Technical Report #321, ISBN 82-7368-274-9
Department of Informatics, University of Oslo
March 2005

NEMAN: A Network Emulator for Mobile Ad-Hoc Networks

Matija Pužar, Thomas Plagemann
Department of Informatics, University of Oslo
{matija, plageman}@ifi.uio.no

Abstract – **Development of applications and protocols for wireless ad-hoc networks has always been a challenge. Specific characteristics such as frequent topology changes due to nodes moving around, popping up or being turned off, need to be considered from the earliest stages of development. Since testing and evaluation using genuine wireless devices is both expensive and highly impractical, other tools need to be used in the development phase. Simulators give a very detailed model of lower layers' behaviors, but code often needs to be completely rewritten in order to be used on actual physical devices. Emulators present a trade-off between real test beds and simulators, providing a virtual wireless network at the lowest layers, and yet allowing real code to be run on the higher layers. In this paper, we present such an emulation platform, called NEMAN, that allows us to run a virtual wireless network of hundreds of nodes on a single end-user machine. NEMAN has shown to be an important and very useful tool during development of different applications and protocols for our project, including a key management protocol and a distributed event notification service.**

Keywords – mobile ad-hoc network, network emulation, topology

I. Introduction

Information sharing is a mission critical key element in rescue and emergency operations. Mobile ad-hoc networks (MANETs) could provide a useful infrastructure to support information sharing, but appropriate applications are needed. In addition, middleware support has to be present to facilitate efficient application development for this type of infrastructure. In the Ad-Hoc InfoWare project [9], we are addressing these needs by developing middleware services for information sharing. The core building blocks of these services are knowledge management, a local and a distributed event notification service, resource management, and security and privacy management. As part of the development process, it is necessary to analyze and compare design and implementation alternatives for the building blocks, understand qualitatively and quantitatively the design trade-offs, to test whether the protocols and algorithms actually work, and to evaluate their efficiency and compare them with related solutions from others. Basically, there are three kinds of development environments that support these tasks: simulation, emulation, as well as implementation and field tests.

The development environment that comes closest to real live deployment are field tests of MANETs. However, field tests of wireless scenarios are expensive with respect to number of devices and number of persons that are needed to perform them as well as the time it takes to prepare them. Furthermore, it is very hard to entirely control all parameters in a field test and to perform repeatable experiments. Field tests are clearly not perfectly suited to support the development of middleware protocols, services and applications for MANETs. On the other hand, both simulation and emulation provide controlled environments which enable repeatable experiments and are generally much cheaper than field tests. To facilitate the development and testing of such applications and protocols, it is important to carefully choose a suitable simulation or emulation tool.

Simulators, such as GloMoSim [14] and ns-2 [12], have long been used in the ad-hoc field and make it possible to experience very diverse communication situations and a large scale of deployment. One of the goals of these simulators is to give a detailed representation of the physical layer. A major drawback is that learning how to use and program a simulator like ns-2 takes a substantial amount of time. Furthermore, the code that has been developed for these simulators needs to be rewritten for later deployment on real systems. By using an emulating platform, however, real processes run in real time and one can immediately start writing the very same code that will be later used on wireless devices.

In particular, we have the following requirements on a simulation or emulation environment for the development of protocols, middleware services and applications in the Ad-Hoc InfoWare project:

- *Minimal initial effort*: installing and learning to use a particular simulation or emulation environment should not consume too much time and effort such that it can also be efficiently used from novices in shorter projects, like master theses.
- *Costs*: we need an affordable solution, including hardware and software costs as well as human resources. Thus, the environment should be is able to run on a single standard PC.
- *Scalability*: the chosen platform should be able to run a high number of nodes without severe performance loss.

This work has been funded by the Norwegian Research Council in the IKT-2010 Program, Project Nr. 152929/431.

