JiST/MobNet: Combined Simulation, Emulation, and Real-world Testbed for Ad hoc Networks

Tronje Krop, Michael Bredel, Matthias Hollick, and Ralf Steinmetz
Multimedia Communications Lab (KOM), Technische Universität Darmstadt,
Department of Electrical Engineering and Information Technology & Department of Computer Science,
Merckstrasse 25, 64283 Darmstadt, Germany
{firstname.lastname}@kom.tu-darmstadt.de

ABSTRACT

Mobile ad hoc networks are a promising technology enabling the spontaneous formation of communication networks without dedicated infrastructure. However, ad hoc networks are not yet ready for large-scale deployment, because several unsolved research challenges persist. Evaluation methods such as analytical modeling, simulation, emulation, and real-world experiments aid in addressing these challenges. Applying a single method alone often leaves doubt as to the accuracy of the obtained results. Thus, there is a strong need for tools that support the task of modeling, evaluation, and analysis. These tools should allow for protocol validation, performance analysis, and proof-of-concept implementation using multiple evaluation methods. In this paper, we present a new approach for conducting simulation, emulation, and real-world experiments in mobile ad hoc networks using a single tool that allows for vertical validation of experiments. We explain the principles behind the architecture of our tool and systematically assess its limitations.

Categories and Subject Descriptors: C.2.1 [Network Architecture and Design Network]: Architecture and Design – *Wireless communication*

General Terms: Design, Measurement, Performance

Keywords: Mobile ad hoc networks, simulation, emulation

1. INTRODUCTION

Mobile ad hoc networks (MANETs) – envisioned as next generation networks – are a promising technology allowing spontaneous formation of communication networks without dedicated infrastructure. However, MANETs are not ready for large-scale deployment. The complexity arising from mobile devices – which communicate in an ad hoc fashion via a shared, unmanaged wireless medium – poses a variety of research challenges, many of which are as yet unsolved. Key research areas include service discovery, network addressing, dependable routing, distributed media access, and radio interference. The development of efficient protocols and distributed algorithms is further complicated by cross-layer interaction and the use of multiple interfaces.

1.1 Motivation

Design, development, and investigation of MANETs are important research issues. Generally, for analysis and comparison of different protocols and algorithms four techniques are applied: (1) analytical modeling, (2) network simulation, (3) network emulation, and (4) real-world experiments. The potentials and limitations of these methods have been widely discussed in literature, e.g., [8].

Analytical models require many simplifications and are difficult or even impossible to deploy, while simulation models are often criticized for inaccuracies in capturing realistic node mobility behavior and wireless medium characteristics [13]. Furthermore, emulation cannot ensure accurate modeling of the wireless shared medium, while real-world testbeds are typically very limited in scope and introduce high management overhead.

To adjust to the shortcomings of each method, it is absolutely necessary to combine several methods to cross-validate and verify the results. However, there is a lack of emulation and real-world experiments that demonstrate the feasibility of ad hoc protocols and the consistency of simulation and real-world results. This is probably due to the complexity of getting familiar with setup mechanisms and constraints of multiple simulators, emulators, and real-world testbeds in order to compare various simulation and real-world experiments using same scenario setups. The task gets even more complicated if different protocol implementations and monitoring tools have to be taken into account during the analysis of results.

To overcome these problems, [6] and [7] recommend combining simulation with emulation and/or real-world testbeds to perform measurements in all areas mentioned above. Such a system should have the following characteristics:

- A *common code base* for simulation, emulation, and real-world testbed to avoid re-implementation of protocols and algorithms.
- A *simple* and *identical setup* of scenarios and monitoring tools to promote the specification of comparable experimental designs.
- A *common set of metrics* and *tools* for monitoring and analysis to enhance the comparison of results.

However, it is a challenging task to devise a combined framework for simulation, emulation, and real-world experiments, which (1) observes the different timing semantics of simulation and real-time, (2) utilizes various mobile hardware platforms as well as operating systems (OSs), and (3) incorporates existing real-world applications and services.

1.2 Contribution

In this paper we present JiST/MobNet, a Swiss army knife for combined simulation, emulation, and real-world experiments in MANET research. We describe a virtual machine-based real-time event concept based on Java in Simulation Time (JiST) [4], which allows for transparent coordination of simulation, emulation, and real-world testbed components. In combination with our Mobile Networking (MobNet) extension, which is based on the Scalable Wireless Network Simulator (SWANS) [5], we have built a cross platform experimentation framework for Windows and Linux.

In this paper we point out the advantages of using only a single testbed architecture, and demonstrate how such a system can be instrumented to gain new insights into realistic modeling of MANETs. We evaluate the performance limits of our design with respect to simulation and real-time models, and quantify the scalability of our solution. We present results from running various simulations, emulations, and real-world tests with synthetic as well as real traffic demonstrating the power and flexibility of our system.

