A scalable workbench for implementing and evaluating distributed applications in mobile ad-hoc networks[*]

Johannes K. Lehnert , Daniel Görgen, Hannes Frey and Peter Sturm
System Software and Distributed Systems
University of Trier, Germany
{lehnert|goergen|frey|sturm}@syssoft.uni-trier.de

"**Keywords**" multihop ad-hoc network

Abstract

This work presents a Java-based development platform aimed to ease the task of building applications for mobile multihop ad-hoc networks. The platform follows a three-tier development principle composed of simulation, emulation and deployment on real mobile devices. Opposed to pure network simulators, this development environment primarily focuses on an easy to use event-based programming model and scalability regarding simulating thousands of mobile devices. Additionally, utmost code reuse is provided, since attaching real hardware to the simulation and running the application on real devices are an integral part of the workbench. Performance evaluation by means of a benchmark application demonstrates that simulating over ten thousand mobile devices can be performed faster than in real-time. Also experiences gained from implementing a mobile auction system for ad-hoc networks proved that the integral parts for emulating and deployment are of high value when building real life applications for mobile multihop ad-hoc networks.

INTRODUCTION

Future paradigms such as pervasive computing and ubiquitous computing [1] are characterized by countless numbers of small and thus mobile devices that communicate with neighboring peers using wireless transmission techniques. Groups of these mobile devices may form highly dynamic ad-hoc networks at any given time. The number of hops required in these networks to reach a given destination or to enter a wireful and more reliable backbone structure is used to distinguish between at least two different classes of networks. Single-hop ad-hoc networks such as cellular phone networks, WLAN infrastructures based on access points or Bluetooth piconets are state of the art. In contrast to this, spontaneous multihop networks with high device mobility and frequent fluctuation are still a research issue. Since it is expensive for any mobile device within such a multihop network to obtain reliable global information, self-organization as the primary design principle is essential to cope with the anticipated frequency of change. As a consequence, decisions within a mobile device can only be based on local information, e.g. the state of the device itself and its current neighborhood. In such an environment, most of the goals of a distributed mobile application are achieved by synergy through altruistic behavior of all involved devices.

Successfully evaluating self-organizing applications for multihop ad-hoc networks requires more mobile devices than can be provided at the moment. Additionally, the number of volunteers needed to run these experiments is hard to obtain even in a university environment and they are even harder to orchestrate for the sake of reproducible scientific results. A promising approach is to provide a uniform workbench supporting experiments ranging from pure simulation of several thousand mobile devices, over hybrid scenarios with interaction among simulated as well as real life devices up to dedicated field trials as proofs of concept. From this uniform workbench, a development process can be derived that starts with implementing, testing, and evaluating algorithms and applications in a purely simulated environment first. In subsequent steps, dedicated mobile devices can be cut out of a simulation run and be transferred to a real mobile device in order to deal with real user interaction and to evaluate the user experience. In a final phase, specific field trials can be defined and executed on a number of mobile devices. The hybrid nature of this approach leads to the additional requirement, that the simulator must be capable to achieve real-time execution behavior even in the case of several thousand mobile devices.

In the remainder of this paper, a Java-based implementation of such a uniform workbench for multihop ad-hoc networks is presented. The next section starts with a discussion of related work. In the following section 3, the simulator of the workbench is presented in detail. Emphasis will be put on parts of the simulator core that have been modeled explicitly for extension and adaption to specific mobil-

[*] This work is funded in part by DFG, Schwerpunktprogramm SPP1140 "Basissoftware für selbstorganisierende Infrastrukturen für vernetzte mobile Systeme"

ity models, different dynamic aspects of ad-hoc networks as well as the visualization of simulation results. By measuring benchmark simulations, it is shown in section 4 that the simulation of several thousand mobile devices in real-time is possible, which was a primary design goal. In section 5, the remaining two parts of the workbench, hybrid simulation and support for execution on real hardware, are introduced briefly. Finally, in section 6 conclusions drawn from experiences with a self-organized auction system for ad-hoc networks are discussed.

