically as a result of 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 and maximum speed. We averaged the results over several runs of the experiment and present them in Table 3. When nodes are stationary (300 seconds pause time), we note that the total number of packets exchanged between the streaming player VN and display VN are around 16100 for various values of maximum speed, while this number falls to 5490 packets when the maximum speed of the nodes is set to 20 m/s and the pause time is 0 seconds.

VI. Related Work

Zhang and Li [19] 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 many deployment scenarios. Noble and Satyanarayanan [9] 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. Netbed [17, 18] is a network testbed at the University of Utah. Netbed provides a testbed of real mobile nodes using real mobile hardware and software. In contrast to our work, Netbed is a real testing environment, not an emulation or simulation infrastructure. Emwin [20] is a network emulator similar to MobiNet. While

Emwin performs MAC emulation, it does not implement ad hoc routing protocols. Emwin also maps emulated topologies as in MobiNet, it does not, however, have separate edge nodes and cores, somewhat limiting the operating system and application scenarios that they may support,

JEmu [2] is another emulation system for mobile and ad hoc networks. Its scalability is limited as each emulated node must run on a dedicated physical machine. SeaWind [6] enables wireless emulation by changing various parameter sets. However, they do not provide a test-environment for ad hoc routing protocols.

Walsh and Sirer [16] have used staged simulation, a technique to improve the scalability and performance of network simulators. They have shown that ns2 tends to scale 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 [7] is a simulator for TinyOS wireless sensor networks. It captures network behavior by using a probabilistic bit error model. TOSSIM cannot capture the real time packet collisions that might take place for a given environment, 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 [5] 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.

VII. Future Work and 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 wireless scenarios, including laptops, and PDAs. Current approaches to such experimentation include simulation and live deployment. 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. It allows researchers to rapidly experiment with a variety of MAC, routing, and communication (layers 2-4) protocols that may not be easily available in live deployments. MobiNet also supports the deployment of different mobility and traffic models. It is a framework for researchers to perform comparative studies of applications under a range of conditions. It enables repeatability of tests in a manner difficult to achieve with live deployment. We further 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] 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.

[3] 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.

[4] 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.

[5] Glenn Judd and Peter Steenkiste. Using Emulation to Understand and Inprove Wireless Networks and Applications. In *Proceedings of NSDI*, May 2005.

[6] 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.

[7] 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.

[8] Priya Mahadevan, Adolfo Rodriguez, David Becker, and Amin Vahdat. MobiNet: A Scalable Emulation Infrastructure for Ad Hoc and Wireless Networks . In *UCSD Technical Report CS2004-0792*, July 2004.

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

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

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

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

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