MoViT: The Mobile Network Virtualized Testbed

Eugenio Giordano*
University of California Los Angeles
Computer Science Department
Los Angeles, California
giordano@cs.ucla.edu

Lara Codeca' †
University of Luxembourg
Interdisciplinary Centre for Security, Reliability and Trust
Luxembourg City, Luxembourg
lara.codeca@uni.lu

Brian Geffon
University of California Los Angeles
Computer Science Department
Los Angeles, California
briangeffon@ucla.edu

Giulio Grassi
University of Bologna
Computer Science Department
Bologna, Italy
grassig@cs.unibo.it

Giovanni Pau
University of California Los Angeles
Computer Science Department
Los Angeles, California
gpau@cs.ucla.edu

Mario Gerla
University of California Los Angeles
Computer Science Department
Los Angeles, California
gerla@cs.ucla.edu

ABSTRACT

MoViT is a distributed software suite for the emulation of mobile wireless networks. MoViT provides researchers and developers with a virtualized environment for developing and testing mobile applications and protocols for any hardware and software platform that can be virtualized. The distributed nature of MoViT allows for the emulation of mobile networks of arbitrary size. Additionally, the network connectivity is shaped transparently such that the connectivity observed by each virtual node resembles that of a physical mobile network. In this paper we present the MoViT architecture, the models used to emulate the wireless channel, the details of our initial implementation and, finally, the results of our evaluation regarding the scalability, realism, and versatility of MoViT . x

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

*This work was supported by Microsoft Research through its PhD Scholarship Programme.

†Lara Codecà is also with the University of Bologna Computer Science Department.

VANET'12, June 25, 2012, Low Wood Bay, Lake District, UK.

Keywords

Framework, virtual testbed, vanet

1. INTRODUCTION

The last few years have seen the emergence of new mobile applications and services including mobile navigation systems, intelligent transportation systems, mobile marketing, mobile multimedia communications, mobile social networks, and participatory sensing.

What all those mobile services have in common is that they are designed for large scale deployments. To support the development and the evaluation of such applications we built MoViT: the Mobile network Virtualized Testbed. Our goal is to enable the development and evaluation of protocols and applications on large scale mobile networks yet retaining hardware and software realism. In MoViT nodes are virtualized and the network characteristics are carefully modeled **in realtime**. Each virtual appliance runs actual applications, on a real operating system and network stack. MoViT replicates all the necessary modeling details (mobility, network parameters, urban constraints, etc.) in order to provide a **realistic** yet **fully controllable** playground on which to develop and evaluate applications for the mobile environment. MoViT captures the system constraints of an actual deployment without the hardware costs, the uncertainties, and time associated with repeated actual experiments. Protocols and applications developed using MoViT can be seamlessly ported to real devices. MoViT is scalable and, according to the number of emulated nodes, it can be run on a small server farm or on a large server cloud. Our long term goal is to provide a scalable tool that can potentially run mobile services and applications to serve a wide and diverse community.

In this paper we describe MoViT's architecture as a general emulation system for mobile networks; we present the implementation details of MoViT including the mechanisms for the mobility management and the channel emulation. In this first implementation of MoViT we modeled the mobility and the channel to reflect those of a Vehicular Ad Hoc Network (VANET). Results show that MoViT accurately reflects real world behavior and can scale to hundreds of emulated nodes.

2. RELATED WORK

In the last years various attempts to build mobile networks testbed and emulators have been made. However, very few large-scale mobile network testbeds have been built to date, mainly because of prohibitive deployment and maintenance costs. Among them, we can mention the commuter bus network DIESEL-Net operated by the University of Massachusetts at Amherst, which has led the field in delay tolerant experiments [9]; the UCLA C-VeT designed for passenger vehicles and streaming applications [11] and the Orbit testbed that features 400 nodes all located in a large room [19] enabling the detailed evaluation of various wireless technologies. However, most testbeds are either still too small in size (few tens of nodes) or do not entail node mobility to yield conclusive results about real world performance of mobile applications and protocols.

To overcome these limitations, a variety of hybrid simulation tools and wireless emulators have been developed which include **real** hardware and which enhance modeling realism with respect to simulation. TWINE [23] and its evolution WHYNET [16] are mobile networks hybrid emulators. They integrate simulation, emulation and real hardware. The TWINE system is designed to aggregate all the approaches, but it doesn't overcome the limits related to actual testbeds. MoViT is designed to be scalable and enable experiments with a large number of nodes. QOMET [7] is a multipurpose wireless emulator able to provide an environment flexible and highly scalable. However, QOMET provides limited mobility modeling and lacks dynamic link layer modeling as the user has to statically define bandwidth limitations, delays and packet loss. MoViT provides realistic channel modeling that is dynamically computed as the experiment is being performed. Another example is Emulab and its mobile extension [15]. Emulab reproduces mobility by physically controlling the movement of real robots in a limited environment. Emulab's approach is more realistic as communications happen on a real wireless channel. However, the kind of scenarios that can be reproduced is extremely limited and bound to the actual testbed hardware. In NRL Mobile Network Emulator (MNE) [17] each node requires a real PC and runs under Linux. The network connectivity is emulated using iptables and the mobility is emulated through a synthetic GPS that provides the nodes with their current position. The system shapes the network connectivity through changes of iptables rules according to propagation and mobility models. MoViT uses virtual machines to emulate the network nodes and thus it is OS independent. Furthermore the mobility is accurately reproduced through a microscopic mobility simulator. Finally, MoViT has the ability to scale to large network sizes with very limited costs as the nodes are virtual machines and not actual PCs.