RELATED WORK

This work presents a development environment for mobile applications running in a large scale mobile multihop ad-hoc network. The environment divides the design process of mobile applications in three parts, simulation, emulation and execution on real devices.

The CMU wireless extension to ns2 [2] and GloMoSim [3] are the commonly used tools for simulating protocols in mobile multihop ad-hoc networks. Both simulation environments focus on a detailed simulation of protocol layer 1 to 4. This high level of detail increases the computational complexity of the simulations. Thus, only a small number of devices can be simulated in acceptable time.

Emulation is used in order to allow user interaction and to allow existing applications to be tested in a simulation environment without the need of managing hundreds of real devices. An emulation platform using ns2 is proposed in [4]. It is an extension of the Vint/NS simulator [5] for emulation of wired networks using the wireless extension to ns2 [6]. This emulation platform scales well up to about 100 devices simulated by ns2. Other emulation approaches forego the simulation of mobile devices but emulate only node mobility and the resulting network behavior. Mobility data can be created synthetically as proposed in [7] or captured from real life test runs as done in [8].

Network experimentation using a testbed is useful to prove the applicability of a protocol or application in a real life scenario. Ad-hoc network examination in such experiments were performed by [9] and [10] or [11] for single hop networks. An obvious disadvantage of using a testbed to examine protocol or application characteristics is that results are not reproducible and that it takes great efforts to manage the large number of real mobile devices. In order to allow more reproducible results [12] proposes users walking around by following the instructions of a scenario script running on each mobile device.

SIMULATION

The main focus of the simulation platform presented in this paper is to ease the development and simulative evaluation of applications for mobile multihop ad-hoc networks. A high abstraction level based on an object-oriented design in Java enables the developer to concentrate on application design without worrying about technical details of the simulator. Instead of aiming to simulate the lower network layers as exactly as possible, the focus is on the simulation of the topological properties of the ad-hoc network. The combination of a high abstraction level and the limitation on the topological properties reduces the computational complexity of the simulation and allows simulating a high number of devices while maintaining short simulation times.

Additionally, the high abstraction level allows implementing the same abstractions in the simulation environment as well as on real hardware platforms. Applications developed and tested in the simulation environment can easily be transferred to a real hardware platform (See section 5). Thus, field trials are based on well-tested code. Traditional simulators, in contrast, require a complete rewrite of the application in order to port it to a real hardware platform.

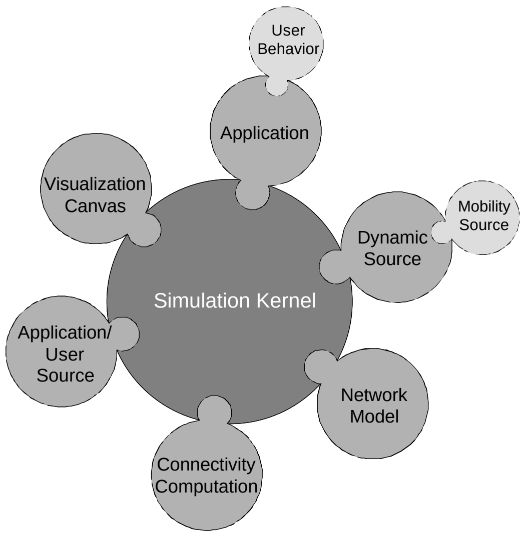

Figure 1. Architecture of the simulation environment: a simulation kernel extended by pluggable components.

Figure 1 shows the architecture of the simulation environment. A discrete-event simulation kernel is extended by multiple components. These components are based on generalized interfaces and thus are exchangeable. Developers can use the default implementations provided by the simulation environment or use their own implementations. In particular, these pluggable components are:

Application: The application to simulate.

User Behavior: The simulated user behavior used to trigger actions in the simulated application.

Mobility Source: Provides mobility information for all mobile devices in the simulation.