1.3 Outline

The remainder of this paper is organized as follows. In Sec. 2 we give a brief overview of background concepts and related work in the area of simulation, emulation, and real-world experiments. In Sec. 3 we explain our proposed architecture and illustrate some implementation details. In Sec. 4 we conduct an extensive study of our approach and discuss the limitations of our solution. We close in Sec. 5 with a summary of our findings and an outlook for future work.

2. BACKGROUND AND RELATED WORK

This section presents the theoretical background and surveys related work. We analyse drawbacks of existing simulation and testbed systems with respect to a combined investigation of MANETs and summarize our findings.

2.1 Terminology and Semantics

In this paper we utilize the following important terminology in the context of simulation, emulation, and real-world testbeds:

Entity: An *entity* is a self-contained simulation/emulation component owning a temporal state. An entity class provides an event interface that defines the borders for events. It may comprise multiple objects keeping its state.

Event: An *event* is a time-based method call across entity borders to the event interface of another entity. Events are marked according to the time context of the caller for processing them in chronological order in the time context of the called entity.

The above terminology allows us to define network components as collections of entities representing the communication layers, services, and applications of participating devices. More terminology and semantics about emulation of networks can be found in [11].

2.2 Network Simulation

Many wired and wireless network simulators have been developed, so far. They differ with respect to programming language and tools, but also to accuracy, performance, and scaling of simulations. The most popular full featured, open source, discrete event wireless network simulators in research are NS-2/3 [1, 2], GloMoSim [22], JiST/SWANS [4] and OMNeT++ [20].

While **NS-2** was initially intended for wired networks, it has been extended to support MANETs. The code of NS-2 is split into C++ for its core engine and Tcl for configuration and simulation. This complicates development and verification of new core models and protocols. Moreover, the split object concept and the inadequate packet design result in huge memory consumption making it very hard to simulate more than a few hundred nodes. The generic monitoring interface of NS-2 has spawned a number of advanced analyzing tools, e.g., [14]. However, its inefficient design is responsible for up to 80% of simulation and analysis time.

NS-2 has been applied in emulations [17], but the drawbacks mentioned above become even worse in this domain. While using NS-2 code for real-world tests or vice versa is possible, e.g., [9], this approach lacks simplicity and platform independence as additional OS dependent hooks and tools are needed.

Currently, **NS-3**, the designated successor of NS-2, is under development. It will resolve many of the aforementioned drawbacks, but will remain platform dependent. Since the core design of NS-3 is not compatible with NS-2, NS-2 components cannot be reused in NS-3.

GloMoSim is a MANET simulator based on the discrete event simulation capabilities provided by Parsec [3]. It optimizes memory by so called node aggregation, which means multiplexing multiple simulation nodes in a single Parsec entity. This leads to lower performance costs but increases code complexity. The language and aggregation concept complicates the extension of GloMoSim for emulation and real-world experiments.

JiST/SWANS consists of two main parts; the Java in Simulation Time (JiST) kernel and the Scalable Wireless Network Simulator (SWANS), both running on a standard Java Virtual Machine (JVM). JiST provides discrete event simulation semantics to a standard Java Runtime Engine (JRE), which allows easy development of simulation models based on a pure entity concept. SWANS on top of JiST provides a basic (but very limited) set of simulation models and protocols, and the capability to attach real-world Java applications. In [5] the JiST/SWANS approach has been shown to out-perform popular simulators like NS-2 and GloMoSim. Although, it seems to be no longer supported by its developers, it has gained some attention in the community.

2.3 Network Emulation

Wireless network emulation is an even wider field than network simulation. A survey on emulation concepts and testbeds for MANET research can be found in [12]. Wireless network emulations provide very different characteristics with respect to degree of abstraction, emulation accuracy, and scalability. This is mainly due to the real-time restriction of emulation. Here we focus our description on MobiNet [16] and Neman [19].

MobiNet is a MANET emulator split in two parts, a controller part called *core nodes* and a part hosting the actual ad hoc nodes called *edge nodes*. One or even a set of core nodes are used for emulating topology-specific and hop-by-hop network characteristics, while the edge nodes generate the network load and forward their packets to the core nodes. Although MobiNet is sophisticated, its setup is complicated and, therefore hard to scale for a higher number of simulated nodes. With regards to our intention, it seems impossible to reuse the MobiNet concept and code for real-world experiments and simulations.

NEMAN uses a so called *topology manager* to manage several virtual tun/tap network devices (see also Sec. 3.5). Each of these devices represents a virtual node within the emulation. Processes attached to these interfaces implement network services. Connections between two nodes are established by tunnels between the corresponding virtual devices. The advantage of NEMAN is its high performance and the possibility of using real software within the emulation. However, the lower layer modeling is oversimplified, e.g., interference or multi-path effects are not taken into account. Moreover, it is hard to adapt the NEMAN approach for real-world experiments or simulations.

2.4 Real-world Experiments

Real-world experiments are very scarce due to the high costs and the fact that OSs today are not equipped for pluggable network and routing layers as needed for MANET research. Solutions such as the Ad hoc Protocol Evaluation (APE) testbed [15] or TrueMobile [10] have to extend OS capabilities. In addition, different OS dependent tools and scripts are needed to control and monitor experimental setups, making an adaptation for simulation impossible.

