Emulating Mobile Ad-hoc Networks of Hand-held Devices. The *OCTOPUS* Virtual Environment

Fabio D'Aprano Massimiliano de Leoni Massimo Mecella
Dipartimento di Informatica e Sistemistica
SAPIENZA - Universita' di Roma, Sede distaccata di Latina
{daprano,deleoni,mecella}@dis.uniroma1.it

ABSTRACT

Nowadays, Mobile Ad-hoc NETworks (MANETs) are investigated and adopted in many application scenarios, such as emergency management. Unfortunately, testing and validating complex systems developed on top of MANETs is still a difficult and error-prone task, due to the scarce availability of virtual environments. In this paper, we present OCTOPUS, a virtual emulation environment for MANETs, in which designers and developers can create scenarios in large scale areas including obstacles, defining different movement models, and use real hand-held devices for deploying their software/applications.

Categories and Subject Descriptors

I.6.7 [**Simulation and Modeling**]: Simulation Support Systems—*Environments*

General Terms

Experimentation

Keywords

Mobile Ad Hoc Networks, Emulation, Hand-held Devices, Wireless, Emergency Management

1. INTRODUCTION

A Mobile Ad-hoc NETwork (MANET) is a P2P network of mobile nodes capable to communicate with each other without an underlying infrastructure.

In the context of the European project WORKPAD[1], we are developing a two layer architecture for disaster management [1]. At front-end, different organizations involved in handling emergency (such as Fire Brigades, Homeland Security, etc.) send to the affected area some teams, each one arranged in a MANET. The back-end layer is in charge of providing information to teams and coordinate their intervention to enhance efficiency, by assigning to each teams specific non-overlapping tasks.

In order to develop and test MANET applications for the front-end, a complete development environment is required. As part of the development process, it is needed to study alternatives for design and implementation of software modules, analyze possible trade-offs and verify whether specific protocols, algorithms and applications actually work. There exist three way to perform analysis and tests: *(i)* simulation, *(ii)* emulation and *(iii)* field drills.

Simulation and emulation allow to perform several experiments in a cheaper and more manageable fashion than field tests. Simulator and emulator (i.e., hardware and/or software components enabling simulation or emulation) do not exclude each other. Simulation can be used at an earlier stage: it enables to test algorithms and evaluate their performance before starting actually implementing on real devices. Simulators, such as NS2[2] [3], GlomoSim [4] or OMNeT++ [5], allow for several kinds of hardware, through appropriate software modules (such as different device types, like PDAs or smartphones, or networks, like Ethernet or WLAN 802.11). Even if the application code written on top of simulators can be quickly written and performances easily evaluated, such a code must be throw out and rewritten when developers want to migrate on real devices.

The emulators' approach is quite different: during emulation, some software or hardware pieces are not real whereas others are exactly the ones on actual systems. All emulators (for instance, MS Virtual PC or PDA emulator in MS Visual Studio .NET) share the same idea: software systems are not aware about working on an emulated layer (at all or partially). On the other hand, performance levels can be worse: operating systems running on Microsoft Virtual PC work slower than on a real PC with the same characteristics. Anyway, software running on emulators can be deployed on actual systems with very few or no changes.

On the basis of such considerations, in order to develop a complete research environment for MANETs in emergency scenarios, we have designed and developed an emulator, named OCTOPUS [3]. Our emulator is intended to emulate small scale MANETs (10-20 nodes). Instead of making the whole MANET stack virtual, which would require duplication of a large amount of code, we decided to emulate only the

[1]`http://www.workpad-project.eu`.

[2]NS2 enables both simulation and emulation. Here, we refer to NS2's simulation features.

[3]Downloadable at: `http://www.dis.uniroma1.it/~deleoni/Octopus`. At the URL, the reader can also download a user manual.

physical MAC layer, leaving the rest of the stack untouched. OCTOPUS keeps a map of virtual areas that users can show and design by a GUI. Such a GUI enables users to put in that map virtual nodes and bind each one to a different real device. Further, users can add possible existing obstacles in a real scenario: ruins, walls, buildings.

The result is that real devices are unaware of OCTOPUS: they believe to send packets to destinations. Actually, they are captured by OCTOPUS, playing the role of a gateway. The emulator analyzes the sender and the receiver and takes into account the distances of corresponding virtual nodes, the probability of losses as well as obstacles screening direct view[4]. On the basis of such information, it decides whether to deliver the packet to the receiver.