Dynamic Source: Provides information on all other dynamic aspects of the devices in the simulation

Network Model: The implementation of the wireless communication network in the simulation.

Connectivity Computation: Calculates the connectivity between the devices in the simulation.

Application/User Source: Provides the coupling of devices, applications and user behavior.

Visualization Canvas: Implements a visualization method for the simulation.

All these components are described in detail in the following subsections.

3.1 Mobility models

The simulation uses a simple but effective abstraction for mobility models. A mobile device enters the simulation area at a given point in time and remains in the simulation area until no more mobility data is available. After entering the scenario devices repeatedly move directly from one point to another within a given period of time. The velocity of a device is constant during such a move, but can be changed for each new move. Choosing an identical starting point and endpoint stops the device for the duration of the "move". This abstraction allows the implementation or approximation of arbitrary mobility models. Additionally, it supports the optimized calculation of connectivity data as shown in subsection 3.3.

Mobility information in the simulation following this abstraction is provided by *Mobility Sources*. The simulation kernel queries the mobility source (through the dynamic source) for mobility information and executes the corresponding movements. If no more mobility information for a device is available, the device is removed from the simulation.

Mobility sources are exchangeable and their implementation is not determined. This leads to flexibility because mobility information may be computed on-the-fly but also read from a file as well. Thus, the usage of external mobility generators like BonnMotion [13] is fairly easy.

3.2 Further dynamic aspects

The wireless communication technology leads to additional dynamic aspects (actions) of the mobile devices. Uni- and bidirectional connections between mobile devices are established and closed whenever devices enter or leave their mutual sending ranges. The link reliability of an existing connection may change over time and devices might change their sending power.

All these dynamic aspects are provided by *Dynamic Sources*, together with the mobility information. On request of the simulation kernel the dynamic source provides the next mobility information or the next dynamic action of a device. As with mobility sources the implementation of dynamic sources is not determined and they are exchangeable. Dynamic actions may be computed on-the-fly, read from a file or result from user input.

Dynamic aspects are reusable for identical scenarios even if the simulated application and/or its simulated user behavior are changed. This reduces the simulation time significantly since the expensive connectivity computation is avoided.

3.3 Connectivity computation

Computing the connectivity between mobile devices is a computationally intensive task in the simulation. The simulation uses proactive connectivity computation. The connectivity between all mobile devices is computed in advance and independently of the network communication. Therefore, this information is available all the time.

The *Link Calculator* component computes all connectivity information in the simulation and acts as a dynamic source. It uses the properties of the mobility model abstraction to compute the connectivity information efficiently. Since all devices move with constant velocity on straight lines, a device starting a new move is able to compute all connectivity events for this move. The mobile device compares its move with all actual moves of other devices in the simulation and computes if and when the distance between two devices creates or breaks a link by solving the corresponding linear system of equations. The device computes the connectivity events for both devices. Combined with a special handling for devices entering and exiting the simulation this method allows to compute the connectivity information efficiently. Instead of computing connectivity information in fixed time intervals this information is computed once for the complete movement.

3.4 Network models

The network model of the simulation is based on two communication primitives: local unicast and local broadcast. Local unicast provides communication with one device in the sending range while local broadcast allows communicating with all devices in the sending range. Connectivity information provided by the dynamic source is used by the network implementation. Actions indicating a new connection are interpreted as "potential new connections", thus enabling the network implementation to ignore the connection. This allows the implementation of simple network models as well as complicated statistical models, which take into account the device density and other disturbances of the wireless network. Even very detailed network models as the IEEE 802.11 model of ns-2 could be mapped to the network model abstraction.

Messages sent over the network may be traced: the sender is notified when the message leaves the local queue. Additionally, network implementations may provide notifications about successful, unsuccessful or undefined delivery as well.

Applications in the simulation never use a network implementation directly but use a collection of network protocols instead. Network protocol implementations use the two communication primitives of the network model to implement complex communication protocols.