APE has two modes of operation. In the laboratory mode, nodes are logically connected by a MAC filtering tool. This emulates simple binary connectivity on fixed time instances derived from predefined mobility scripts. In the field mode, users are guided to move around by a visual tool. The APE framework provides a set of automated scripts for traffic generation control, as well as for collection, aggregation, and evaluation of monitored data.

TrueMobile is an extension of the EmuLab Testbed [21] for mobile wireless experiments. It is a kind of indoor testbed using small mobile robots carrying wireless communication nodes. Thus, a high accuracy of connection modeling can be reached at the expense of scalability as well as high costs.

2.5 Summary

As we have seen in the previous discussion it is difficult and time consuming to deal with all the different tools on different layers necessary for setting up, monitoring, and analyzing MANET experiments. Nevertheless, validation and verification of simulation and emulation must be done to obtain realistic and reliable results. A first attempt was performed by [18] in comparing different ad hoc emulation and real-word testbeds and their experimental results. [7] demonstrated significant differences in the results of different simulation, emulation, and real-world systems when applying comparable scenarios. [6] pointed out that there are important divergences even between simulators due to different implementations. Both blamed the lack of real experiments that prove the feasibility of wireless protocols as one cause of this problem.

To solve this problem, it is necessary to conduct not only *horizontal* analysis, i.e., investigating different testbeds on almost the same level of abstraction, but *vertical* analysis as well. Obviously, this is even harder if different tools measuring different metrics have to be taken into account.

Table 1: Capability of selected MANET research tools to produce results applying methods (2-4).

tool	simulation	emulation	real-world
GloMoSim	++	- -	- -
NS-2/NS-3	++	+	o
JiST/SWANS	++	o	- -
JiST/MobNet	++	++	++
MobiNet	-	++	-
NEMAN	-	++	+
APE	-	+	++
TrueMobile	-	+	++

While it is possible to extend open source tools to support new research methods, none of the existing tools has satisfying capabilities for producing results applying several of them (Tab. 1).

3. TESTBED ARCHITECTURE

Our testbed JiST/MobNet is based on the work of Rimon Barr [4]. It consists of the JiST kernel and rewriter, as well as the actual MobNet simulation libraries and drivers, all running on a standard Java Virtual Machine (JVM). This approach is independent of the underlying OS and should run on Linux, Windows and every other device providing a full featured JVM. In addition, Java provides the benefit of a standardized, highly-developed language. It is object-oriented and supports reflection; reasoning about simulation state at runtime is easy. Type-safety and garbage-collection simplifies writing of extensions and drivers by reducing common programming errors. Well designed development tools, e.g. the Eclipse framework, highly accelerate development, prototyping, and integration of new code.

We extended the original JiST to enable high performance simulations and emulations by supporting:

- *real-time* event execution without changing simulation code or models, and
- *parallel* event execution, to benefit from multi-core processor systems.

To reach these goals the original JiST kernel was refactored to reduce the lines of code and to increase its quality without changing its core function and behavior. The simplified code enabled us to implement and investigate several new JiST controllers for parallel event execution as well as real-time event handling.

The MobNet simulation extension runs on top of the JiST kernel and can be used to simulate a wide range of (even large scale) wireless ad hoc network scenarios. In contrast to SWANS, MobNet provides an advanced setup environment for various simulation scenarios, artificial traffic generators, on/off-line monitors, and analysis tools. The provided protocols and models in MobNet have been validated, corrected, and improved compared to SWANS.

To facilitate MobNet to support emulation and real-world testbeds, several extensions were introduced to enable integration of:

1. *standard applications*, using the common socket interface via a virtual tunneling device (tun wrapper), and
2. *network devices*, to allow for communication beyond the border of the simulation/emulation system (pcap wrapper).

These goals are achieved by implementing a protocol wrapper and a medium access layer wrapper using the Java Native Interface (JNI). They are mainly developed in Java using only a few lines of well documented C code. In the following the architecture of our system is described in more detail.

3.1 JiST - Simulation Time Controller

JiST consists of four basic components (Fig. 1). The compiler and runtime environment are standard Java components. The JiST rewriter is a dynamic class loader that modifies the Java classes while preserving their program logic. It facilitates simulation semantics by defining simulation entities and creating event calls, which are handled by the JiST kernel.

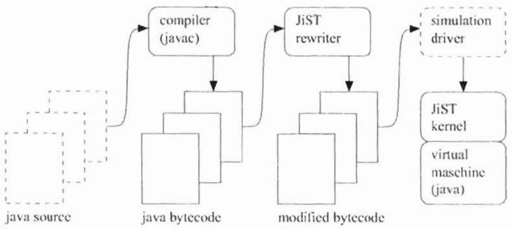

Figure 1: JiST architecture with compiler, rewriter, simulation kernel, and virtual machine showing JiST components highlighted [5].