3. THE EXPERIMENTAL PLATFORM

The general functionality of MoViT is summarized in Figure 1. MoViT is a distributed system that runs on several physical machines called *hosts*. Every host has two physical network interfaces connected to two separated networks: the experimental network and the control network. The experimental network is the one that *emulates* the behavior of a mobile network. Each host runs a set of virtual machines which represent network nodes. Each of these virtual machines is mapped onto one of the nodes participating in the *emulated* network. MoViT reproduces the connectivity of the emulated network in the experimental network by filtering the packet flows between virtual machine pairs. If the emulated network is mobile, its connectivity will change consistently over time and therefore the corresponding connectivity among virtual machines onto the experimental network. To follow these dynamic changes, information regarding the connectivity of the emulated network needs to be distributed to all hosts. The control network facilitates this purpose and, in addition, allows unrestricted access to the virtual machines. Thus, MoViT provides a fully controllable set of virtual nodes whose connectivity is evolving over time according to the connectivity of an emulated network. The timely distribution of connectivity data is managed by the **experiment controller** through the control network. The experiment controller is a collection of centralized and distributed software components that in addition to distributing connectivity data also manages the execution of experiments. In the following sections we describe in detail each component of MoViT .

3.1 Experiment Controller

The experiment controller manages all functionalities necessary for the execution of an experiment. These functionalities include the instantiation of virtual machines, the initialization of experimental applications, the distribution of mobility data and the collection of experimental results.

The experiment controller is in charge of generating the mobility data, contextualizing it to the environment where the experiment is run and redistributing it among the hosts. In particular, the experiment controller will produce subsequent snapshots of mobility taken at periodic intervals. Mobility is either simulated or reproduced from a pre-recorded trace, not contextualized to a specific period of time. Given the distributed nature of MoViT, sufficient time between the generation of a mobility snapshot and its actual emulation on the experimental network must be allowed for the data to be transferred to all participating hosts. Thus, each mobility snapshot is assigned an *application time* (in the future) in which the network connectivity corresponding to the mobility snapshot will be *emulated* on the experimental network. For example, let us assume that the mobility consists of snapshots of mobility taken at each second. The first snapshot will be assigned an *application time* that corresponds to the time at which the experiment will start; all following snapshots will be assigned increasing *application times* one second apart. As previously discussed, each MoViT virtual machine is a node in the emulated network. The experiment controller keeps track of the correspondence between the IDs of emulated nodes and the IDs of the virtual machines, ensuring consistency throughout the experiments.

A MoViT experiment is generally broken into three parts: instantiation, the actual experiment, and the collection and evaluation of results. To perform these tasks we chose to use the well known Orbit Control and Management Framework (OMF), as it has already been proven stable and very versatile [1]. OMF is a very portable framework, the modifications to OMF necessary to integrate with MoViT are minor. The initialization of experiments consists of: an assessment of the resources required by the experiment (hosts and virtual machines), the instantiation of those resources and, finally, the bootstrap of the distributed emulation software. OMF, through the control network, executes experimental applications on virtual machines and, finally, collects the results at the end of the experiment. OMF-enabled application offer the possibility of collecting experimental results as the experiment is being performed, enabling the visualization of live charts.

3.2 Host Configuration

On each *host* runs a set of Virtual Machines (VMs) that represent network nodes. The VMs could be run using any virtualization technique, in our implementation we use the Kernel-based Virtual Machine (KVM) for Linux solution [4]. Figure 2 presents the architecture of each host software configuration. The inter-

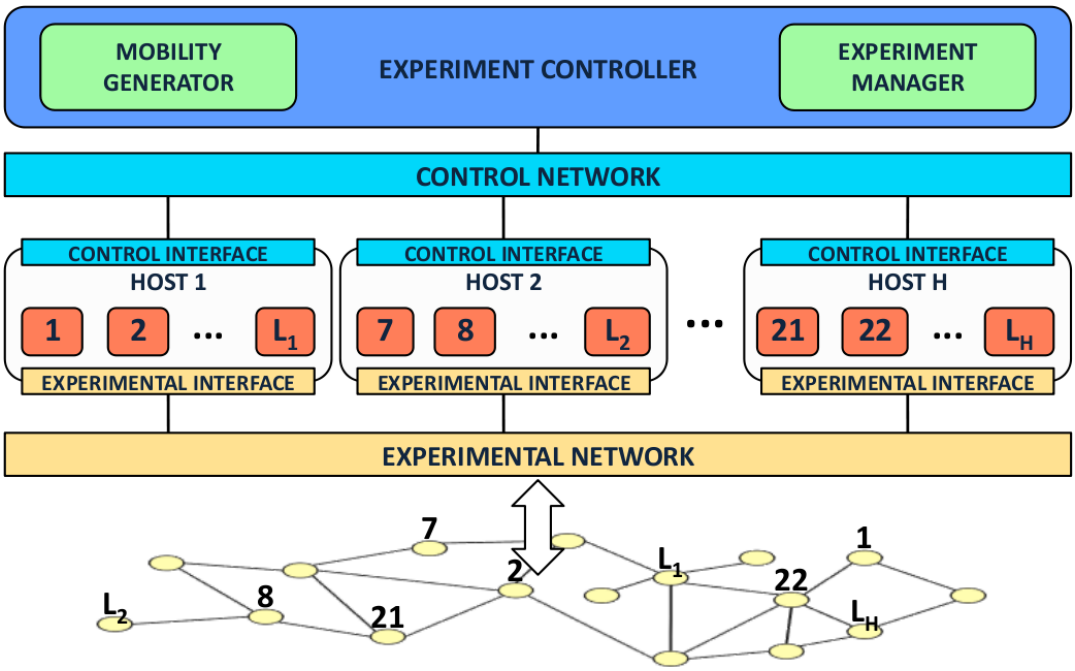


Figure 1: MoViT Functionality Block Diagram.

nal network configuration between Hosts and VMs is similar to the one explained in [1] due to the usage of OMF. On each host the **Connectivity Manager** shapes the connectivity among VMs. The shaping of connectivity is performed by a specific component called **Network Shaper** that filters packets between VMs applying drop rates and delays according to the connectivity rules generated by the **Channel Module**. The channel module generates connectivity rules based on the mobility data generated by the experiment controller.