The advantage of OCTOPUS is that, in any moment, programmers can remove it and perform field MANET tests without any kind of change. The aim of this paper is to present OCTOPUS and its novel features. Section 2 investigates existing solutions by taking into account several comparing dimensions, specifically:

Minimal initial effort. The time amount necessary to learn and start using the emulator. Several emulators require to write complex scripts to model channels in detail. We are interested in algorithms for the application layer (and not for the network one), whose performances are only slightly modified by the channel and network parameters.

Portability. This feature gets a twofold meaning: from one hand, it means code to be ported in non-emulated environments with few or no changes. From the other hand, we refer portability as the ability to enable, during emulation, the use of several platforms, such as PCs with Linux or Windows and PDAs with Windows CE or PalmOS.

Handling of Obstacles. The virtual map, which emulator holds, should allow users to insert obstacles representing walls, ruins, buildings. Virtual nodes should move into the map by passing around without going over such obstacles. Movements should be as realistic as possible, according to well-know patterns.

Run-time Event Support. During experiments, destinations of the nodes are required to be defined at runtime, according to the behavior of client applications. Essentially, movements cannot be defined in a batch way; conversely, during emulations, nodes have to *interactively* inform the emulator about the movement towards given destinations.

As of our knowledge, OCTOPUS is the first MANET emulator enabling clients to *interactively* influence changes in the topology, upon firing of events which were not defined before the beginning of the emulation. Other emulators require the specification in batch mode, i.e., when the emulation is not yet started, of which and when events fire.

In addition, OCTOPUS allows to include whichever kind of device, even PDAs or smartphones, and applications, whereas other approaches support only some platforms and applications coded in specific languages. Finally, OCTOPUS supports and handles possible obstacles, packet losses and enhanced movement models, like Voronoi [13]. Section 3 goes more in detail about provided functionalities and used models. Section 4 illustrates the OCTOPUS architecture with some technical details and use experiences. Section 5 concludes the paper, by remarking future developments.

[4]We assume whenever two nodes are not directly visible, every packet sent from one node to the other one is dropped.

2. RELATED WORK

There exist several mobile emulators in the literature, even if they do not provide the features which we need for our intends.

NS2 [3] on its own enables to emulate only wired networks. Anyway, Magdeburg University has developed a NS2's patch to perform wireless emulation [6]. This patched NS2 version can emulate an arbitrary IEEE 802.11 network by connecting real devices to the emulator machine. This solution actually enables to build applications as if the emulator were not present and to switch them working between a real environment and an emulated one without any change. Anyway, it gets some drawbacks: *(i)* client hosts have to be Linux-based and, thus, Windows-based computers or PDAs (the most available ones) cannot be used; *(ii)* it is needed to write complex TCL scripts to set up all emulated aspects of wireless links. So a detailed MANET configuration makes sense when people want to emulate protocols of lower layers and it is important to consider several physical parameters. But in the case where we want to test application software (whose performance and correctness is only slightly affected by such parameters), we would like to easily configure emulated MANETs by a GUI so to minimize initial effort. Moreover, *(iii)* NS2, even patched, does not allow to put possible obstacles on the map. At the most, people can define some Voronoi's paths for node movements to get a similar result, assuming them to be around obstacles. However, we want that two virtual nodes are unable to communicate with each other if they are not in direct sight (e.g., a building is located between them). This is not possible by NS2. Finally, *(iv)* possible events during emulation are decided at batch time in TCL scripts. So, clients cannot affect any change in nodes topology.

Other emulators, such as MobiEmu [7], MNE [8] MobiNet [9], EMWIN [10] and NEMAN [11], show similar problems. EMWIN is one of the few emulators supporting any kind of devices. It works in a distributed fashion: so-called *emulator nodes* are real machines and physically attached to a fast ethernet switch. Emulator nodes can be installed on whichever platform, PCs or PDAs. Every emulator node represents a sort of virtual hub where up to 8 Virtual Mobile Nodes (VMNs) can be connected. Therefore, EMWIN can emulate any platform (PDAs included), even if it does not handle obstacles, nor it allows to insert new events at run-time.