- *Portability*: the code developed for the applications and protocols should be portable to genuine wireless devices with minor or no changes at all.
- *Realistic network layer:* our protocols are supposed to utilize existing ad-hoc routing protocols, therefore, a real routing protocol should run in the simulation or emulation environment. During the development process, our main concern is connectivity and loss of connectivity between nodes. Therefore, quality of service and specific lower layer issues such as collisions in the air, hidden terminals, etc. are of lower importance for us than using real routing protocols and reflecting the effects of mobility in MANETs on connectivity.
- *Possible comparability*: much work in the area is done with ns-2 and we want to be able to compare our solutions with those from others without re-implementing them in our simulation or emulation environment. Using standard formats for scenario files, such as those from ns-2, would enable us to perform experiments with the same scenarios and to compare our results with results obtained with other tools.

Many researchers are aware of the need for appropriate development environments of MANET protocols, but the majority is being focused on link layer and network layer issues. Therefore, we could not find existing simulation or emulation environments that completely fulfill our requirements. Inspired by the approach of the network emulator MobiEmu [15], we have developed an emulation platform called NEMAN. To the best of our knowledge, NEMAN is the only platform able to emulate MANETs consisting of hundreds of nodes on a single PC. Based on scenario descriptions from ns-2, NEMAN controls the physical connectivity between the virtual mobile nodes. Currently, we use the Open Link State Routing Protocol (OLSR) [1] with NEMAN to establish and maintain the IP layer of the emulated network. The processes in these virtual mobile nodes bind to virtual network interfaces, i.e., TAP interfaces. By this, the code developed for NEMAN can be used in real wireless nodes with a minimal effort. It is the aim of this paper to describe the design, implementation and first evaluation of NEMAN, and to analyze its strength and weaknesses.

The rest of the paper is structured as follows. Section II shows some of the most important related network emulators. In Section III, we present the architecture of NEMAN. Implementation details are described in Section IV, while in the following sections we explain the application development process (Section V) and describe our experiences in using NEMAN for the development of a key management protocols and a distributed event notification service in the Ad-Hoc InfoWare project (Section VI). Performance and scalability are described in Section VII. Finally, Section VIII gives a conclusion and ideas for the future work.

II. Related Work

A common way to perform emulations is to have a single machine for each emulated node. In such cases, filtering at the MAC layer is used to achieve the notion of wireless topology, often by means of *iptables*.

MobiEmu consists of several *slave* nodes, as well as one *master* node. As the master gives instructions on topology changes, the slaves set local *iptables*-rules preventing them from hearing traffic from those nodes they have no physical connectivity with. However, the concept of having separate physical or virtual machines for each emulated node, made MobiEmu very impractical for us to use it as such.

MNE (Mobile Network Emulator [6]) uses a static network infrastructure to interconnect devices. Each device has two interfaces, where one acts as a mobile emulation control channel while the other is used for the emulated wireless network. The latter can be an actual wireless interface, allowing for some lower layer effects (such as collisions) to be taken into account as well. Information about topology changes is sent through the control channel, causing the nodes to set or remove *iptables*-rules accordingly, as it is done in MobiEmu. The main problem of this approach is that it still needs a separate device for each emulated wireless host.

EMWIN [16] improves the issue with the number of physical machines by allowing each node to have several network interfaces, each acting as a separate wireless node. EMWIN intends to provide emulation of some MAC layer effects by introducing an additional emulated MAC (eMAC) layer. Again, due to a relatively high number of machines required, this approach is still impractical for our needs.

MobiNet [7] consist of *core* nodes, used to emulate topology-specific and hop-by-hop network characteristics, and *edge* nodes. It is able to emulate a much larger number of virtual wireless devices by having multiple Virtual Edge Nodes (VNs), with different IP addresses, on each *edge* node. MobiNet has a built-in routing protocol (DSR) and emulates MAC layer effects as well. Although the number of physical devices required to run MobiNet is drastically reduced, and the platform seems to be very well developed, its setup is still somehow complicated, with regards to our requirements.

