On the Accuracy of MANET Simulators *

David Cavin
david.cavin@epfl.ch

Yoav Sasson
yoav.sasson@epfl.ch

André Schiper
andre.schiper@epfl.ch

Distributed Systems Laboratory
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne

ABSTRACT

The deployment of wireless applications or protocols in the context of Mobile Ad-hoc NETworks (MANETs), often requires to step through a simulation phase. For the results of the simulation to be meaningful, it is important that the model on which is based the simulator matches as closely as possible the reality. In this paper we present the simulation results of a straightforward algorithm using several popular simulators (OPNET Modeler, NS-2, GloMoSim). The results tend to show that significant divergences exist between the simulators. This can be explained partly by the mismatching of the modelisation of each simulator and also by the different levels of detail provided to implement and configure the simulated scenarios.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed networks, Wireless communication*; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*; D.4.8 [**Operating Systems**]: Performance—*Simulation, Measurements, Modeling and prediction*; I.6.6 [**Computing Methodologies**]: Simulation Output Analysis

General Terms

Measurement, Performance

Keywords

Mobile Ad-hoc networks, MANET, flooding, simulations, simulators, accuracy, OPNET, NS-2, GloMoSim

*The work presented in this paper was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant 5005-67322.

POMC'02, October 30-31, 2002, Toulouse, France.

1. INTRODUCTION

Context

During the last ten years, Mobile Ad-hoc NETworks (or MANET) have become more and more popular. This still growing interest requires to adapt solutions from the traditional wired networks to the wireless environment (broadcast, routing, etc.). The deployment and the debugging of wireless applications on a real network can be rather tiresome if large networks are considered (typically hundreds of nodes). This is why simulation is an important tool in the sense that it can often help to improve or validate protocols. All simulators provide a complete toolkit to the developers that enables metrics collection and various wireless network diagnostics.

Accuracy of simulations results

There exists several popular simulators, such as OPNET [10] Modeler, NS-2 [9] or GloMoSim [2]. They all provide advanced simulation environments to test and debug any kind of networking protocols, including wireless applications. But, for the simulations to be helpful, it is necessary that the simulated behaviors match as closely as possible the reality. This latter requirement implies to address several issues. Firstly, the application is likely to rely on existing components, such as collision detection module, radio propagation or MAC protocols. The correct modelisation of these components in the simulator is crucial. Each algorithm that is being evaluated is modeled in detail, but the interaction with the other layers is often not taken into account. Secondly, the simulation parameters and its environment (mobility schemes, power ranges, connectivity) must be realistic. Incorrect initial conditions, for example may lead to unexpected results not exploitable in a real network.

This paper presents a set of measures collected during the simulation of the flooding algorithm on different simulators (OPNET, NS-2, GloMoSim). This very simple broadcast algorithm (which simply consists in forwarding to the neighboring nodes every message received for the first time) is a basic building block for several wireless network protocols (such as routing or groups membership protocols, for example). We have carefully implemented it in each simulator paying special attention in setting the same parameters and considering the same scenarios. But, surprisingly, we have collected very different results, barely comparable.

The rest of the paper is structured as follows. The next Section presents the flooding algorithm and the simulated application. Section 4 discusses briefly the architecture and

the deployment procedure of each simulators. Then Section 5 describes in detail the different scenarios and presents the results. Section 6 concludes this paper.

2. RELATED WORK

The literature in the context MANETs provides a lot of papers discussing the efficiency of wireless algorithms and comparing their relative performances using simulations. But very few of them really focus on the possible divergences that can be observed between the simulators, probably because the developers are usually familiar with a single simulator and also do not expect to notice any significant differences. The modelisation of the physical layer in NS-2 and OPNET have already been presented in [12] as well as the important parameters that influence its behavior. It has been shown that the configuration phase affects seriously the absolute performance of a protocol and can even change the relative ranking among protocols for the same scenario.