The actual JiST kernel with its controller and scheduler transparently deals with all runtime aspects of the simulation, i.e., time-state abstraction and event call handling. The JiST kernel is very efficient, because it neither serializes message objects nor deals with threads and processes. Events between entities are automatically scheduled by the JiST kernel and handled as simple method invocations.

Entities communicate by timeless messages, i.e., messages that do not change over time, using a highly efficient zero-copy semantic. The messages are passed via the JiST kernel by reference. Message destruction is controlled by the garbage collector. Due to these features the resulting simulation code becomes very compact and intuitive. Thus, components in JiST are much smaller and more efficient than in GloMoSim or NS-2 [5].

3.2 JiST - Emulation Time Controller

As the original JiST only deals with event-based simulation time, we also applied an event-based emulation time semantic for our testbed. We modified the JiST kernel to utilize the system clock for timing and to distinguish between artificial delays, used for modeling computation times, and timeouts. In simulation time, events are delivered as fast as possible. In emulation time, events are delayed if the system time has not reached the time at which the even is scheduled. Thus, an accurate mapping of event execution times and delays to system execution times is crucial. This time mapping leads to several challenges and constraints that cannot be resolved easily.

In simulation systems, time depends only on simulation progress; the actual computation time of events has no effect on this progress. In contrast, in emulation systems, time advances independently of emulation progress. Thus, if the computation of an event takes longer than it should take in reality, an accurate emulation is almost impossible. While in simulation systems parallel events can be serialized without effect, this is not possible in emulation systems. We define the following time concepts:

Schedule Time: t_{sched} defines the time at which an event is actually scheduled and should be executed.

Execution Time: t_{exec} defines the time at which an event is really executed – before or after the scheduled time.

Computation Time: t_{comp} defines the time needed for computation of an event on real hardware.

Note, while t_{comp} is measured on the emulation system, it is independent from the estimated computation time of the emulated system. Based on these definitions we can formalize delays to describe the accuracy in emulation systems.

Execution Delay: The delay between the actual schedule t_{sched} and begin of event execution in t_{exec} is defined as execution delay $d_{exec} = t_{exec} - t_{sched}$.

Scheduler Delay: In general, there is a basic scheduler delay d_{sched} when preparing an event for execution, because of context switching. It depends on the internal clock and synchronization mechanisms of an emulation system.

Serialization Delay: If there are more events with a specific t_{sched} than available CPUs, events must be serialized. The later events experience a serialization delay d_{serial} that is at least as long as the accumulated $\sum t_{x,comp}$ of the previous executed events x.

Computation Delay: If t_{comp} of an event takes too long and, thus, exceeds the point in time t_{sched} when a follow up event should be scheduled, the child event experience a computation delay $d_{comp} = t_{parent,exec} + t_{parent,comp} - t_{child,sched}$.

To regard the above delay challenges we evaluated three design alternatives for event timing, i.e., system time, execution time, and schedule time. While base events are always scheduled with system time, child events can be scheduled in relation to t_{sched} or t_{exec} of parent events. In addition, different delay compensation methods can be applied.

While designing the emulation controller we were most concerned about the real-time constraints of the physical layer and its high degree of parallelism when receiving messages. We expected an accumulation of d_{serial} that would finally lead to a very high d_{comp} for received events on higher layers. To address these issues, we focused on a solution similar to [17], where child events are scheduled in relation to t_{sched}. In addition, we support three different delay models, i.e., no-delay, real-delay, and sim-delay. This allows for recovering from delay accumulation and fine-grained control of emulation scenarios with respect to hardware-dependent delay assumptions.

To guarantee at least soft real-time constraints and to avoid inaccuracies, it is necessary that events can be processed with a high performance. As this yields a deep impact on the number of nodes that can be emulated at the same time, we decided to stay on a single-threaded real-time controller to prevent negative effects from synchronization that were observed in the sim-time controller. Nevertheless, it can be shown that our approach already offers high performance. The real-time kernel can be configured to warn or to abort on violation of variable event-based limits.

3.3 MobNet - Simulation Extension

The original MobNet simulation extension runs on top of the JiST kernel and can be used to simulate a wide range of wireless ad hoc network scenarios. It is based on the TCP/IP reference model and provides different application models, transport and routing layer protocols, radio transmission, reception and noise models, as well as mobility models. The components are managed by the communication field, the node manager providing different node models, and a common traffic manager (Fig. 2).

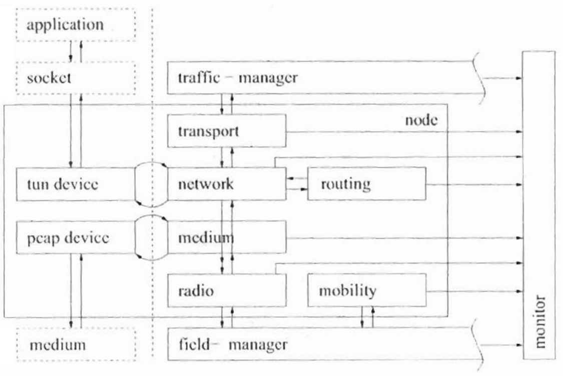

Figure 2: Component-based architecture of MobNet simulation extension (right) and emulation/real-world extension (left).