In order to speed-up application and protocol development, messages in the simulation are standard Java objects implementing a message interface. These objects "know" which method to call at the receiver and thus can be handled generically in the network model implementation. Messages do not need to be space-optimized since a special size attribute allows the realistic simulation. This eases the development of new message types.

Application messages are wrapped in protocol messages when they are sent using one of the network protocols. At the receiver side the protocol message is handled and triggers the handling of the application message if necessary. Message objects must be cloned when they are sent to other devices in the simulation to preserve the inner state of the messages when multiple devices receive the same message. The developer can disable this cloning for stateless messages, thus allowing faster simulation runs due to a significantly reduced number of objects.

3.5 Application interface

The simulation environment allows one application per mobile device and uses *Application User Sources* to handle the assignment of applications and devices as well as the definition of the simulated user behavior. Application User Sources provide the simulation kernel on request with a tuple of application and user behavior. By implementing a suitable Application User Source, the developer is able to define exactly which application is executed on each device. The separation of the simulation scenario, i.e. mobility, simulation area, sending ranges etc., from the simulated application and the user behavior allows using precomputed connectivity and mobility data to speed-up the simulation.

The application has no direct access to the mobile device, but uses the operating system abstraction of the simulation kernel to get information about the device. The operating system makes available the current time, the position and the moving direction of the device. A neighbor discovery service informs the application whenever another device enters or leaves the communication range of the device running the application. Additional information like the current position and direction of other devices are available from the neighbor discovery service as well.

3.6 User behavior

The simulated user behavior has access to the application and thus can trigger application events. Additionally, the simulated application can trigger actions of the simulated user behavior as well.

3.7 Visualization

The visualization architecture of the simulation allows the visualization of all simulation components. It is based on shapes like rectangles, ellipses, images, text and polygons, which draw themselves on a canvas. Due to this generic approach the implementation of the visualization can be varied and is not limited to windowing systems. Visualization backends like Postscript or movie files are possible. The current version provides canvas implementations using Java 2D, SWT [14], OpenGL [15], XML and Postscript.

The simulation allows the visualization of the simulation area, the mobile devices, the network, network messages, protocols and applications. By defining shapes or collection of shapes for these components, the developer can define the needed level of detail. The visualizations of the components are placed on different layers. The order of these layers may be specified by the developer as well.

Using the visualization is computationally expensive and slows the simulation down. But often the visualization provides insights which would be very hard to obtain by evaluating traditional trace files. Additionally, the visualization can be disabled and causes no delay in this case.

3.8 Statistics

The evaluation of simulation runs is simplified by a statistics system. This statistics system allows the developer to collect and evaluate local as well as global data during simulation runs. All collected data may be saved in a report after the simulation run is finished. Additionally, live visualization of the statistical data is possible.

PERFORMANCE EVALUATION

In order to allow a better evaluation of the scalability and performance of the simulation environment, a sample message dissemination application has been tested. This application was simulated with the Random Waypoint Mobility Model [6] and a simple network implementation for 10 minutes. The sending radii of the mobile devices are uniformly distributed between 10 and 100 meters and the moving speed of the mobile devices is uniformly distributed between 0.8 and 1.0 meters per second.

Three series of measurements were done on a Pentium IV (HT) with 3 GHz and Java 2 SDK 1.4.1 (IBM). The first one measured the execution speed for an increasing number of devices on a fixed simulation area of 500m x 500m, thus increasing the average device density in each step. The second measurement increased the size of the simulation area while increasing the number of devices, thus keeping the average device density the same. Table 1 and 2 clearly show that device density is an important factor for the performance of the simulation environment. The third measurement increases the device density just as the first, but also increases the network load. An increasing number of devices flood a 50m radius region around them with a constant bit rate (one data package every 4s, rebroadcast by every device in the area).

In case of a constant device density the simulation environment scales well with the number of devices, because the number of links per devices does not increase. In all cases the execution time is slightly quadratic in the number of the devices, because all devices has to be compared with every other. But in the precomputed constant device density case this is only linear due to the fact, that the links per device is constant. In both cases the real-time simulation of a high number of devices is possible: for the constant device density a simulation of 8000 more than twice as fast as real-time is possible.