JEmu [3] was developed by the Networks & Telecommunications Research Group (NTRG) to emulate the radio components of their particular communication stack. To represent nodes in the emulation, JEmu uses genuine wireless devices with different types of wireless communication links, as well as stationary machines. JEmu has a somehow different approach when it comes to topology simulation. Every packet is first sent to the emulation engine which then decides whether certain nodes are able to receive it or whether there should be a collision, in which case it depends on the specific configuration what should be done.

TABLE I
PROPERTIES OF VARIOUS NETWORK EMULATORS

	MobiEmu	MNE	EMWIN	MobiNet	JEmu
Usage	✓				
Low costs				✓	
Scalability			✓	✓	
Portability	✓	✓	✓	✓	✓
Routing				✓	
Comparability				✓	

Table I summarizes roughly the properties of these related works with respect to our particular requirements. As it can be seen from the table, none of them fulfills all the requirements.

III. ARCHITECTURE

NEMAN is designed to emulate a relatively large scale wireless network, up to hundreds of nodes, within a single physical machine. With that respect, NEMAN is closest to MobiNet. The NEMAN architecture comprises the following three elements, as shown on Fig. 1:

- the *user processes* represent actual applications and protocols that are being tested, including routing daemons,
- the *topology manager* manages virtual network interfaces and performs packet switching according to the topology information at a certain moment in time, and
- the *graphical user interface* (GUI), used to visualize the emulated network and to induce the topology information to the topology manager

All the components, including the topology manager, run in the user space of the Linux operating system. Root-privilege is only needed to configure the virtual network interfaces.

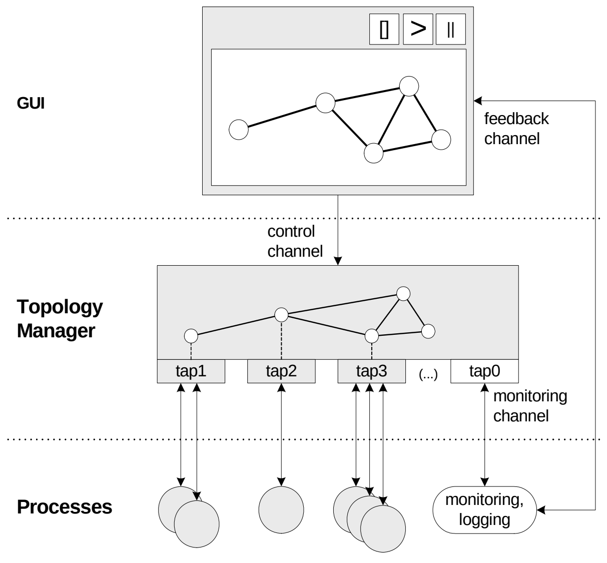

Fig. 1. NEMAN architecture

On top of this basic emulated network infrastructure, user processes hook to virtual Ethernet network devices, called TAP devices. TAP devices are available in the Linux kernel and provide low level support for Ethernet tunneling. User processes can send and receive data via TAP interfaces using the classical socket API, thus achieving portability of code. The only requirement for the sockets is to use the specific option SO_BINDTODEVICE, ensuring that a process' socket will listen and send only to the specified interface, and thus not interfere with traffic addressed to some other process. This is an important requirement having in mind that all the emulated processes run on the same machine. One example of such user processes are the routing daemons. Since this infrastructure emulates the physical and the link layer of MANETs, we must first establish an IP infrastructure by means of routing daemon processes hooked to the TAP interfaces. Other processes that are hooked to the same interfaces can afterwards use the established IP infrastructure, allowing us to implement and test middleware and application layer protocols, which was exactly our goal. A group of processes hooked to the same TAP interface represents a node in the emulated network, called *virtual node*.