Each of these components is represented as a JiST entity communicating via the kernel. Based on standard interfaces, only relatively small components have to be created and can be exchanged easily. Thus, it is very easy to improve, simplify, or even exchange components. One needs only to implement a new component with the specific API to be able to plug it in.

The setup of simulation scenarios is managed by the simulation driver. A simulation scenario owns a number of virtual nodes, each connected to a common field layer. The field layer simulates the wireless medium and can be invoked with different fading and path loss models (Fig. 3).

3.4 MobNet - Emulation Extension

For emulation, the MobNet simulation extension was sufficiently advanced. We developed a component that allows standard software to communicate with the emulator using the common socket interface. Similar to NEMAN [19], the emulator provides virtual network devices that are bound to the virtual nodes.

To generate these interfaces, we use the tun/tap-driver for setting up virtual tunnels called *tun device*. A virtual tunnel acts on one side as network interface, on the other side

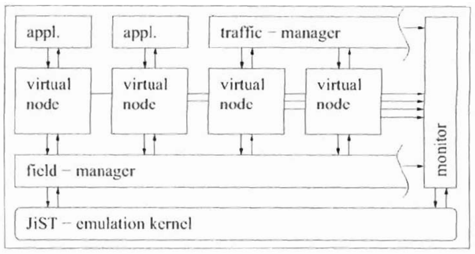

Figure 3: General JiST/MobNet architecture for combined simulation (right nodes) and emulation (left nodes) setup.

as pure file descriptor. Thus, packets send to these interfaces are written to files and vice versa. User applications just open their sockets for sending and receiving on these interfaces. The MobNet emulation reads/writes these messages to/from the related file descriptor and forwards them according its routing tables (Fig. 2).

Even so, Java does not provide the file descriptor needed to open a tun device. Thus, we implemented a small JNI interface to address this problem. It opens the tun device and returns a Java file descriptor for reading and writing. The specific binding for applications is done via our *socktun* library. Thus, it is possible for any application to open a dedicated connection to a virtual node by using the predefined tun device. In this way any number of applications can be bound to the virtual nodes, which forward messages via the emulated field layer (Fig. 3).

To handle incoming and outgoing messages concurrently on the tun device, MobNet uses multi-threaded architecture. As events can now occur outside of the causality chain, the JiST kernel must be synchronized for correct handling.

3.5 MobNet - Real-world Testbed

For the real-world approach, the emulation system is extended to use real network devices for communicating with other ad hoc systems. Nevertheless, using raw network devices with Java, as needed to bypass OS routing or device driver manipulations, is challenging.

We decided to use the *Packet Capture Library* (pcap) provided by the wireshark package in combination with a small JNI interface. The *pcap device* allows native communication to the network interface by sending and receiving packets as byte arrays. Thus, we extended MobNet to support serialization of all messages with respect to their correct packet format and pass them to the real interface (Fig. 4).

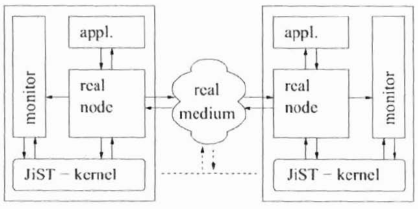

Figure 4: General JiST/MobNet architecture for real-world setup.

This solution has some drawbacks. The function of the pcap library depends on the capabilities of the network card driver. Most drivers can only be used for sending IEEE

802.3 packets. Thus, a fine-grained control and observation of IEEE 802.11 packets is impossible. Besides, the pcap library is inefficient and known to cause packet loss in combination with some drivers, but our approach provides enough performance for a simple real-world node setup on laptops or even PDAs (Fig. 4).

3.6 Summary

We highlighted the most important details of JiST/MobNet – a combined simulation, emulation, and real-world testbed framework for MANET research. JiST/MobNet is largely independent from the underlying OS using only (often already available) standard software components, i.e., Java, pcap, tun/tap. No special kernel patches, exotic software, or complicated configuration is necessary. This allows for simple real-world setups to provide full MANET capabilities on laptops.

4. PERFORMANCE EVALUATION

In this section we carefully evaluate the performance of our design and identify limits as well as possible improvements. In addition of black box tests for native communication, we develop several methodologies to evaluate the overall testbed and gain new insights into the general emulation problem. We present several results that demonstrate the performance trade-offs of our testbed.

4.1 Methodologies

To evaluate the performance of the testbed system we developed a pluggable monitor component for the JiST kernel. Sensing elements were placed in the controller to observe event creation and processing. Based on these, we analysed the observed events as follows:

Event Analyser: The *event analyser* analyses all arising events by counting them and calculating their execution times. As it considers all event methods and classes, it allows for identification of the most critical events for emulation.