JEmu [12] replaces, for each client, the lowest layer of the communication stack by an emulated one. The emulated stack sends packets to the JEmu server. It decides, taking into account certain information (e.g., distance, collision, etc.) whether the actual destination can receive them (even if ostacles are not handled). If so, the emulator forwards these packets to the JEmu client of the receiver. JEmu is totally written in Java so it works only with Java software. Furthermore, applications need many changes if emulated by JEmu.

Table 1 summarizes the features which we are interested

	Initial effort	Code needs changes?	Platform	Obstacle handling	Run-time event support
Patched NS2	High	No	Linux	Yes (Little)	No
MobiEmu	Low	No	Linux	No	No
MNE	Medium	Yes	Linux	No	No
MobiNet	Medium	No	All	No	No
EMWIN	Low	No	All	No	No
NEMAN	Low	No	Linux	No (but planned)	No (but planned)
JEMU	Low	Yes	All (only Java)	No	No

Table 1: Summary of features provided by some manet emulators

in for OCTOPUS. In this table, "Patched NS2" refers to NS2 enhanced by Madgeburg's patch. Its "Little" obstacle support means that people might define Voronoi's paths for node movements, assuming paths to be around obstacles. The NEMAN entries referred as "planned" are the features which authors will implement in future releases: specifically they plan in future to handle obstacles and to enable applications to influence at run-time links topology.

As you can see from the table, no emulator allows applications to modify at run-time nodes topology. All emulators are based on the same workflow: at design-time, possibly through a GUI, users set up the scenario and a virtual map, binding virtual wireless nodes to real devices, as well as the moments when events fire, such as reaching a given position. After, applications are got running on devices to be emulated. When such a preparation phase finishes, emulation starts and events fire according to the specified schedule. We want to enable the firing of events which were unforeseen during the arrangement of emulation scenarios. In the "real world" the events, such as movements, are caused by users, which interact with applications on board of devices. In general, and especially when testing novel prototypes of application software for MANETs, applications on devices may influence the link topology and nodes motion. Therefore batch emulations might be completely useless. Moreover, obstacles are not handled by other emulators. We think that these aspects are important to make emulations realistic. So, we introduced such novel features in OCTOPUS.

3. FUNCTIONALITIES AND MODELS

OCTOPUS provides functionalities to emulate a wireless local area network by an intuitive and user friendly graphic interface. Main features provided by OCTOPUS are described as follows:

Integrated graphical scenario editor. Emulation scenario setup is fully managed through a GUI and there is no need to know or use any scripting language at all. This choice has been made to allow the average user, even with only basic network knowledge, to focus mainly on the experimental aspects.

Real time node mobility management. In our target experiments, destinations that nodes want to reach, have to be defined at run-time, according to the behavior of client applications. Essentially, movements cannot be defined in a batch way before emulation starts; on the contrary, during emulations, nodes have to somehow inform emulator about their movements towards a given destination[5]. This feature is implemented as a TCP server listening for special "movement" commands sent by software on board of devices. We know this breaks our constraint, which states software on devices do not have to be modified when removing emulator. Anyway, changes, if any, are extremely bounded. Basically they consist in "commenting" invocations to OCTOPUS. In this case we could not avoid to violate it: since those events are generated at run-time by software on devices, only such software can send those commands. However, if we don't need this feature, software on devices actually does not have to be modified when the emulator is removed.

Packets losses. The emulation system supports user-defined packet loss policies, described by a customizable range based function $p_d(r)$. The function $p_d(r) = k$ means the probability of a packet sent by a node to be delivered to a node r-meters far is k. Section 3.2 gives more details.

Obstacle-aware mobility model. Two movement models are available in OCTOPUS: Way-Point and Voronoi [13]. The first one assumes nodes to move straight on the line joining starting and destination point. The latter is more realistic and it takes into account even possible obstacles along the path. The devised algorithm is based by the Voronoi plane tessellation model. Section 3.1 gives more detail about this algorithm.

Broadcast address emulation support. In some algorithms, we may want peers to broadcast a message to every peer in radio range. Since devices are connected through a real LAN[6], we cannot use the normal broadcast IP address (i.e., `x.x.x.255`), as it would send the packet to all peers in the network without considering the routing table. We want to broadcast only to virtual neighbors. This issue is resolved by adding a customizable virtual broadcast address instead of the usual one.