If the mobility and connectivity data is precomputed, the execution times decrease significantly. Especially in the case of constant device density the performance gain is substantial. A simulation run with 3200 mobile devices is finished nearly 20 times as fast (243.5s instead of 4784.1s). For the measurement with increasing device density the execution time is reduced by 70% (1173.9 instead of 3797.7).

Using a Java2D canvas implementation, visualization in real-time (or even faster) is possible up to 3200 devices in the case of increasing device density. For constant device density the maximum number of devices for a real-time visualization is 8000. Further speed-ups are possible by using precomputed mobility and connectivity data.

Table 1. Execution times for for the example simulation on a simulation area of 500m × 500m.

#devices	Execution time (seconds)	Execution time (precomputed) (seconds)
50	0.52	0.87
100	0.76	1.08
200	1.30	1.61
400	3.31	3.69
800	11.77	11.97
1600	47.90	40.72
3200	245.13	167.30
6400	3797.74[1]	1173.89
12800	n/a[2]	n/a

Table 2. Execution times for the example simulation with an average device density of 0.002 devices per m^2.

#devices	Simulation area (meters)	Execution time (seconds)	Execution time (precomputed) (seconds)
125	250 × 250	1.24	1.81
250	250 × 250	2.20	2.81
500	500 × 500	4.78	5.23
1000	707 × 707	11.69	10.12
2000	1000 × 1000	29.50	19.87
4000	1414 × 1414	77.05	35.57
8000	2000 × 2000	233.10	66.32
16000	2828 × 2828	807.87	118.73
32000	4000 × 4000	4784.08	243.48

Increasing the network load linearly causes only linear increasing execution times. After omitting the link calculation times (0 sending devices) doubling the network load increases the execution time only by the factor 1.7-2.2. The data rate (10240 bytes/s) and the sending data rate (128 bytes/s) were chosen so that the network is nearly saturated in the case that 640 of 3200 devices are sending. All results of this measurement are depicted in table 3.

Table 3. Execution time (seconds) for increasing device density and increasing network load.

Devices sender	50	100	200	400	800	1600	3200
0	0.47	0.61	0.96	2.49	9.73	40.50	210.89
10	0.76	1.08	1.58	3.94	16.61	62.68	311.48
20	0.90	1.15	2.00	4.87	21.64	82.60	399.88
40	0.94	1.36	2.54	7.53	30.62	129.91	558.92
80	x	1.79	3.93	12.57	53.62	231.52	982.71
160	x	x	6.70	23.85	95.68	444.40	1936.53
320	x	x	x	44.29	190.65	861.43	3882.74
640	x	x	x	x	394.41	1703.27	7662.28

EMULATION AND REAL HARDWARE

Based on the high abstraction level of the simulation environment two further evaluation environments are provided. The first one, a hybrid simulation platform, enables the emulation of applications developed for the simulation environment while the second one allows the execution of these applications on real hardware.

5.1 Hybrid simulation platform

The hybrid approach allows real clients to be attached to a running simulation using RMI over a network connection. Mobile devices using a wireless connection as well as workstations in a traditional fixed network may act as clients in this approach. This enables users to interact with the simulated application, thus replacing the simulated user behavior with more realistic input. The visualization of the simulation allows the visualization of the global simulation state regarding the simulated application. Even though the application is running in a simulation environment, nevertheless one gets a feeling about the application running in real life.

Using the hybrid simulation platform frees the developer from testing his application directly in field trials. Instead of having to configure lots of devices and organize a field trial, he can use a network of workstations and still gain insights into the behavior of the application in a real world scenario. Additionally, the hybrid simulation platform allows the easy demonstration of applications.

The hybrid simulation platform is based on a client/server architecture. Simulation server and simulation control are running on the server side. The simulation server acts as a RMI server with the simulation clients registering as RMI clients. Simulation control runs the simulation platform in a second thread and passes user input from the simulation clients to the simulation platform and vice versa. The core simulation is not aware of external clients,

[1] Unlike the other simulation runs the also tested Sun JVM performed better here and needed only 2686.99 seconds.