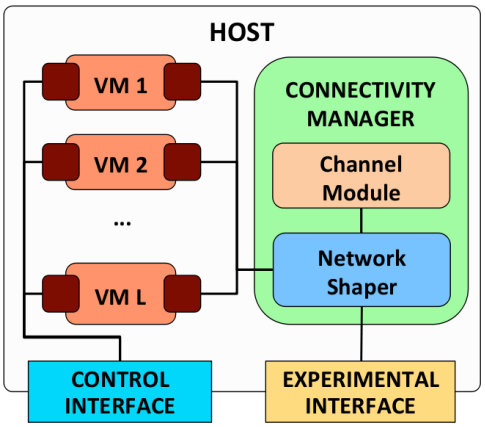


Figure 2: Host Functionality Block Diagram.

3.3 Connectivity Manager

The connectivity manager is in charge of shaping the network connectivity among the VMs. This reshaping is possible through the filtering of packet flows between all source and destination pairs. This filtering is made possible by applying artificial drop rates and additional delays (i.e. connectivity rules) on a *per packet* basis. The drop rate and delay need to be *dynamically* modified according to the network topology that needs to be reproduced. To allow the emulation of several channel models, we decided to split the connectivity manager into two separate components: a channel module and a network shaper. The channel module is in charge of producing the connectivity rules related to the mobility and the Network Shaper handles the reshaping of the network through the filtering of packet flows.

3.3.1 Channel Module

In order to shape the network connectivity among the network nodes (i.e. virtual machines), connectivity rules need to be present on each *host* participating in the experiment. The connectivity rules consist of drop rates and additional delays to be applied to packet flows between each source and destination pair. The drop rate and delay experienced on a wireless link between two nodes depend on the path loss, the number of nodes contending for the channel and on interference. External interference cannot be directly computed and needs to be approximated through the use of statistical models. Conversely, path loss, co-channel interference and channel contention can be directly inferred from the mobility of nodes with regards to the environment they are moving in.

MoViT decentralizes the computation of the connectivity rules by running an instance of the channel module onto each host. To achieve this decentralization, the mobility data needs to be distributed to all hosts. The optimal solution for the distribution of data to multiple hosts would be a multicast tree. However, network layer multicast is not supported on all network technologies. Therefore, we decided to implement an application layer multicast tree among the channel modules running on the hosts. The mobility data generated by the experiment controller is transferred only to the first level of the tree. At each level of the tree all channel modules will forward the data to the underlying level, until all channel

modules are reached. A detailed evaluation of the performance of the tree structure is reported in Section 5.1.1.

3.3.2 *Network Shaper*

Overview. To the best of our knowledge there is no existing tool that implements all the features needed to efficiently reproduce the connectivity of a dynamically changing network topology. Hence, we need a new tool able to enforce *per packet* and *dynamic* filtering of packet flows at the data-link layer and that can perform the change of large amounts of connectivity rules efficiently. We chose to replicate the structure of the well known NETwork Emulation tool for linux, netem. We implemented a kernel module that can be associated to a Queueing Discipline (QDisc) [6]. QDiscs allow the module to enforce filtering policies inside the kernel, between device drivers and the network stack. Figure 3 presents the architecture of the proposed filtering solution which we will refer to as the Network Shaper. The kernel module is associated with a user-space application, called *Rule Switcher*, used to retrieve the connectivity rules from the channel module.

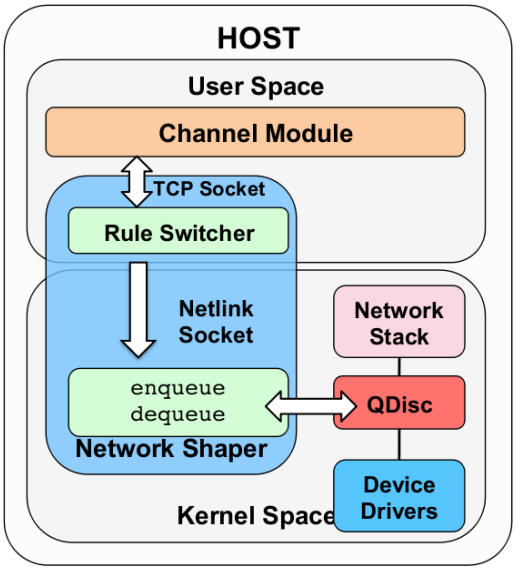


Figure 3: Architecture of Network Shaper and interaction with channel module.

Filtering Solution. Shaping the network requires a system architecture that is able to take control of all the network traffic generated by the VMs on the experimental interface. This kind of granularity can be achieved through the use of the QDisc system implemented in the Linux kernel. By registering Network Shaper as a queueing policy onto the QDiscs that manage the experimental interfaces of all VMs, we force all network traffic generated and incoming to be processed by it. This placement of the module allows the filtering to happen at the lowest possible layer right before the packets are actually passed to the physical interface. In addition, as detailed in the next paragraphs, the use of QDiscs allows MoViT to enforce a filtering policy that is transparent from inside the VMs, providing a reduced connectivity, reproducing what would be experienced by a real node in a real network.