Related to our work, the effect of detail in MANET simulations has also been studied in [5]. Wireless simulations raise many new questions about appropriate levels of detail in simulation models for radio propagation an energy consumption. Too detailed simulations may not be easily adapted to quickly explore alternatives. On the other hand, simulations which lack of necessary details can result in misleading or incorrect answers. This problem becomes more serious if the same algorithm is implemented in several simulators without the same level of detail.

3. FLOODING ALGORITHM

Introduction

The broadcast of messages is a frequently used operation to spread information to the whole network. It is the simplest building block used by network algorithms and is often required by higher level protocols such as most routing algorithms. For this reason, it is important for the broadcast to be implemented in the most efficient way. Its performance is likely to affect the global efficiency of any protocol using it.

For the simulations we considered a peer-to-peer wireless network of several tens of mobile nodes (typically 50) randomly placed on a 1km x 1km area. The wireless network operates in the ad-hoc mode, without any central infrastructure (such as an access point). Every nodes (or peer) has the same possibilities and functionalities. The description of this environment will be described in detail later in Section 5.

Flooding

A rather direct and simple way to implement broadcast is to flood the message over the network. When a node initiates a broadcast, it transmits the message to its neighborhood. By neighborhood, we mean all the nodes within the sender's transmission range. Then, when the message is received for the first time, the recipient re-broadcasts it. An example is shown in Figure 1 with a network composed of five mobile nodes labeled from A to E.

Node A initiates a broadcast by flooding a message `m` to its surrounding nodes. In step (a), A floods `m` to its single neighbor B. Then, in step (b), node B, which has received `m` for the first time, re-broadcasts `m` to nodes C and D and

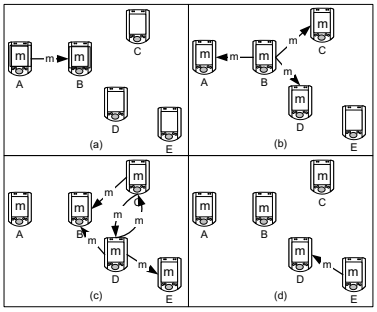

Figure 1: Flooding example

so on. In (c), `m` is completely flooded through the whole network and delivered to every node. In this example, at least three steps are required in order to reach node E on the right. The last step (d) is useless as every neighbor of E has already received `m`.

This technique has an important drawback : it leads obviously to an overhead of flooded messages in the network. With ideal conditions (i.e. all node receive the broadcast) in a network of `N` nodes, a single broadcast will generate exactly `N` copies of itself which are likely to increase the probability of collisions. Moreover, most nodes will receive the same message several times keeping the shared medium unnecessarily busy.

Architecture

The flooding algorithm is build on top of the MAC layer protocol as shown in Figure 2.

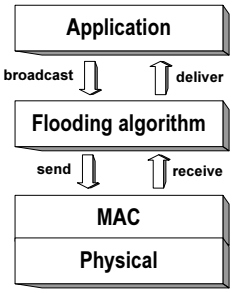

Figure 2: Algorithm protocol stack

The application layer, not described here, accesses the `Flooding algorithm` (`FA`) by the `broadcast` and `deliver` primitives. The `broadcast` primitive is called each time the application wishes to initiate a new broadcast. The `deliver` primitive is called by the `FA` whenever a new broadcast message is received. The `send` primitive floods a message by locally broadcasting it to the node's neighborhood. When a new message is sensed by the MAC layer, it is forwarded to the `FA` by the mean of the `receive` function. All messages exchanged during the simulations are sent to a broadcast MAC address. This means that each node that senses an incoming message will always deliver it to the `FA` layer. Finally, no transport protocol such as UDP or TCP is used and the `FA` is directly connected to the MAC layer. This minimal protocol stack ensures that the comparison between simulators will only depend of the differences between the MAC and physical layers modelisations.

Flooding Algorithm

The flooding algorithm is pretty simple. When a node initiates the broadcast of a message `m`, it sends `m` to all its neighbors. Whenever a node receives `m` for the first time, it rebroadcasts it by relaying `m` to its own neighbors. The code executed by a node n_i is given in Algorithm 1.