[2] The available RAM (1792 MB) of the machine used for the measurement was not sufficient to run the simulation. This is due to the high device density and the resulting number of events.

since by using an external user object, input and output regarding a certain client is treated in the same manner as for a common simulated user. Thus, there is no need to change any code from the pure simulation to the hybrid simulation platform.

The simulation client is responsible to provide a frontend for the user to control its assigned mobile device and application. In general, it consists of an application GUI replacing the simulated user behavior and providing information about the application state. When porting an application from the pure simulation to the hybrid simulation platform, this is the only part where additional code has to be written.

5.2 Execution on real hardware

The hardware execution platform provides exactly the same interfaces as the simulation platform. A hardware abstraction layer provides all functionality of the simulated platform such as network communication, neighbor discovery, positioning and the same event-based programming model including system timers and message events. With these preconditions the application code of a simulated application can be used without any modifications. Solely the simulated user behavior is replaced by a GUI which enables the user to interact with the application running on the mobile device. If the hybrid simulation platform has been used to evaluate the application, the GUI from this evaluation can be reused.

The current implementation of the hardware execution platform is implemented for Pocket-PC or notebooks running a JVM. Devices receive positioning information using GPS [16] units and wireless communication is done over IEEE 802.11 [17]. The network model of the simulation platform is implemented using UDP/IP unicasts and broadcasts. One hop communication between devices is mapped to these primitives directly. All higher level communication protocols like topology-based routing are implemented in the simulation platform and can be used without modifications because they use the network model interface of the simulation platform. Messages sent by applications are serialized with the standard Java serialization mechanism and sent over the network. If needed, this could be replaced with a more efficient method.

Neighbor discovery is implemented with a beaconing protocol. A beacon contains the device address, its current position and its moving direction. Each device broadcasts beacons in regular intervals. To detect bidirectional connections, devices respond with an empty unicast message when they receive a beacon. Neighbor information is cached and provided to the application as needed.

At the moment the execution platform solely supports GPS to receive positioning information, but other positioning services such as [18, 19] would also be possible. The spherical (longitude/latitude) coordinates of GPS are mapped to the Cartesian coordinates of the mobility model of the simulation platform by using a Gauss-Krüger projection with a shifted zero-point.

Simulated applications are inherently event-driven. Thus, the execution platform must provide the same event-driven programming environment. This is achieved by multithreading and operating system timers. The singlethreaded nature of the simulated application code is preserved with multiple threads and local event queues. Contrary to the simulation, events on the execution platform have no ordering, therefore no assumptions should be made in case of contemporaneity of independent events.

CONCLUSIONS

The simulation environment presented in this paper provides the developer of applications for mobile multihop ad-hoc networks with a scalable and comfortable evaluation platform. Developers are supported by a high abstraction level, a suitable programming model and a powerful visualization interface. Furthermore, the simulation environment allows the efficient simulation of a high number of mobile devices.

Applications developed in the simulation environment can be transferred to a hybrid simulation environment and a real hardware platform without modifications in the application code. This accelerates the evaluation process significantly, since well-tested code is used in the emulation as well as in field trials.

UbiBay, an auction system for mobile multihop ad-hoc networks, has been implemented using the simulation platform. UbiBay uses a middleware based on the marketplace communication pattern [20] to realize a completely self-organized auction system. Based on the first implementation of the auction system in the simulation environment, UbiBay was evaluated both in the hybrid simulation environment as well as on real hardware (HP iPAQs with IBM J9 VM). Porting the application from the simulation environment to the hybrid simulation environment proved to be trouble-free. Only a graphical user interface had to be implemented to allow the interaction of human users with the application. Moving UbiBay from the hybrid simulation environment to the real hardware platform required little effort since the graphical user interface could be reused and needed only minor modifications.