Queue Analyser: The *queue analyser* counts the events arising for parallel execution in the scheduler queue. The more parallel events exist, the more problematic an adequate emulation becomes.

Delay Analyser: The *delay analyser* calculates the time difference between event time and execution time, and allows for identification of the overall performance of the emulation system.

Delay Distribution: In addition to the delay analyser, the *delay distribution* calculates a histogram of the measured execution delay d_{exec}, also considering the event methods and classes.

To measure the impact of the monitoring on the system performance, we evaluated the execution time of the JiST kernel with and without monitors (Tab. 2). The results presented here are the mean of 50 runs with 1 million *empty events* per run.

4.2 Interface Tests

To investigate the performance of the real-world communication using *tun* and *pcap devices*, we accomplished black box tests for the raw throughput of these devices on our Core Duo System. E.g., for the tun device we measured 22.3 MByte/s in both directions with no packet loss and a latency of 120 μs, which is adequate to most needs.

Table 2: Impact of measurement on system performance of JiST kernel.

evaluation method	execution time	deviation
No Analyser	3046.8 ms	+0.00%
Event Analyser	3083.3 ms	+1.20%
Queue Analyser	3099.9 ms	+1.74%
Delay Analyser	3124.4 ms	+2.55%
Delay Distribution	3176.8 ms	+4.27%

4.3 Performance Evaluation

To demonstrate the potential of the system we conduct several experiments to evaluate the implementation. All experiments have been conducted on an Intel Core Duo System with a 1.6 GHz CPU and 1.5 GB RAM. The nodes where initially placed in a symmetric grid in a way that only adjacent stations can communicate with each other. The reference setups for real-time simulation and emulation consists of a 9 nodes scenario (*small*) with 5 CBR streams in parallel and a 100 nodes scenario (*large*) with 10 CBR streams. Each CBR stream comprises 1 KByte/s using 512 byte packets.

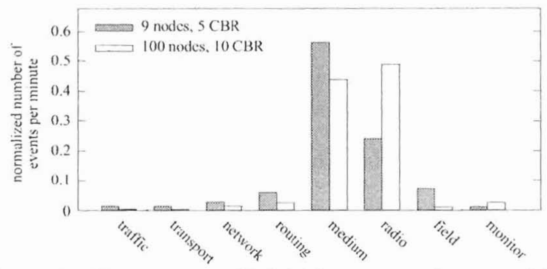

Figure 5: Frequency of MobNet event classes within JiST real-time simulations.

Fig. 5 shows the frequency distribution of event classes within a real-time simulation. An event class includes different event methods on the same layer, e.g., transmit and receive on radio layer. Medium access and radio layer events represent more than 80% of all events. In larger scenarios the radio layer (receive) is the dominating event class.

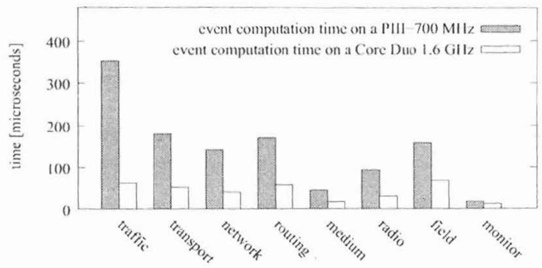

Figure 6: Average computation time t_{comp} of MobNet event classes within JiST real-time simulations.

Fig. 6 shows the average computation times t_{comp} of event classes. Even though event classes include different event methods, this provides a good first estimate, as more often occurring events are mostly faster. It can be seen that frequently occurring lower layer events are executed very fast and should fulfill most delay constraints, e.g., sending a small IEEE 802.11 message using 11 MBit/s needs around 1 ms. Most medium access layer events are even faster than the very small slot time of 20 μs used for the backoff interval.

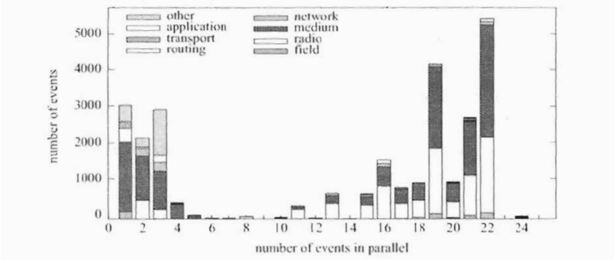

Figure 7: Number parallel events in a real-time simulation (9 nodes and 5 CBR streams in parallel).

Fig. 7 shows the frequency and deviation of parallel events in a *small* simulation. It can be seen, that at least 90% of all events can be executed in parallel. This effect stems from the broadcast behavior of the wireless medium in combined with omission of transmission delays that are in general <1 μs. Using this model, we can estimate the average computation delay t_{comp} of an event peek to be 560 μs, and 1041 μs for the worst case uniq 30 parallel events peek not shown.