Figure 4 summarizes the flow of packets between VMs on the same host and on different hosts. All the virtual interfaces and

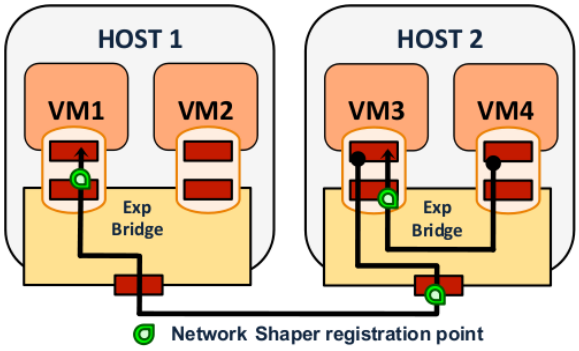


Figure 4: Packet flow among virtual machines running on the same host or on different hosts.

the physical experimental interface are linked through the experimental bridge. Each network interface in the host, either virtual or physical, has its own QDisc system linked to both its incoming and outgoing packet buffers. Network Shaper is registered to the outgoing buffer for the following reasons: filtering outgoing packets minimizes the amount of useless traffic traversing the network; the VMs perspective of the network is consistent and coherent with the experienced connectivity; the filtering of broadcast packets can still be performed at the destination.

Packet processing. The QDisc system calls the Network Shaper in two separate cases: when a packet is generated by the network stack and when the device driver is ready to send a packet.These two cases are handled by the Network Shaper through two separate functions, `enqueue` and `dequeue` respectively. The `enqueue` function receives a packet from the network stack and according to the connectivity rules it will decide if it has to be dropped or for how long it needs to be delayed. If the packet is not dropped, it is assigned a *time_ to_send* based on the assigned delay. The packet will then be stored in a the Network Shaper's internal priority queue, for which the priority is determined by the *time_ to_send*. When the `dequeue` function is called, the Network Shaper will determine whether there is any packet in the queue for which the *time_ to_send* timeout has expired. In such case it will return the packet to the QDisc system to be sent to the device driver. This simple queuing solution allows for accurate delays without the use of timers. Figure 4 shows the different points (green tear drops) at which the Network Shaper is registered in the network; following the flow of packets, we observe that in some cases the same packet can traverse the Network Shaper more than once. Consequently, the `enqueue` function must determine at which point of the network the packet is being processed. Together with the broadcast and unicast classification, in Table 1 we define four classes of packets that will be processed differently.

	External	**Internal**
Broadcast	No delay applied. Further processing at the destination	Delayed or dropped according to connectivity rule
Unicast	Delayed or dropped according to connectivity rule	Double rule check at source and destination [1].

Table 1: Packets classification inside the Network Shaper.

The reshaping of the network requires processing each single packet traversing the network. To avoid the Network Shaper becoming the bottleneck of the network, the processing of each packet must be as fast as possible. Hence, the number of operations needed to *retrieve* and *apply* a connectivity rule (drop rate and delay) must be minimized.

Rule Retrieval. The rules for a given VM interface are retrieved using an index. The selection of index always implies a trade-off between memory usage and access time. For this, the best choice is a hash table. To avoid the memory waste due to the usage of generic hash tables, the rules are stored using consistent hashing. For simplicity, since we have control on the VM ethernet interfaces, we assign the virtual machines MAC addresses and we use the last four bytes if it as index for the connectivity rule table.

Rule Application. To reduce the rule application time, we also introduce a tabulation of the drop rate and delays to be applied. To keep the Network Shaper's filtering policy independent from the channel model we want to reproduce, these tables are an input of the Network Shaper together with the connectivity rules. In order to account for the different behavior of broadcast and unicast packets, the Network Shaper takes as an input a double set of tables.

Drop rate, delay average and delay variance are stored in a bi-dimensional table for which the row index represents the connectivity status (e.g. the path loss) and the column index represents the packet length. A *Connectivity Rule*, between source and destination, is then composed of a set of three table row indexes representing the related drop rate, delay average and delay variance. Delegating the definition of the drop, delay average and variance to the channel module allows the Network Shaper to be completely independent from the channel model used to reproduce the connectivity. In order to provide the possibility to account for fast fading, an additional random process can be taken as input and added to the drop rate row index; in our experiments, for example, this random process was a Rayleigh distributed random variable [18].

The application of a connectivity rule involves only a series of memory lookups making it a very fast and efficient process.

Rule Set Switch. Each mobility snapshot is assigned an *application time* that defines the time when the resulting connectivity rules set must be transferred simultaneously to all the Network Shaper instances. This synchronization (in the order of milliseconds) is achieved through the use of the Network Time Protocol (NTP) [5], so that the transfer of the connectivity rules set is triggered on all hosts approximately at the same time. The switch between two subsequent set of rules must be a very efficient process which execution time is as much as possible independent on external factors (e.g. machine or network load). In additionthe transition time from a rule set to another is negligible with regards to network delays (in the order of microseconds). For these reasons the rule set switch process is performed by a specific application, **Rule Switcher**, running in user space, that obtains the connectivity rules from the channel module and passes them to the Network Shaper. This solution allows the implementation of the rule switcher to be independent from the channel module, allowing the use of different propagation models if needed.

To perform the transfer of the rules to the Network Shaper, we use a standard communication method between user and kernel space: a Netlink socket. As the size of the "emulated" network increases the size of each rule set increases as well. In order to provide scalability and reduce the load on the network stack, only the location (a pointer to memory) and size of the rules set are transferred to the Network Shaper that will perform a simple unbuffered copy of the rules.

[1]This double check also allows the Network Shaper to keep out of the VM interfaces network traffic that is not generated on the experimental network.

4. MoViT FOR VANET

Thus far we have described the functionality of MoViT as a general emulation system. The communication among all the previously described software modules follows a well defined protocol, allowing for the substitution of each of them. This allows for the use of different channel models on different network interfaces, enabling the emulation of hybrid networks in which nodes have multiple wireless interfaces (e.g. WiFi and cellular connectivity).

We focused our implementation of MoViT on the emulation of Vehicular Ad hoc Networks (VANETs). However, MoViT's architecture can be used to emulate any kind of mobile network.