Algorithm 1 : Flooding algorithm

```
 1: UPON BROADCAST(M) BY A NODE N_i :
 2: send(m) to all neighbors;
 3: deliver(m);
 4:
 5: UPON RECEIVE(M) BY NODE_i :
 6: if m is received for the first time then
 7:    deliver(m);
 8:    wait for a random time;
 9:    send(m) to all neighbors;
10: else
11:    discard(m);
12: end if
```

The `send(m)` operation (lines 2 and 8) corresponds to a single transmission of `m` to all listening nodes within the transmission range of the sender. As a node cannot sense the message it has just sent, the messages must be locally delivered each time a node initiates a new broadcast (line 3). When a new message `m` is received, the recipient tests if `m` has already been received (line 5). To achieve this, we assume that every message has a unique ID and that each node maintains a list of message IDs. A incoming message `m` is only delivered and its ID inserted in the list if `m` is received for the first time (i.e. its ID is not present in the list). Then `m` is relayed to the surrounding nodes (line 8) after a random delay (line 7). This enables to shift the retransmission times and prevent collisions of messages relayed at the same time by several neighbors. Finally, if the message has already been delivered, it is simply discarded (line10).

4. THE SIMULATORS

Introduction

This section gives a brief overview of each simulator. Its aim is to summarize the different implementation approaches of each simulators. The way a new algorithm is integrated can be pretty different from one simulator to another.

OPNET Modeler

OPNET Modeler is a powerful network simulator developed by OPNET. It can simulate all kinds of wired networks, and a 802.11 compliant MAC layer implementation is also provided. Although OPNET is rather intended for companies to diagnose or reorganize their network, it is possible to implement one's own algorithm by reusing a lot of existing components. Most part of the deployment is made through a hierarchical graphic user interface.

Basically, the deployment process goes through the following phases. First you have to choose and configure the node models (i.e. types) you want to use in the simulations - for example a wireless node, a workstation, a firewall, a router, a web server, etc. Then you build and organize your network by connecting the different entities. The last step consists in selecting the statistics you want to collect during the simulations.

In our case, we had to create a new node model which encapsulates the 802.11 MAC layer of OPNET and, on top of it, an application process that implements the flooding algorithm. A process model (such as the flooding algorithm) is described as a state machine. Each state can have some code that is executed when it gets active. A transition that links two states is followed whenever a certain condition carried by the transition is true.

The difficulty with OPNET Modeler is to build this state machine for each level of the protocol stack. It can be be pretty difficult to abstract such a state machine starting from a pseudo-coded algorithm. But anyway, state machines are the most practical input for discrete simulators. In summary, it is possible to reuse a lot of existing components (MAC layer, transceivers, links, etc.) improving the deployment process. But on the other hand, any new feature must be described as a finite state machine which can be difficult to debug, extend and validate.

NS-2

NS-2 [9] is a discrete event network simulator that has begun in 1989 as a variant of the REAL network simulator [1]. Initially intended for wired networks, the Monarch Group at CMU have extended NS-2 to support wireless networking such as MANET and wireless LANs as well [11]. Most MANET routing protocols are available for NS-2, as well as a 802.11 MAC layer implementation [4].

NS-2's code source is split between C++ for its core engine and OTcl, an object oriented version of TCL for configuration and simulation scripts. The combination of the two languages offers an interesting compromise between performance and ease of use.

Implementation and simulation under NS-2 consists of 4 steps: (1) implementing the protocol by adding a combination of C++ and OTcl code to NS-2's source base; (2) describing the simulation in an OTcl script; (3) running the simulation and (4) analyzing the generated trace files.

Implementing a new protocol in NS-2 typically requires adding C++ code for the protocol's functionality, as well as updating key NS-2 OTcl configuration files in order for NS-2 to recognize the new protocol and its default parameters. The C++ code also describes which parameters and methods are to be made available for OTcl scripting. The NS-2 architecture follows closely the OSI model. We have adapted the implementation of flooding provided in NS-2 in the context of diffusion in sensor networks [6]. An agent in NS-2 terminology represents an endpoint where network packets are constructed, processed or consumed. Such an *Agent* was implemented at the Application layer for the broadcast source, and the simulation trace was collected at the MAC layer.