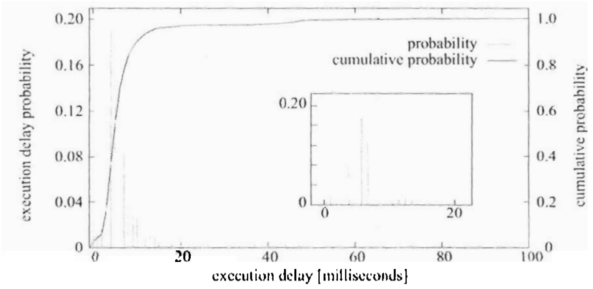

Figure 8: PDF and CDF of d_{exec} in a real-time simulation (9 nodes and 5 CBR streams in parallel).

Fig. 8 shows the probability distribution for event execution delays d_{exec}, i.e., the time between the actual schedule and begin of event execution, in a *small* real-time simulation. It can be seen that almost 90% of the events are executed within 10 ms. Only a few events are delayed around 50 ms, which cannot originate from parallel event peeks. We assume that the observed d_{exec} results from the setup phase, which is slowed down by bytecode rewriting, Just-In-Time (JIT) compiler, and prolonged Garbage Collector (GC) cycles >10 ms. This has to be analysed in future evaluations.

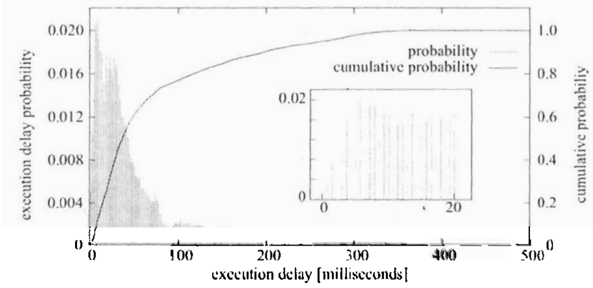

Figure 9: PDF and CDF of d_{exec} in a real-time simulation (100 nodes and 10 CBR streams in parallel).

Fig. 9 shows an identical *large* real-time simulation. It can be seen that the maximum as well as the average delay time increases. Only 80% of events are executed in 100 ms, even though the total real-time simulation execution time complies with the 60 s setup time. While high execution delays d_{exec} do not matter in real-time simulations, because they are corrected by the time concept (see Sec. 3.2), high d_{exec} certainly affects emulation results. For large scenarios, either the system performance has to be improved, or the models will have to be simplified.

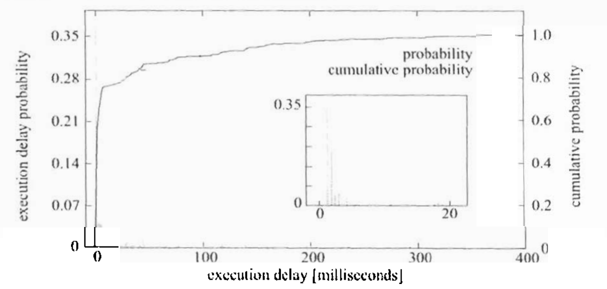

Figure 10: PDF and CDF of d_{exec} in a real-time emulation (9 nodes and 5 CBR data flows in parallel).

Fig. 10 shows the results obtained for a *small* emulation with 9 nodes and 5 CBR streams produced by Iperf according to the *small* real-time simulation (Fig. 8). The results show a high event peek at 1 ms, while some events are delayed for unexpected long times d_{exec}. We assume this to result from the missing real-time support by the OS, JIT compiler, and GC. The effects might be reduced by class pre-loading and pre-rewriting as well as running the GC after emulation setup.

4.4 Discussion

The performance evaluation of JiST/MobNet indicates a good scalability of the emulation testbed up to a few dozen nodes. Nevertheless, such performance tests provide only an indication of expected emulations accuracy. It should be noted, that the above results were gained without tuning the OS, the JRE, or the JiST kernel for real-time execution. E.g., we observed scheduler delays t_{sched} of up to 12 ms without any further event pending, which must have been induced by the OS, JIT compiler, or GC.

For a real-world testbed, where only a single node is running on each physical machine, the performance requirements are low compared to the emulation testbed. Thus, system performance can be assessed as adequate for this purpose. We conducted several real-world experiments using 5 Intel Pentium-III 700 MHz Laptops and external PCMCIA network cards without any problem; the results cannot be presented here due to space constraints.

5. CONCLUSION AND OUTLOOK

In this paper we presented JiST/MobNet – a combined simulation, emulation, and real-world testbed for MANETs. We explained the challenges and principles behind our approach and discussed its benefits. Furthermore, we derived the key features for using a discrete event simulation tool in an emulation and real-world domain. We described the challenges of providing correct and well-performing event execution in all domains as well as the communication mechanisms across domain borders. Based on these findings, we presented the actual design, implementation, and evaluation of our testbed.

There are several other emulation and testbed systems. However, to the best of our knowledge, JiST/MobNet is the only system allowing for simulation, emulation, and real-world experiments based on the same source code with common setup and monitoring environment. At the same time, JiST/MobNet is scalable, well performing, and can be easily extended with new protocols and services.