4.1 Mobility Generation

Mobility is the main contributor of the network topology changes in vehicular networks. Mobility determines the topology characteristics and most of the topology dynamics that are peculiar of vehicular networks. MoViT offers the possibility to reproduce mobility starting from recorded vehicular traces, or to simulate it in real time from predefined end to end requirements using the Simulation of Urban MObility simulator (SUMO) [13]. SUMO allows the simulation of thousands of vehicles in real time. In addition, SUMO allows the dynamic control of the simulation through the Traffic Control Interface (TraCI) [21]. TraCI provides control of simulated vehicles as the simulation is being performed. This detailed control enables the evaluation of a multitude of applications related to vehicular safety or Intelligent Transportation Systems (ITS) such as alert distribution or closed-loop traffic optimization.

4.2 Connectivity Modeling

The Network Shaper allows the application of drop rates and additional delays to packet flows according to a set of connectivity rules. As described in section 3.3 the connectivity rules consist of row indexes of tables. The tables contain the actual values of drop rate and additional delay to be applied. In the following paragraphs we describe in detail the models we employed for the reproduction of the behavior of a VANET.

Packer Error Rate. The Packet Error Rate (PER) in wireless scenarios is a function of the Bit Error Rate (BER) and of the length of the packet and can be expressed by the following expression:

$$PER = 1 - (1 - BER)^{N_b} \quad (1)$$

where N_b is the length of the packet in bits. The BER for IEEE802.11 networks can be computed with the following expression, as proposed in [3]:

$$BER = Q(\sqrt{11 - SNR}) \quad (2)$$

where *SNR* represents the Signal to Noise Ratio. The SNR that is derived from the signal path attenuation computed using the CORNER propagation model [14]. CORNER is a realistic propagation model for urban scenarios that takes into account the presence of propagation obstacles in the surrounding environment. We use the *PER* of equation 1 for broadcast packets, instead for unicast packets we compute a separate table. Indeed, unicast packets in IEEE802.11 employ the use of ACK and subsequent retransmissions in case of failure, therefore the drop rate is computed as a

function of the maximum number of allowed retransmissions. The PER for unicast packets can be approximated by the following expression:

$$PER_{unicast} = 1 - \sum_{i=0}^{r_{max}-1} (PER)^i (1 - PER) \quad (3)$$

where r_{max} is the maximum number of retransmission specified by the IEEE802.11 standard.

Packet Delay. In wireless networks, the delay experienced by each packet is the sum of two components, the transmission delay d_t and the channel delay d_c:

$$d = d_t + d_c \quad (4)$$

The transmission delay d_t can be deterministically computed given the transmission rate. The channel delay d_c however can only be statistically modeled. We approximate the distribution of d_c to a normal distribution:

$$d_c = N\left(\overline{d_c}, \frac{\overline{d_c}}{2}\right) \quad (5)$$

where $\overline{d_c}$ is the average of the channel delay. As previously mentioned, in IEEE802.11 unicast packets can be retransmitted, whereas broadcast packets can not. For this reason the $\overline{d_c}$ has a different expression according to the kind of packet:

$$\begin{aligned} \overline{d_c} &= \overline{d_a}, && \text{broadcast} \\ \overline{d_c} &= \overline{d_a} + \overline{d_r}, && \text{unicast} \end{aligned}$$

where d_a is the access delay, and d_r is the retransmission delay.

d_a represents the delay experienced by a node to perform the random access to the wireless channel. To approximate this delay we use Bianchi's model [8], that is one of the most credited and accurate found in literature:

$$\overline{d_a} = \sigma \frac{1 - P_{TX}}{P_S P_{TX}} + T_C \left(\frac{1}{P_S} - 1\right) \quad (6)$$

$$P_{TX} = 1 - (1 - \tau)^n \quad (7)$$

$$P_S = \frac{n\tau(1-\tau)^{n-1}}{1 - (1-\tau)^n} \quad (8)$$

$$T_C = \frac{\text{packet length}}{\gamma} \quad (9)$$

where τ is a parameter that represents the portion of time in which the channel is busy, σ is defined bi the standard (DIFS = 50 μs) [22] and n is the number of nodes contending for the channel at the time of transmission, that we approximate to the number of neighbors computed using the CORNER model. Then according to Bianchi's model d_a is a function of τ, the number of neighbors and the packet length.

The average retransmission delay for unicast packets is a function of the PER and therefore of the SNR. $\overline{d_r}$ can be represented as follows:

$$\overline{d_r} = \sum_{r=1}^{r_{max}} (PER)^r d_t \quad (10)$$

5. EXPERIMENTAL EVALUATION

In this section we present the results of our evaluation campaign aimed at highlighting the scalability, realism and versatility of the proposed system.

5.1 Scalability

5.1.1 Mobility data distribution

In order to assess the scalability of the mobility data distribution architecture described in Section 3.3.1 we performed a large scale experiment on the Amazon Elastic Compute Cloud (EC2) [20]. We employed 101 server instances, one running the experiment controller and the remaining one hundred running the channel module. As described in Section 3.3.1, the channel modules are organized in a structured tree in order to ensure the timely distribution of the mobility data. With this experiment we investigate the delay in the distribution of each mobility snapshot for three different tree configurations, reported in Table 5.1.1 . Each tree configuration is obtained by fixing the maximum number of possible children in the tree for each channel module. A higher number of children results in a higher overhead for the channel module, but also in a smaller delay in the overall distribution of mobility data.

Number of hosts	Max Number of children	Number of tree levels
100	2	6
100	3	4
100	10	2

Table 2: Large scale cloud experiment: tree configuration.