Node follow feature. Once emulation is started, a node can move in emulated environment so to "follow" another node. This is intended to study some algorithms in our project, such as disconnection handling in MANETs [2]. In OCTOPUS, when a node A "follows" another node B, that means A goes after B when B moves, keeping at least a fixed minimum distance.

In the following, some details of the models implemented in OCTOPUS are provided.

[5]This makes sense when behavior of client applications is controlled by humans.

[6]OCTOPUS and other actual devices have to be deployed in the same LAN in order to have OCTOPUS to be able to reach other devices.

3.1 Voronoi Mobility Model

In order to develop a realistic mobility management, the nodes, living in the emulated environment, move avoiding obstacles. As a matter of fact, humans follow predefined paths to reach a place, such as roads and sidewalks: emulated environments should show similar behaviors. OCTOPUS allows to define polygonal obstacles in the virtual map and it generates the graph of all possible segments of paths that do not cross them. The algorithm we have devised derives from Original Voronoi algorithm. Original Voronoi assumes to have a given set P of points in the plane and builds some special lines. Voronoi's lines describe closed polygons in the plane. Each polygons includes exactly one $p_i \in P$ of the original points. For each p_i, the corresponding polygon contains all points which are closer to p_i than other $p_j \in P$.

Since obstacles are polygons and not simply points, a generalization is needed:

1. Generate a "sampled" version of every obstacle by sampling every side of every obstacle and replacing each one with a sequence of points. The sampling rate can be defined by users.

2. Generate Voronoi diagram by considering points generated at point 1.

3. Remove segments crossing one or more obstacles. That means all segments having at least one of the two vertices inside an obstacle are removed.

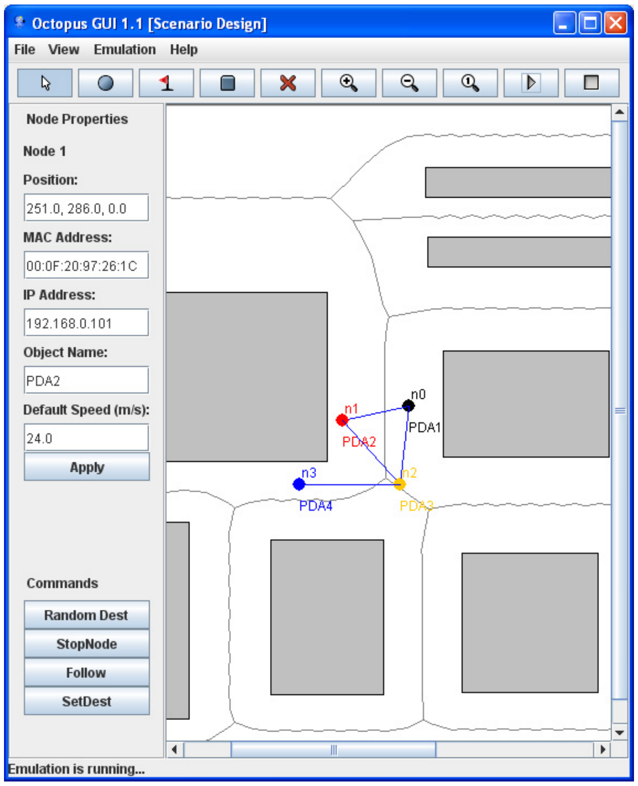
OCTOPUS Voronoi diagram is computed as dual of *Delaunay triangulation* [14], as it gets actually lower realization complexity. Each segment generated by the Voronoi algorithm represents a possible part of the path that nodes are forced to follow in order to move without crossing an obstacle.

3.2 Packet Loss Models

A probability-based packet loss policy is provided. In real WLANs, every sent packet may be lost due to the unreliability of the physical channel. In order to provide a realistic emulation, OCTOPUS supports a probabilistic approach to determine whether or not a packet has to be dropped (so to emulate loosing). In order to let advanced users to define their own loss function $p_d(r)$, we left it opened and customizable[7].