The *topology manager* is the core of NEMAN. It is the user-space application creating and maintaining the TAP devices. Since TAP devices provide Ethernet tunneling, we ensured independence with regards to the IP version being used. Every frame received on a TAP interface is available to the topology manager, and every frame induced by the topology manager into an interface is available to the processes hooked to it. In other words, when the topology manager gets a frame sent to one of its TAP interfaces, it can then decide to forward it to some of the other interfaces (or none), according to the topology information it has at the moment. Due to the fact that all the nodes run on a single machine, the routing table of the kernel is not taken into account (this is described more in depth in Section IV). One TAP interface (in our case, tap0) is reserved as the *monitoring channel*, having an open bidirectional connection to all the other TAP interfaces, independent from the topology. This is a very important feature, allowing us to perform analysis of the network traffic using standard tools such as *tcpdump* or *ethereal*. Moreover, having in mind that the monitoring channel works both ways, we are able to use the same channel to induce traffic into the virtual network from the "outside world". This feature comes useful when applications or services need to be triggered at a specific moment of time. An example of such a service is shown in Section VI.B.

The implementation of the GUI is currently based on MobiEmu's GUI. It is a Tcl/Tk script, independent from the topology manager and can run on a separate machine. The GUI shows the current position of nodes, their transmission ranges and links between nodes that can directly communicate with each other. Topology and node movement data are acquired from standard ns-2 scenario files, created by, for example, ns-2's *setdest* program. Scenario files are interpreted sequentially, allowing us to introduce some application-specific events at specific moments in the emulation, thus achieving repeatable results. Information about topology changes is sent to the topology manager through the control channel, in form of UDP packets. The GUI allows also any user process to give it some feedback, so that important state changes in a user process can be visualized as e.g. color changes in the GUI. An example, where this has proved to be a very useful feature, is described in Section VI.A.

The current implementation of the emulator provides us a working network infrastructure, essential for the development process of higher layer applications. In the conclusion, we discuss some possible future extensions, including the emulation of characteristics typical for wireless networks, such as collisions in the air, hidden terminals, obstacles, etc.

IV. Implementation Details

Since all virtual nodes run on the same physical machine, we had to solve some problems that are not present in the "real" world, i.e., where each virtual node is also a physical node.

One of the major problems we noticed was that the Linux kernel gracefully ignores all incoming packets that have been sent from one of its own interfaces. Although in most situations this is a reasonable thing to do, with respect to security issues and prevention from possible message loops, it was not an option in our case, where local interfaces were the only ones communicating between themselves. The solution was to implement the send-to-self (STS) patch developed by Ben Greear, which fixes the problem, allowing for local traffic to be received as well.

The next problem emerged when the first ARP packets started coming from our test-applications. For any ARP packet coming to the machine, independent on the IP address being queried, all local interfaces that hear the query answer with their own MAC address. Again, this might be a useful feature in most of the standard situations where an IP address represents a physical machine, but in our case the amount of unnecessary traffic generated (with false information) was unacceptable. There are two possibilities to solve this problem. The first one is to implement a kernel patch, called *hidden*, developed by Julian Anastasov. This patch allows for certain devices to be hidden from other devices (hence the name), when it comes to ARP requests. The second possibility is to let topology manager directly answer all ARP requests between its TAP interfaces. Although the second approach might reduce a bit on realism at the lower layers, for our case, where we are mainly concerned about connectivity and topology changes, it is acceptable. Furthermore, it turned out that the multi-hop routing problem can be solved in a corresponding way

The multi-hop routing problem is caused by the fact that IP addresses of "remote" virtual nodes are tied to local TAP interfaces. This contradiction prevents the routing protocols to set routes towards such addresses, which is the case for all of our emulated nodes. Using dynamically created *iptables*-rules to perform routing on a single machine proved not to work either. Therefore, we decided to solve the problem on a higher layer, without introducing additional patches into the kernel. The implementation of the OLSR routing protocol we are using in our emulated network, the *olsr.org OLSR daemon* [13], writes every route change to the standard output. This enables us to induce that information directly into the topology manager, through a parser developed especially for that purpose. The topology manager then forwards every packet hop by hop according to the requests from the OLSR daemons to its final destination. By solving the routing issue that way, no changes were needed to either the routing daemon or kernel. In addition, this approach allows us to use other routing protocols (such as AODV [8]) as well, with either no changes at all or just minor changes to the routing daemon's output and/or to our parser.