For this experiment we simulated a 4×4 Km urban area using SUMO, with 700 active vehicles at all times. It is remarkable that SUMO was able to simulate one second of mobility on average in 8ms. In our implementation, each vehicle is represented by an integer identification number and two double precision floating point representing latitude and longitude, for a total of 20 bytes per vehicle. The total size of a mobility snapshot is then approximately 14KB. We then measure the delays between the generation of a mobility snapshot and the time in which the connectivity rules have been computed by the propagation module and, therefore, ready to be applied. For simplicity, Figure 5 shows the cumulative distribution cut at the 99^{th} percentile. The maximum delays measured for 2, 3 and 10 maximum children configurations are 960, 970 and 820 milliseconds respectively (with an occurrence rate of $\sim 10^{-5}$). We observe that for all three configurations more than 99% of the mobility snapshots are received within 300ms. However, the configurations with 3 and 10 number of children obtain better results as 90% of the delays are below 50ms. Considering the additional load on each propagation module caused by the service of a higher number of children, we can select the 3 maximum children configuration as the best trade-off. With this configuration none of the measured delays was above 1 second. In Section 3.1 we discussed the requirement of allowing a certain time period between the generation of a mobility snapshot and its actual *application time* on the emulated network. These results provide us with the lower bound of 1 second for this requirement. Such a short interval between the generation of mobility and its actual emulation enables a wide range of applications that require the closed-loop interaction between network nodes and the mobility simulator. An example of such applications is vehicular traffic flow optimization, for which vehicles need to be rerouted reacting to some sensed or advertised event, such as accidents or traffic jams.

5.1.2 Network Shaper

The introduction of a kernel module that processes all packets traversing the system inevitably introduces some overhead. In this

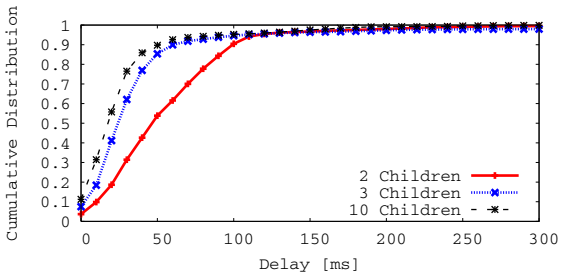


Figure 5: Cumulative distribution function of the mobility data distribution delay for different tree configurations.

section we investigate the amount of overhead introduced by the Network Shaper . We performed an experiment employing a single host machine running up to 50 virtual machines at the same time. We then periodically generate bursts of 5 broadcast packets and measure the processing delay introduced by the kernel module. It is important to understand that a single broadcast packet, in such an environment, is transformed to multiple duplicate packets, resulting in many simultaneous calls of the kernel module `enqueue` function. We observe that the average processing delay is in the order of tens on nanoseconds, independent of the number of VMs running on the host. We observe a considerable spike in the maximum processing delay when the host is running between 20 and 30 VMs. This spike is due to the bursty overload of the host system which has to handle a large amount of events at the same time. Nonetheless, the maximum processing time, always smaller than 10μs, is still negligible with regards to regular packet processing delays which are in the order of milliseconds.

Remarks. We successfully tested the emulation of up to 50 VMs on a single host machine and showed that our filtering solution introduces a negligible overhead; we showed the scalability of the emulation data distribution up to a 100 host machines. These results suggest that MoViT could easily scale to several hundreds of emulated nodes. The scalability is then bound to the hardware availability.

5.2 Realism

To assess the level of realism that is obtainable using MoViT, we performed a small scale experiment with real nodes and real cars and we then reproduced it with MoViT. We placed four fixed nodes at the four corners of the engineering school building on our campus. The resulting rectangle is approximately 60 meters wide (East-West) and approximately 50 meters in length (North-South). Each fixed node is in line of sight with the nodes placed at neighboring corners and therefore connected to them. The building blocks diagonal connections, and thus nodes placed at opposite corners cannot directly communicate with each other. The fixed nodes antennas are placed on top of a 1.5 meter stand to ensure good signal propagation. In addition we had one mobile node, placed on vehicle that drove around the perimeter of the building. The mobile's node antenna is placed on the rooftop of the car to avoid interference due to the car's metal frame. All nodes are equipped with a IEEE802.11b/g wireless card, a 8 dB gain omnidirectional antenna and a GPS sensor.

Each node runs a mactrace tool that periodically (every 250 ms) broadcasts hello messages, containing position and timestamp information. These hello messages are generated directly at layer 2 using Unix raw sockets. This solution avoids the intrinsic delays introduced by higher network layers (e.g. IP and UDP). Each node keeps a table storing its current neighbors together along with the time the last packet was received from that neighbor. Upon receiving a mactrace hello message, each node will update its neighbor table (i.e. add the message sender to the table if it was not present before or update the time this neighbor was last seen). Gathering the mactrace logs from all the nodes, we are able to construct the network connectivity matrix over time for further investigations. Each node is running the latest available release (0.6.0) of the `olsrd` daemon for Linux that implements the well known Optimized Link State Routing (OLSR) [12]. We set the parameters for OLSR as reported in table 3.

Hello message interval	500ms
Hello message validaty time	1secs
Topology control (TC) message interval	1sec
Topology control (TC) message validity	2sec

Table 3: OLSR Parameters.

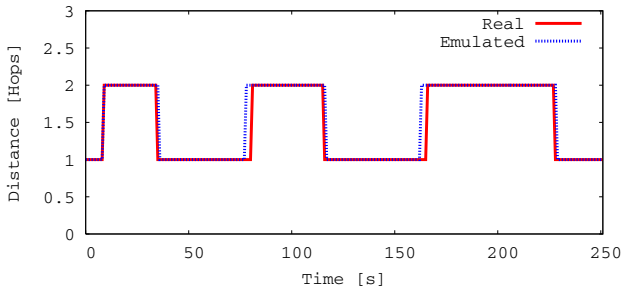
We evaluated the performance of a UDP constant bit rate transfer from the mobile node to one of the fixed nodes. The UDP traffic consisted of 20 bytes packets generated every 100 ms. Using the logs from the mactrace tool, we can reconstruct the network topology matrix over time. With this matrix, we can infer, at any given time, if any two nodes are connected. By running the Dijkstra shortest path algorithm, we can obtain the *optimal hop count* which is defined as the shortest hop count from the mobile sender to the fixed receiver.