Since obstacles are present in the virtual area, we assume radio signal do not pass through obstacles; this means that each packet sent by a node to another is surely dropped if the couple of nodes is not in direct sight. In the real world, a wireless device may measure its distance to the others by *signal to noise ratio* (SNR) techniques: the higher is the physical distance, the higher is the noise in communication channel and, hence, SNR. However, that gives an approximate "communication distance" between two peers: this method does not give the exact physical distance for other factors, such as thin obstacles among devices or other interferences, which can cause noise incrementing. So, *communication distance* d_c^{em} and *real distance* d_r^{em} may differ. It is too difficult (and perhaps even impossible) to emulate

[7]For instance, users can model perfect reliability by defining $p_d(r) = 1\ \forall r \in [0, rrange]$, where $rrange$ is the radio range of the specific transmission technology, e.g., 100 meters for IEEE 802.11b/g and 10 meters for Bluetooth.



Position: 251.0, 286.0, 0.0

MAC Address: 00:0F:20:97:26:1C

IP Address: 192.168.0.101

Object Name: PDA2

Default Speed (m/s): 24.0

Emulation is running...

Figure 1: An OCTOPUS screenshot.

physical factors affecting communication distance. Therefore, OCTOPUS define communication distance between two nodes a and b as follow:

$$d_c^{em}(a,b) = \begin{cases} d_r^{em}(a,b) & \text{if nodes are in direct sight} \\ \infty & \text{if at least one obstacle divides} \\ & a \text{ and } b \end{cases}$$

The probability to deliver a packet is given by evaluating the user defined loss function where input is d_c^{em}. So, the probability to deliver to a node b a packet sent by a node a:

$$p_{a,b} = p_d(d_c^{em}(a,b)) \in [0,1]$$

When a wants actually to send a packet to b, OCTOPUS computes $p_{a,b}$. Then, it generates a random value $x \in [0,1]$ from an uniform distribution. Finally, OCTOPUS follows the rule "`if` $x \leq p_{a,b}$ `then deliver else drop`" to decide whether such a packet has to be delivered or dropped.

4. DESIGN OF THE OCTOPUS ARCHITECTURE AND USE EXPERIENCES

OCTOPUS is completely written in Java; in particular, it has been tested to work fine both on Windows and on Linux. The OCTOPUS architecture relies basically on four modules:

Environment Manager. It is the core module and the OCTOPUS' behavior depends on its setting. Users can setup several parameters, such as area size, node positions, radio ranges and obstacles. It also computes the Voronoi's graph. This module is used by the *Gateway*

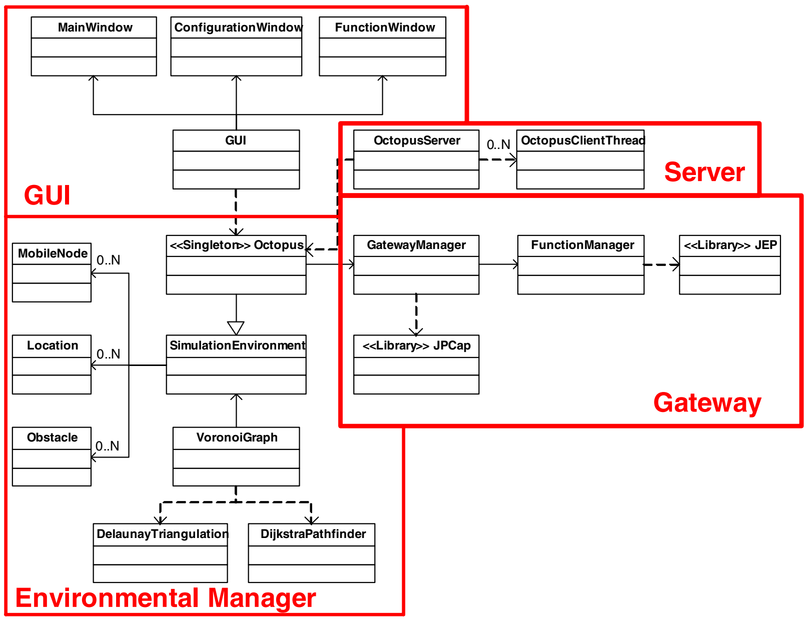


Figure 2: The OCTOPUS' class diagram

module to get information to learn whether a packet has to be delivered or not.

Gateway. OCTOPUS plays the role of a gateway: this module intercepts all packets sent by nodes involved in the emulation and addresses every network issue. It decides whether to forward by taking into account distance information from the *Environment Manager*. The *Gateway* module implements the packet dropping policy described in 3.2.