The Linux kernel has a hard-coded upper limit of 100 network interfaces of the same kind. To overcome this limitation it is only necessary to increase the maximum number of iterations in a particular loop in the kernel. This minor change in the kernel increases the scalability of NEMAN such that only the server's resources determine the upper limit.

V. Application Development

One of the main points of using an emulator is to be able to port applications from the emulating platform to genuine wireless devices without need having to change the code. Any application able to hook to e.g. Ethernet (eth*), 802.11 (wlan*) or similar devices can be used directly with TAP devices as well. The only precondition is that, in order not to mix with other applications' data, an application has to listen and send only to the specified devices, which is accomplished by using the previously mentioned SO_BINDTODEVICE option when creating the socket.

Fig. 2. shows the typical process of using NEMAN.

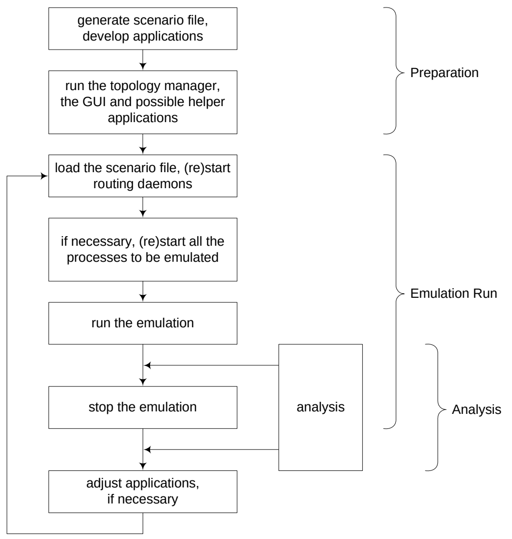
generate scenario file, develop applications
run the topology manager, the GUI and possible helper applications
Preparation
load the scenario file, (re)start routing daemons
if necessary, (re)start all the processes to be emulated
run the emulation
Emulation Run
stop the emulation
analysis
Analysis
adjust applications, if necessary

Fig. 2. Workflow diagram of using NEMAN

The process is roughly divided into three phases. In the preparation phase, the scenario files and initial user process' code are developed, and the emulator core is initiated. In the next phase, user processes are started. The analysis phase can overlap with the run-phase, since traffic can be monitored on the fly through the monitoring channel. The last two phases are then iterated until the wanted functionality is achieved.

Apart from the applications being tested, additional helper applications might be needed to, for example, restart routing daemons on demand or to listen to messages triggered by custom events in the scenario file. The monitoring channel provides the means to perform analysis on what is going on in the network on the fly, by hooking to tap0 with e.g. standard tools like tcpdump. In addition, all the traffic can be saved for a later, more detailed analysis.

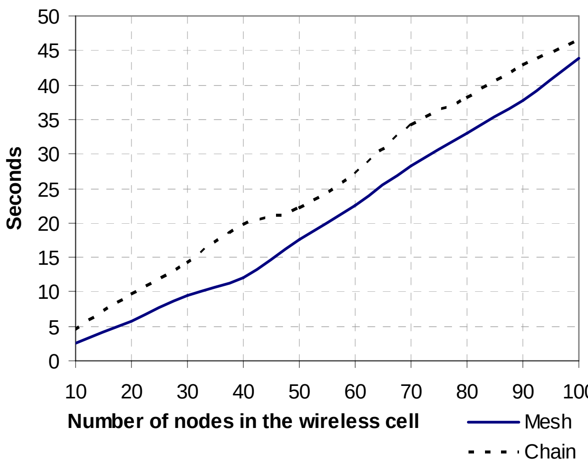

Fig. 3. Time needed to achieve a stable shared key

VI. Experiences

NEMAN was developed primarily to test and evaluate applications and protocols for the Ad-Hoc InfoWare project. Here, we present our experiences with some of the applications and protocols we tested and which already benefit from the emulation platform.