Using the GPS traces recorded during the real test we reproduced the same experiment using MoViT. Since network nodes in MoViT are virtual machines, we were able to use the same software configuration used in the real test; we used the same mactrace application, the same version of the `olsrd` daemon and the same UDP traffic generator.

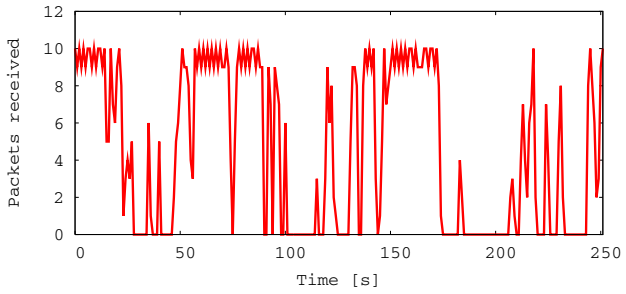
Figure 6 shows the results of this experiments. In particular Figure 6(a) shows the *optimal hop count* as a function of time for both the real world experiment and for the emulated experiment. We observe that the hop distance between sender and receiver alternates between 1 and 2 hops, as expected. In addition, we can observe that the real and emulated *optimal hop count* match almost exactly.

Figure 6(b) shows the number of packets received per second by the fixed receiver as a function of time. We can observe the alternation of periods of good throughput with periods of low and unstable throughput. One would expect the throughput to collapse at the moment the topology changes and then pick up as soon as the routing recognizes the topology change. However, we observe that as long as the nodes are 2 hops apart the throughput is low and unstable.This surprising result has a simple explanation. The `olsrd` daemon implements the routing by modifying the routing table of the operating system (OS), adding the one hop neighbors as default gateways for all destinations. This solution is suitable for static or slowly evolving topologies and unsuitable for fast evolving topologies such as those of VANETs. In fact, the loss of one `hello` packet may potentially result in a change in the routing table. Such frequent changes in the OS routing table often will cause the loss of packets that are already in the buffer.

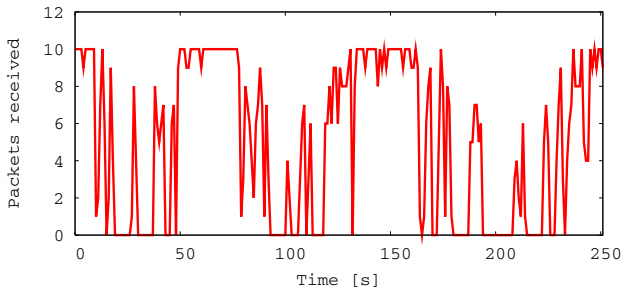
Figure 6(c) shows the UDP throughput as a function of time for the test repeated on MoViT. We can observe that the results obtained with MoViT are consistent with the results obtained in the real tests, as expected. The results do not match exactly due to the



(a) *Optimal hop count* for real and emulated experiments.



(b) UDP throughput for the real expriment.



(c) UDP throughput for the emulated expriment.

Figure 6: Comparison between real and emulated experiment.

impossibility of reproducing external interference, nonetheless the general behavior is correctly reproduced.

5.3 Versatility

This section aims at presenting the versatility of MoViT in terms of protocols and applications that can be developed and tested on it. We show the results of two experiments that we believe to be representative examples.

As a first example we picked the emulation of a Vehicular Ad Hoc Network (VANET). We generated the vehicular mobility using SUMO in a 800×800 meters square of a residential area of a large US city. The mobility is composed of 100 vehicles which remain in the area at all times. To achieve this, when a vehicle hits the boundary it is rerouted to a random destination on the boundary. We activate an increasing number of network nodes, up to 64, onto corresponding VMs. The VMs are evenly distributed on two host servers. An additional server is used for the experiment controller, for a total number of three servers. We tested the behavior of a simple content distribution application: a randomly picked vehicle initiates the periodic, every 200ms, broadcast of a 1072 bytes UDP packet which is again periodically re-broadcasted by all vehicles that receive it. Figure 7 show the average portion of vehicles that received the content after one minute as a function of network size, with error bars representing minimum and maximum. We observe that for a network size of 32 nodes we obtain full distribution on all runs. Regardless of the actual result, this experiment shows that with as little as three servers we can emulate a 64 node VANET, evaluating the functionality of an application that can be directly ported to real machines.

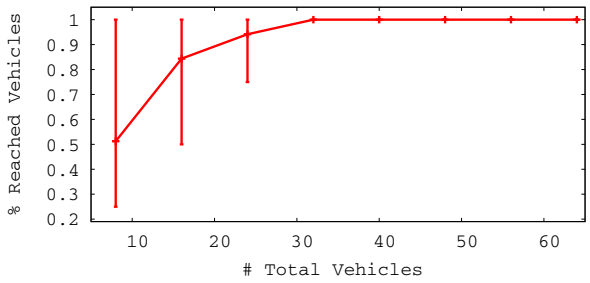


Figure 7: Content distribution evaluation: portion of vehicles with content after one minute.

As a second example we emulated the connection of a wireless device to a fixed wireless router. We emulate the mobile node moving alternatively closer and further from the router. Therefore, the wireless signal strength periodically fluctuates, we fixed this period to be 120 seconds. The wireless mobile node initiates the download of a YouTube [10] video using the Video LAN Client (VLC) [2]. Figure 8 presents the data throughput measured by VLC as a function of time. We observe that the fluctuations of the signal strength result in corresponding fluctuation of the throughput. Although the results of this experiment are noteworthy, the most unique aspect of this experiment is the fact that we were able to use a real world application to interact with an Internet resource.