Server. This module implements the TCP Server, listening on the `8888` port. Such a server is intended to receive command from applications about events (like movements) to trigger and to reply to queries coming from clients. For instance, a client could ask which are neighbors or which is the distance from them. The communication protocol is a trivial textual protocol. We have also realized a C# module masking socket accesses behind an easy API.

GUI. In order to minimize the effort to setup initial scenario and bind virtual nodes to the actual devices, OCTOPUS is provided with a Graphical User Interface. It enables to perform any configuration aspect in a friendly fashion, without having users to learn any special scripting language. At design-time users can insert in the virtual area nodes, obstacles and buildings by "point-and-click", as in any drawing software. GUI allows users to load/save scenarios and settings from/to XML files without having to setup every time scenarios from scratch. At run-time, GUI shows the exact position of virtual nodes in the maps. Figure 1 depicts an OCTOPUS screenshot: the right panel shows the virtual area, whereas the top part is used at design-time to configure scenarios (nodes, positions, etc.) The left panel describes the nodes mappings and other information, allowing, also, users to change position of nodes by firing manually some events. The gray rectangles and lines represent, respectively, obstacles and Voronoi lines, which nodes follow during motions. If a proper option is active (as it is in the figure), the GUI shows virtual neighbor nodes by a blue line connecting each couple of nodes in radio-range. Another option enables the GUI to design a circle centered in every node to show the radio range.

4.1 Implementation Details

Figure 2 shows the classes composing OCTOPUS and classifies them with respect to modules described above:

Environmental Manager. `Octopus` class is singleton (i.e., at most one instance may exist) and derives from the `SimulationEnvironment` class. `SimulationEnvironment` describes the physical environment to be emulated and manages also the mobility aspect by `VoronoiGraph` class. `SimulationEnvironment` class contains a list of `MobileNode`, `Location` and `Obstacle` instances in order to get a complete environment description. Since Delaunay triangulation is dual of Voronoi but computationally more efficient, a `DelaunayTriangulation` class is used by `VoronoiGraph` class. `DijkstraPathFinder` class is used by `VoronoiGraph` to compute a path from a source point to a destination.

Gateway. The network level is managed by the `GatewayManager` class that uses the JPCap library [15] in order to capture and forward LAN packets. To evaluate whether a packet has to be delivered or lost, the `GatewayManager` is supported by `FunctionManager` that makes use of the JEP library in order to parse a user-defined loss function.

Server & GUI. The OCTOPUS TCP/IP server is multi-threaded and implemented by the `OctopusServer` class. It is multi-threaded as it manages multiple connections at the same time: each connection is handled by a different `OctopusClientThread` object.

4.2 Use Experiences

In order to verify that the emulator is faithful to MANETs being emulated we simply made use of the PING utility: assuming a certain function $p_d(r)$ for packets loss, we experienced actual percentage of replies to ping requests to deviate from theoretical value at most for 4%[8].

Moreover we have used it to test a Bayesian approach for disconnection prediction in MANETS [17]. OCTOPUS was proved to be useful to test such an algorithm. Most of the devices were PDAs: in experiments they were sending commands to OCTOPUS to instruct it to move corresponding nodes in the virtual map. Movements took place, according to unforeseen events, which were caused by run-time interactions among nodes. Besides commenting few lines, this module is perfectly working on real scenarios.

5. CONCLUSION AND FUTURE DEVELOPMENTS

MANETs are focus of intense research nowadays, due to their envisioned applicability in highly dynamic scenarios, such as emergency management. Research on MANETs has, up to know, mainly concentrated on the development of appropriate routing protocols, methods for energy preservation, and other issues on the lower four ISO/OSI layers. It is obvious that virtual environments supporting this research are more targeted towards low-level issues and details, as shown in Section 2. But currently, in order to move the research towards the application layer (e.g., for studying adaptive workflow management systems for MANETs, or context-aware applications, or collaboration miners), the availability of emulation virtual environments is needed. Such environments should allow to use real devices (including commercial PDAs) during the emulation, to define virtual maps including obstacles and different movement models, and to have *interaction* between the software to be tested and the emulation environments.

Current MANET emulators fail in most of these requirements. Conversely OCTOPUS, the emulator presented in this paper, has been designed with these requirements as target. Through a couple of applications and experiments, we have shown the advantages, for the researchers in the field, stemming from the availability of OCTOPUS.