The olsr.org OLSR daemon was the first example showing that already existing applications needed no modifications to be able to work with NEMAN. Indeed, at each moment the output from a certain daemon, showing its 1-hop and 2-hops neighbors, would fully match what was displayed in the GUI.

A. *A Simple Key Management Protocol for MANETs in Emergency And Rescue Operations (SKiMPy)*

The SKiMPy key management protocol [10] is used to establish a symmetric shared key between the rescue personnel's devices. At the network layer, the key provides the means for establishing a secure network infrastructure between authorized nodes, while keeping out unauthorized ones. In addition, at the application layer it may be decided whether the established shared key will be used to encrypt data as well. SKiMPy is designed and optimized for highly dynamic ad-hoc networks and it is completely autonomous, requiring no user interaction at all.

In the current implementation, the key management protocol has been coded directly into the security plugin [5] of the OLSR routing daemon. Although SKiMPy is mainly designed to protect all traffic and not only routing, this was still a good opportunity to test and analyze it in a realistic environment with a real routing protocol.

The emulator was an essential tool to test the performance and scalability of SKiMPy. We tested it for two different static scenarios, called chain and mesh.

In a chain scenario, all nodes are lined up in a single chain and the distance between all nodes in the chain is such that only the direct neighbors can communicate in a single hop with each other. We consider this to be the worst case scenario for SKiMPy, since its performance benefits from having more neighbors.

In a mesh scenario, however, nodes have multiple, randomly scattered neighbors, as it is natural in ad-hoc networks. Having multiple neighbors allows the protocol to reduce traffic and resource consumption.

Ten independent runs were performed for each number of nodes and each scenario, and the results on Fig. 3. show that the key management protocol scales linearly with linear increase of the number of nodes and physical network area accordingly (thus giving the same density of nodes). In an additional test, we tried to see the scalability of the protocol when the size of the physical area remains constant while the number of nodes increases. In this case, there is much more network traffic involved, as well as computation, making the test-machine the bottleneck. Therefore, we were only able to test up to 50 nodes, but the results for these measurements have shown minimal differences in the performance, compared to the results when the density of the nodes was constant.

One of the useful features of MobiEmu's original GUI is that it accepts certain feedback messages. A special application constantly monitors all traffic on the monitoring channel (tap0) and analyzes signed OLSR packets, containing the ID number of the key used to perform the signature. When a change is noticed, the ID number is converted to a 24-bit RGB color code and sent as feedback to the GUI, which then colors the node on the screen accordingly. That way, we got a simple and yet effective way to see in real time how the protocol works and how the keys are spread through the network.

To conclude, NEMAN has played an important role in the process of developing, testing and evaluating SKiMPy. It presented us both numerical and graphical proofs that the protocol indeed worked as expected.

B. *Distributed Event Notification Service (DENS)*

DENS [11] is a communication tool in our architecture, providing an asynchronous communication mechanism using the publish/subscribe model which is well suited for a highly dynamic environment such as our scenario. Some of the nodes in the network are chosen as so called DENS-nodes. These nodes have the role of mediators or brokers between the subscribers and publishers, so both subscriptions and notifications are sent to them. In a MANET disconnections causing partitioning are frequent and the DENS-nodes may therefore get disconnected as well. We want the service to be highly available so our solution should survive such network partitions. This is achieved by replicating information about subscriptions among the DENS-nodes. However, the replication of state information together with network partitions can easily lead to inconsistent replications. One important tasks for DENS-nodes is therefore to regain a consistent state as fast as possible. In addition, DENS-nodes may keep notifications for a specified amount of time, allowing nodes that are temporarily out of reach to still get the information. Scalability is also an issue since having too many DENS-nodes would create a lot of traffic when synchronizing and replicating information about subscriptions and notifications, so we need to run tests to find the right trade-off between availability of the service and still making it scalable.