For the future we plan to develop more advanced single- and multi-threaded JiST controllers providing (1) busy-loops for short wait-cycles and (2) optimistic ahead-of-time event execution. In addition, event-chain optimized MobNet models are envisioned to improve emulation performance. We plan to examine these advances in combination with more advanced OS and Java soft and hard real-time support.

6. REFERENCES

[1] The Network Simulator – ns-2, 2004. [Online.] Available: http://www.isi.edu/nsnam/ns/.

[2] The Network Simulator – ns-3, 2007. [Online.] Available: http://www.nsnam.org.

[3] R. Bagrodia and R. Meyer. PARSEC: A Parallel Simulation Environment for Complex System, 1998.

[4] R. Barr. *An Efficient, Unifying Approach to Simulation Using Virtual Machines.* PhD thesis, Cornell University, Ithaca, USA, 2004.

[5] R. Barr. JiST: Embedding Simulation Time into a Virtual Machine. In *Proceedings of EuroSim Congress on Modelling and Simulation*, Ithaca, USA, 2004.

[6] D. Cavin, Y. Sasson, and A. Schiper. On the Accuracy of MANET Simulators. In *Proceedings of the Second ACM International Workshop on Principles of Mobile Computing (POMC 2002)*, pages 38–43, New York, NY, USA, 2002. ACM Press.

[7] F. Haq and T. Kunz. Simulation vs. Emulation: Evaluating Mobile Ad Hoc Network Routing Protocols. In *Proceedings of the International Workshop on Wireless Ad-hoc Networks (IWWAN 2005)*, London, UK, May 2005.

[8] R. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley-Interscience, May 1991.

[9] S. Jansen and A. McGregor. Simulation with Real World Network Stacks. In *Proceedings of the 37th Conference on Winter Simulation (WSC 2005)*, pages 2454–2463. Winter Simulation Conference, 2005.

[10] D. Johnson, T. Stack, R. Fish, D. Flickinger, R. Ricci, and J. Lepreau. TrueMobile: A Mobile Robotic Wireless and Sensor Network Testbed. Technical report, University of Utah, 2005.

[11] M. Kropff, T. Krop, M. Hollick, P. Mogre, and R. Steinmetz. A Survey of Real-World and Emulation Testbeds for Mobile Ad hoc Networks (Living Document). Technical report, Technische Universität Darmstadt, 2006.

[12] M. Kropff, T. Krop, M. Hollick, P. Mogre, and R. Steinmetz. A Survey on Real World and Emulation Testbeds for Mobile Ad hoc Networks. In *Proceedings of 2nd IEEE International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2006)*, pages 1989–1993, March 2006.

[13] S. Kurkowski, T. Camp, and M. Colagrosso. MANET Simulation Studies: The Incredibles. *ACM Mobile Computing and Communications Review*, 9(4):50–61, October 2005.

[14] S. Kurkowski, T. Camp, N. Mushell, and M. Colagrosso. A Visualization and Analysis Tool for NS-2 Wireless Simulations: iNSpect. In *Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS 2005)*, pages 503–506, Washington, DC, USA, 2005. IEEE Computer Society.

[15] H. Lundgreen, D. Lundberg, J. Nielsen, E. Nordstroem, and C. Tschudin. A Large-Scale Testbed for Reproducible Ad hoc Protocol Evaluations. In *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2002)*, Orlando, Florida, USA, Mar 2002.

[16] P. Mahadevan, A. Rodriguez, D. Becker, and A. Vahdat. MobiNet: A Scalable Emulation Infrastructure for Ad hoc and Wireless Networks. In *Papers Presented at the 2005 Workshop on Wireless Traffic Measurements and Modeling (WiTMeMo 2005)*, pages 7–12, Berkeley, CA, USA, 2005. USENIX Association.

[17] D. Mahrenholz and S. Ivanov. Real-Time Network Emulation with NS-2. In *Proceedings of the 8th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2004)*, pages 29–36, Washington, DC, USA, October 2004.

[18] E. Nordstroem, P. Gunningberg, and H. Lundgreen. A Testbed and Methodology for Experimental Evaluation of Wireless Ad hoc Networks. In *Proceedings of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM 2005)*, Washington, DC, USA, 2005. IEEE Computer Society.

[19] M. Puzar and T. Plagemann. NEMAN: A Network Emulator for Mobile Ad-Hoc Networks. In *Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005)*, pages 155–161, Zagreb, Croatia, June 2005.

[20] A. Varga. The OMNeT++ Discrete Event Simulation System. In *European Simulation Multiconference (ESM 2001)*, June 2001.

[21] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An Integrated Experimental Environment for Distributed Systems and Networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.

[22] X. Zeng, R. Bagrodia, and M. Gerla. GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks. In *Proceedings of the Twelfth Workshop on Parallel and Distributed Simulation (PADS 1998)*, pages 154–161, Washington, DC, USA, 1998. IEEE Computer Society.