OCTOPUS can be used in every scenario where MANET emulation is needed. As the only requirement is an IP network, it can be even used in robotic planning and in sensor networks. All described features and models have already been implemented in the version that is downloadable from the site.

As future work, we plan to enhance the tool, by adding more movement models. In order to make emulation more realistic, we plan to model the wireless channel as exactly as possible. We are investigating on using the Ricean fading [16]. Moreover, we plan to use extensively OCTOPUS as test bed platform for the European project WORKPAD, which we are currently involved in.

[8]The reader can find more details about the PING experience at `http://www.dis.uniroma1.it/∼deleoni/PingExperience.pdf`

6. ACKNOWLEDGEMENT

This work has been partly supported by the Italian MIUR through the FIRB 2001 project *MAIS*, and by the European Commission through the FP6-2005-IST-5-034749 project *WORKPAD*.

7. REFERENCES

[1] T. Catarci, F. De Rosa, M. de Leoni, M. Mecella, S. Dustdar et al. WORKPAD: 2-Layered Peer-to-Peer for Emergency Management through Adaptive Processes. In *Proc. 2nd International IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2006)*, 2006

[2] M. de Leoni, F. De Rosa, M. Mecella. MOBIDIS: A Pervasive Architecture for Emergency Management. In *Proc. 4th International Workshop on Distributed and Mobile Collaboration (DMC 2006) @ WETICE 2006*, 2006.

[3] The Network Simulator – ns-2. `http://www.isi.edu/nsnam/ns/`

[4] X. Zeng, R. Bagrodia, M. Gerla. GloMoSim: a Library for the Parallel Network Simulation Environment. In *Proc. 12th Workshop on Parallel and Distributed Systems.* 1998.

[5] OMNet++ – Discrete Event Simulation System. `http://www.omnetpp.org/`

[6] D. Mahrenholz, S. Ivanov. Real-Time Network Emulation with ns-2. In *Proc. 8th IEEE International Symposium on Distributed Simulation and Real Time Applications*, October 2004.

[7] Y. Zhang, W. Li. An Integrated Environment for Testing Mobile Ad-Hoc Networks. In *Proc. 3th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'02)*, June 2002.

[8] J. P. Macker, W. Chao, J. W. Weston. A low-cost, IP-based Mobile Network Emulator (MNE). In *Proceedings MILCOM 2003 - IEEE Military Communications Conference*, 2003.

[9] P. Mahadevan, A. Rodriguez, D. Becker, A. Vahdat. MobiNet: A Scalable Emulation Infrastructure for Ad hoc and Wireless Networks. In *Proc. 5th International Workshop on Wireless Traffic Measurements and Modeling*, 2005.

[10] P. Zheng, L. M. Ni. EMWIN: Emulating a Mobile Wireless Network using a Wired Network. In *Proc. 5th ACM International Workshop on Wireless Mobile Multimedia*, 2002.

[11] M. Puzar, T. Plagemann. NEMAN: a network emulator for mobile ad-hoc networks. In *Proc. 8th International Conference on Telecommunications (ConTEL 2005).*

[12] J. Flynn, H. Tewari and D. O'Mahony. Jemu: A Real Time Emulation System for Mobile Ad Hoc Networks. In *Proc. of 1st Joint IEI/IEE Symposium on Telecommunications Systems Research*, 2001.

[13] A. Jardosh, E.M. BeldingRoyer, K.C. Almeroth, S. Suri. Towards Realistic Mobility Models For Mobile Ad hoc Networks. In *Proc. of MobiCom 2003.*

[14] L. Guibas, J. Stolfi. Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. In *ACM Transactions on Graphics, 4:74-123*, April 1985

[15] JPcap: Java package for packet capture, `http://netresearch.ics.uci.edu/kfujii/jpcap/doc/`

[16] R.J. Punnoose, P.V. Nikitin, D.D. Stancil, D.D. Efficient simulation of Ricean fading within a packet simulator In *52th IEEE Vehicular Technology Conference*, 2000.

[17] M. de Leoni, M. Mecella, R. Russo A Bayesian Approach for Disconnection Management. In *Proc. Workshop on Interdisciplinary Aspects of Coordination Applied to Pervasive Environments: Models and Applications (COMA 2007) @ WETICE 2007*, 2007.