The first implementation of the DENS protocol is currently under development and NEMAN has proven to be an essential tool for analyzing and debugging the protocol. To trigger DENS-nodes to perform subscriptions and notifications, which in the real scenario will be done at the application layer, we use the monitoring channel for its other purpose, i.e. inducing messages. Control messages are sent through the monitoring channel to DENS nodes either directly from the shell, or from the GUI. Special lines can be added to the scenario file, causing the GUI to forward them at the specified time, through a proxy application, into the monitoring channel.

VII. Performance and Scalability

So far, we have been running the emulator using up to 100 nodes and have only noticed degradation of performance when nodes have in average more than 50 direct neighbors and all would be turned on at the same time, causing massive key exchange within the SKiMPy protocol. In all the other circumstances, no problems were noticed. The load on the machine and the amount of control messages coming to the control channel from the GUI and the routing parser is reasonable, even with a high number of nodes and link changes. As one of the next steps, we will try to analyze in more details where exactly, with respect to the number of nodes, do we have boundaries of normal usage.

One of the main places for performance improvement is the GUI. Although it is easy to use and to be extended with new functionalities, its performance becomes highly degraded as the number of nodes and active links is increased, even with the minimal smoothness factor. To avoid this problem, the graphical display can be temporarily turned off in cases where results are needed quickly and they do not depend on time. This can be done either by changing the application or by simply minimizing the window. Another problem with large scenarios (either with regards to the time domain or number of nodes) is that the GUI loads the whole scenario in memory before starting to interpret it. Assuming that reading the scenario file on the fly would not impose a new bottleneck, this might be a point to investigate.

VIII. Conclusion and Future Work

In this paper, we described the requirements, design, implementation, and experiences with our emulation platform for MANETs, called NEMAN. Development of middleware services for emergency and rescue operations is the main focus of Ad-Hoc InfoWare project. In this context, NEMAN has already proved to be an important and very useful tool for the development of our key management protocol SKiMPy and the distributed event notification service.

There are several other simulation and emulation environments of MANETs, however, to the best of our knowledge, NEMAN is the only emulation platform allowing to perform MANET emulations of up to hundreds of nodes on a single machine. By this, NEMAN is a cheap and efficient environment to develop middleware protocols, services, and applications. Furthermore, the code that has been developed on NEMAN can be easily ported and deployed on genuine wireless devices. Obviously, this facilitates considerably the step from development in an emulation environment to “real life” field trials and deployment. We hope that this will also contribute in the future to increase the number of research results that are not only developed and evaluated within a simulation or emulation environment, but are also tested with real devices in field trials.

While the current implementation of NEMAN has already proven to be very useful, there are still some aspects of NEMAN we would like to improve, respectively to extend. We envisage three possible threads of future work. First, we are interested to increase the number of nodes that can be efficiently emulated. To overcome the resource limitations of a single PC, we are looking into design and implementing a NEMAN version that can run concurrently on multiple PCs. We might even include heterogeneous computing platforms into the simulation to emulate the software where it actually should be running. For creating such a distributed simulation for ad hoc networks, we are investigating distributed simulation, component, and multi-agent architectures and technologies such as the High Level Architecture (HLA) [2] and Foundation for Intelligent Physical Agents (FIPA) architecture [4] for multi-agent systems.

Second, we would like to extend the functionality of the topology manager to not only emulate the connectivity between nodes according to a scenario description, but also to emulate some other link layer properties, like Quality-of-Service, hidden terminals, etc. The emulation of these aspects will require much more resources than just deciding whether a virtual device should receive a packet or not. Thus, the availability of a distributed NEMAN version might be necessary to perform such emulations with larger number of nodes.