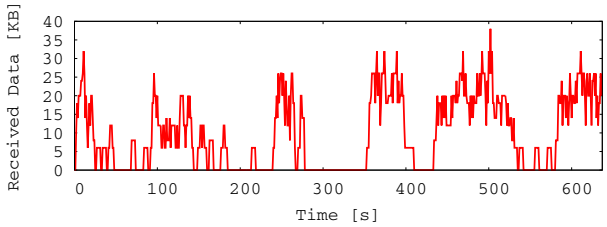


Figure 8: VLC throughput for the download of a YouTube video through an unstable wireless link.

Remarks. We showed that MoViT can easily emulate large networks while requiring only a small amount of resources. In addition, MoViT's emulated nodes can use the real world implementation of applications without having to model their behavior. Finally, MoViT enables the interaction with resources on the internet opening the door for the test and development of a wide range of applications and protocols.

6. CONCLUSIONS

In this paper we introduced MoViT, a distributed software platform for the emulation of mobile wireless networks. Through the

use of virtual machines MoViT provides the user with a sand-boxed environment in which nearly any hardware or software platform can be reproduced. MoViT emulates the connectivity of a mobile network by shaping the network among the virtual machines according to prerecorded or simulated mobility. Our experimental evaluation has shown that MoViT is highly scalable and is in-line with reality. We showed that MoViT allows for the development and testing of real network applications and protocols and for the interaction with real world resources located anywhere on the Internet. Finally, we showed how slight modifications of the MoViT architecture allow for the emulation of virtually any kind of mobile network.

The VANET implementation of MoViT will be made accessible as an on-demand service, allowing researchers around the world to run large scale virtualized mobile experiments. In addition, MoViT's software will be made available to the research community, therefore, in the future, several instances of MoViT could be interconnected through the internet leading to a global scale mobile network testbed.

7. REFERENCES

[1] Orbit management framework. `http://omf.mytestbed.net`.

[2] VideoLAN - Official page for VLC media player, the Open Source video framework. `http://www.videolan.org/vlc/`.

[3] IEEE Recommended Practice for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements Part 15.2: Coexistence of Wireless Personal Area Networks With Other Wireless Devices Operating in Unlicensed Frequency Bands. *IEEE Std 802.15.2-2003*, pages 1 –115, 2003.

[4] KVM - Kernel-based Virtual Machine. `http://www.linux-kvm.org`, Last Accessed in July 2010.

[5] NTP: The Network Time Protocol. `http://www.ntp.org/`, Last Accessed in July 2010.

[6] C. Benvenuti. *Understanding Linux network internals*. O'Reilly Media, Inc, 2005.

[7] R. Beuran, J. Nakata, T. Okada, L. T. Nguyen, Y. Tan, and Y. Shinoda. A multi-purpose wireless network emulator: Qomet. In *Advanced Information Networking and Applications - Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 223 –228, march 2008.

[8] G. Bianchi. Performance analysis of the IEEE 802.11 distributed coordination function. *IEEE Journal on selected areas in communications*, 18(3):535–547, 2000.

[9] J. Burgess, B. Gallagher, D. Jensen, and B. Levine. Maxprop: Routing for vehicle-based disruption-tolerant networks. In *Proc. IEEE Infocom*, 2006.

[10] J. Burgess and J. Green. YouTube: Online video and participatory culture. 2010.

[11] M. Cesana, L. Fratta, M. Gerla, E. Giordano, and G. Pau. C-VeT the UCLA campus vehicular testbed: Integration of VANET and Mesh networks. In *Wireless Conference (EW), 2010 European*, pages 689–695. IEEE, 2010.

[12] T. Clausen and P. Jacquet. OLSR RFC3626, Oct. 2003. `http://ietf.org/rfc/rfc3626.txt`.

[13] D. Z. für Luft-und Raumfahrt e.V. (DLR). SUMO - Simulation of Urban MObility. `http://sumo.sourceforge.net`, Last Accessed in July 2010.

[14] E. Giordano, R. Frank, G. Pau, and M. Gerla. CORNER: a Step Towards Realistic Simulations for VANET. In *The Seventh ACM International Workshop on VehiculAr Inter-NETworking (VANET 2010)*, 2010.

[15] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, ATC'08, pages 113–128, Berkeley, CA, USA, 2008. USENIX Association.

[16] Z. Ji, M. Marina, M. Varshney, Z. Xu, Y. Yang, J. Zhou, and R. Bagrodia. Whynet: A hybrid testbed for large-scale, heterogeneous and adaptive wireless networks. *UCLA Computer Science Department Technical Report CSD-TR060002, Tech. Rep*, 2006.

[17] J. Macker, W. Chao, and J. Weston. A low-cost, ip-based mobile network emulator (mne). In *Military Communications Conference, 2003. MILCOM 2003. IEEE*, volume 1, pages 481–486. IEEE, 2004.

[18] T. Rappaport. *Wireless communications*. Prentice Hall PTR Upper Saddle River, NJ, 2002.

[19] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu, and M. Singh. Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1664–1669. IEEE, 2005.

[20] A. W. Services. Elastic Compute Cloud (Amazon EC2), Last accessed March 2011. `http://aws.amazon.com/ec2/`.

[21] A. Wegener, M. Piórkowski, M. Raya, H. Hellbrück, S. Fischer, and J.-P. Hubaux. TraCI: an interface for coupling road traffic and network simulators. In *CNS '08: Proceedings of the 11th communications and networking simulation symposium*, pages 155–163, New York, NY, USA, 2008. ACM.

[22] I. . working group. IEEE 802.11, part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. ANSI/IEEE Std 802.11, 1999 Edition (R2007).

[23] J. Zhou, Z. Ji, and R. Bagrodia. TWINE: A hybrid emulation testbed for wireless networks and applications. In *IEEE INFOCOM*, pages 23–29. Citeseer, 2006.