Some disadvantages of NS-2 stem from its open source nature. First, documentation is often limited and out of date with the current release of the simulator. Fortunately most problems may be solved by consulting the highly dynamic newsgroups and browsing the source code. Then code consistency is lacking at times in the code base and across releases. Finally, there is a lack of tools to describe simulation scenarios and analyze or visualize simulation trace files. These tools are often written with scripting languages. The lack of generalized analysis tools may lead to different people measuring different values for the same metric names.

The learning curve for NS-2 is steep and debugging is difficult due to the dual C++/OTcl nature of the simulator. A more troublesome limitation of NS-2 is its large memory footprint and its lack of scalability as soon as simulations of a few hundred to a few thousand of nodes are undertaken.

GloMoSim

GloMoSim is a scalable simulation environment for wireless and wired networks systems developed initially at UCLA Computing Laboratory [2]. It is designed using the parallel discrete-event simulation capability provided by a C-based parallel simulation language, Parsec [8]. GloMoSim currently supports protocols for purely wireless networks. It is build using a layered approach. Standard APIs are used between the different layers. This allow the rapid integration of models developed at different layers by users.

To specify the network characteristics, the user has to define specific scenarios in text configuration files : app.conf and Config.in. The first contains the description of the traffic to generate (application type, bit rate, etc.) and the second contains the description of the remainder parameters. The statistics collected can be either textual or graphical. In addition, GloMoSim provides various applications (CBR, ftp, telnet), transport protocols (tcp, udp), routing protocols (AODV, flooding) and mobility schemes (random waypoint, random drunken).

With GloMoSim, the difficulty was to describe a simple application that bypasses most OSI layers. The bypass of the protocol stack is not obvious to achieve as most applications usually lie on top of it. Compared to OPNET, for example, the architecture is much less flexible.

5. SIMULATIONS

Introduction

Several scenarios have been implemented on each simulator (OPNET, GloMoSim and NS-2). For the simulations we considered a wireless ad-hoc network of 50 mobile nodes, uniformly placed on a terrain of 1 km x 1 km. All nodes runs the IEEE 802.11 [4] MAC protocol in the ad-hoc mode (or peer-to-peer). In opposition to the infrastructure mode, the ad-hoc mode does not require any central base station (or access point). The network is completely peer-to-peer. Moreover, since we are in a broadcast communication scheme, the MAC protocol runs without *request-to-send* (RTS) and *clear-to-send* (CTS) [1]. This technique solves the well known hidden terminal problem, but does only apply to point-to-point communications and has no sense for a broadcast environment. The physical layer has a data rate of 2 Mbps. Each simulation run lasts 300 seconds during which 10 nodes initiate 100 broadcasts that will be flooded throughout the whole network.

Mobility

Nodes' mobility is an important metric when evaluating ad-hoc networks and can be modeled in several ways. A lot of papers have compared and discussed the relevance of these models [3]. Our goal was to find a mobility model which allows to vary the mobility in a controlled way. We have chosen the *random waypoint model* [7], probably the most popular. A node chooses a destination within the physical terrain. Then it moves in the direction of the destination at a constant speed uniformly chosen between `MIN_SPEED` and `MAX_SPEED`. After it reaches its destination, the node stays there for `PAUSE_TIME` time period. This model offers more control that other ones since two parameters can influence the mobility, the speed and the pause time.

[1]RTS is a control message sent by a node that wants to reserve the shared medium for a certain amount of time in order to reduce the risk of collisions. The destination node acknowledges a RTS by sending to the source node a CTS control message.

Metrics and parameters

Metrics

In order to evaluate the performance of the flooding algorithm, we have defined three important metrics. The first one gives information about the time needed to flood a message. The second measures the general efficiency of the algorithm. And the last one stores the overhead of messages (unnecessarily) flooded in the network.