The third thread of possible future work is concerned with the GUI. The GUI is currently the performance bottleneck in NEMAN. This bottleneck is caused by the fact that the GUI is implemented in Tcl/Tk. A re-implementation with a language that can be compiled should lead to substantial performance improvements. Furthermore, we are interested to extend the functionality of the GUI. We would like to use the GUI to turn on and off nodes at any stage of the emulation by just clicking on them with, be able to create new nodes on the fly, and even influence the link layer properties by clicking on them.

ACKNOWLEDGMENTS

This work has been performed in the Ad-Hoc InfoWare project and we would like to thank all project team members and related colleagues for many fruitful discussions and other contributions. Special thanks go to Katrine S. Skjelsvik for contributing with valuable feedback on NEMAN as a patient beta tester, and Erek Göktürk for providing important insights into related work. Furthermore, we would like to acknowledge the MobiEmu project, whose GUI we used as a starting point for creating NEMAN.

REFERENCES

[1] Clausen T., Jacquet P., “Optimized Link State Routing Protocol (OLSR)”, RFC 3626, October 2003.

[2] IEEE-SA Standards Board, IEEE Standard 1516

[3] Flynn, J., Tewari, H., O'Mahony, D., "A Real Time Emulation System for Mobile Ad Hoc Networks", Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference, 2002.

[4] Foundation for Intelligent Physical Agents (FIPA) standard no: SC00001L, FIPA Abstract Architecture Specification, 2002.

[5] Hafslund A., Tønnesen A., Rotvik J. B., Andersson J., Kure Ø., “Secure Extension to the OLSR protocol”, OLSR Interop Workshop, San Diego, August 2004.

[6] Macker, J. P., Chao, W., Weston, J. W., "A low-cost, IP-based mobile network emulator (MNE)", MILCOM 2003 - IEEE Military Communications Conference, 2003, 22, 481-486.

[7] Mahadevan, P., Rodriguez, A., Becker, D., Vahdat, A., “MobiNet: A Scalable Emulation Infrastructure for Ad Hoc and Wireless Networks”, UCSD Technical Report CS2004-0792, 2004.

[8] Perkins, C., Belding-Royer, E., “Ad hoc On-Demand Distance Vector (AODV) Routing”, RFC 3561, July 2003.

[9] Plagemann, T., *et al.*, “Middleware Services for Information Sharing in Mobile Ad-Hoc Networks - Challenges and Approach”, Workshop on Challenges of Mobility, IFIP TC6 World Computer Congress, Toulouse, France, August 2004.

[10] Pužar, M., Andersson, J., Plagemann, T., Roudier, Y., “SKiMPy: A Simple Key Management Protocol for MANETs in Emergency and Rescue Operations”, Technical Report, Department of Informatics, University of Oslo, February 2005.

[11] Skjelsvik, K. S., Goebel, V., Plagemann, T., “A Highly Available Distributed Event Notification Service for Mobile Ad-hoc Networks”, ACM/IFIP/USENIX 5th International Middleware Conference (Middleware 2004), Toronto, Canada, October 2004.

[12] The Network Simulator - ns-2, http://www.isi.edu/nsnam/ns/

[13] Tønnesen A., “Implementing and extending the Optimized Link State Routing protocol”, http://www.olsr.org/, August 2004.

[14] Zeng, X., Bagrodia, R., Gerla, M., “GloMoSim: a Library for the Parallel Network Simulation Environment”, Proceedings of the 12th Workshop on Parallel and Distributed Systems, 1998

[15] Zhang, Y., Li, W., “An Integrated Environment for Testing Mobile Ad-Hoc Networks”, MOBIHOC'02, EPFL Lausanne, Switzerland, 2002.

[16] Zheng, P., Ni, L. M., “EMWIN: Emulating a Mobile Wireless Network using a Wired Network”, 5th ACM international workshop on Wireless mobile multimedia, Atlanta, Georgia, 2002.