Current work involves the implementation of a more realistic statistical network model resembling the properties of IEEE 802.11. Additionally, a broader set of mobility models for the simulation environment is implemented to provide developers with a broader choice.

REFERENCES

1. Weiser, M.: The computer for the 21st Century. Scientific American 265 (1991) 94-104
2. Fall, K., Varadhan, K.: The ns manual. The VINT Project - A collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC (1989-2003)

3 Zeng, X., Bagrodia, R., Gerla, M.: GloMoSim: A library for parallel simulation of large-scale wireless networks. In: Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS-98), Los Alamitos, IEEE Computer Society (1998) 154-161

4. Ke, Q., Maltz, D., Johnson, D.B.: Emulation of multi-hop wireless ad hoc networks. In: The 7th International Workshop on Mobile Multimedia Communications (MoMuC 2000). (2000)

5. Fall, K.: Network emulation in the VINT/NS simulator. Proceedings of the fourth IEEE Symposium on Computers and Communications (1999)

6. Broch, J., Maltz, D., Johnson, D., Hu, Y.C., Jetcheva, J.: A performance comparison of multi-hop wireless ad hoc network routing protocols. In: Proc. of the 4th ACM/IEEE Int. Conf. on Mobile Computing and Networking (MobiCom'98). (1998) 85-97

7 Zhang, Y., Li, W.: An integrated environment for testing mobile ad-hoc networks. In: Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing. (2002) 104-111

8 Noble, B., Satyanarayanan, M., Nguyen, G.T., Katz, R.H.: Trace-based mobile network emulation. In: Proceedings of SIGCOMM 97. (1997) 51-61

9. Maltz, D., Broch, J., Johnson, D.: Lessons from a full-scale multihop wireless ad hoc network testbed. IEEE Personal Communications Magazine 8 (2001) 8-15

10. Ramanathan, R., Hain, R.: An ad hoc wireless testbed for scalable, adaptive qos support. In: Proceedings of IEEE Wireless Communications and Networking Conference (WCNC 2000). Volume 3. (2000) 998-1002

11. Agrawal, P., Asthana, A., Cravatts, M., Hyden, E., Krzyzanowski, P., Mishra, P., Narendran, B., Srivastava, M., Trotter, J.: A testbed for mobile networked computing. In: Proceedings of 1995 IEEE International Conference on Communications (ICC `95). (1995) 410-416

12. Lundgren, H., Lundberg, D., Nielsen, J., Nordström, E., Tschudin, C.: A large-scale testbed for reproducible ad hoc protocol evaluations. In: 3rd annual IEEE Wireless Communications and Networking Conference (WCNC 2002). (2002)

13. de Waal, C.: Bonnmotion: A mobility scenario generation and analysis tool (2002-2003) Available from http://web.informatik.uni-bonn.de/IV/Mitarbeiter/dewaal/BonnMotion/

14. Eclipse Project - Universal Tool Platform: SWT: Standard widget toolkit (2003) Available from http://www.eclipse.org/platform/index.html

15. Segal, M., Akeley, K.: The OpenGL graphics system: A specification (version 1.4) (2002) Available from http://www.opengl.org/developers/documentation/version1_4/glspec14.pdf

16. Kaplan, E.D.: Understanding GPS-Principles and Applications. Artech House Publisher, Boston (1996)

17. International Standard ISO/IEC 8802-11: Information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications (1999)

18. Hamdi, M., Capkun, S., Hubaux, J.P.: GPS-free Positioning in Mobile Ad-Hoc Networks. In: Proceedings of HICSS, Hawaii (2001)

19. Bulusu, N., Heidemann, J., Estrin, D.: GPS-less Low Cost Outdoor Localization For Very Small Devices. IEEE Personal Communications Magazine 7 (2000) 28-34

20. Görgen, D., Frey, H., Lehnert, J., Sturm, P.: Marketplaces as communication patterns in mobile ad-hoc networks. In: Kommunikation in Verteilten Systemen (KiVS). (2003)