Time delay: For a node `n`, this is the average time needed for one packet to reach `n`.

Success rate: For a node `n`, this is the difference between the expected and the actual number of messages received at `n`.

Overhead: For a node `n`, this is the sum of duplicated packets received by `n`.

The three metrics above are all locally computed and then averaged among all the nodes. The conjunction of these metrics gives a pretty good idea of the performance of the flooding algorithm.

Parameters

Table 1 summarizes all the common constant parameters of the simulations. In addition to these static parameters, the metric has to be measured against some varying parameters that describe the behavior of an ad-hoc network and that can be set in a controlled way. From a global point of view, we chose to vary the mobility parameters and network load. Table 2 below summarizes these parameters.

Parameters	Value
Terrain size	1km x 1km
Number of nodes	50
Node placement	uniform
Nb. of broadcasting nodes	10
Nb. of broadcasts per node	100
MAC protocol	802.11 without RTS/CTS
Bit rate	2 Mbps
Wireless propagation model	FreeSpace
Antenna Type	Omnidirectional
Mobility model	Random Waypoint
Minimum node speed	0 m/s

Table 1: Static parameters

Results

This part presents the results collected during the simulation of the flooding algorithm with the different simulators. Only

Parameters	Value range	Unit
Flooding rate	[4..20]	packet/seconds
Max node speed	[0..20]	meter/seconds
Pause time	[0..100]	seconds
Power range	[1..300]	meter

Table 2: Varying parameters

the most striking graphs are shown in this paper. We defined several scenarios by varying one or more parameters from Table 2. For each scenario, the set of parameters is given in the table just above the graph.

The first scenario (Figure 3) studies a critical factor that influences the success rate in MANET : the effective transmission range. We notice important differences between the simulators not only in term of absolute value but also in terms of general trend.

Max node speed = 10 m/s, Pause time = 100 s,
Broadcast rate = 4 pk/s

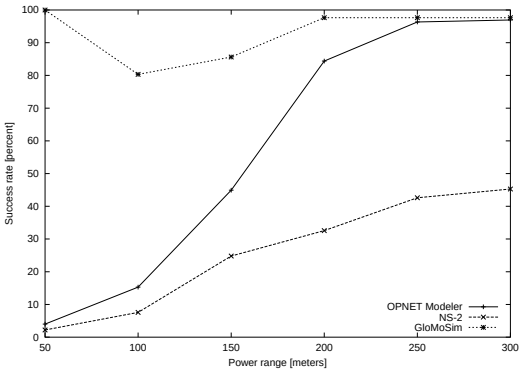

Figure 3: Success rate vs Power range

The next scenario (Figure 4) evaluates the effects of node mobility on the flooding's ability to deliver packets reliably. Here again, we notice a significant difference in terms of success rate. The most important is the maximal success rate that can be achieved by NS-2 for our scenarios. It is less than twice smaller compared to the results of OPNET and GloMoSim.

Power range = 200 m, Pause time = 100 s,
Broadcast rate = 4 pk/s

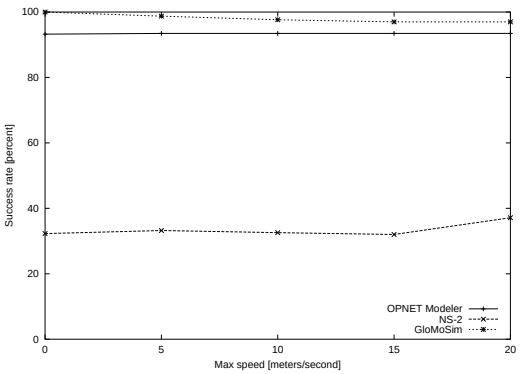

Figure 4: Success rate vs Mobility

The third scenario (Figure 5) presents the average overhead of messages flooded in the network for a single simulation run. This metric is related to the mean number of reachable neighbors (i.e. within transmission range). In this scenario, OPNET receives about three times more duplicate packets.

Power range = 200 m, Pause time = 100 s,
Broadcast rate = 4 pk/s

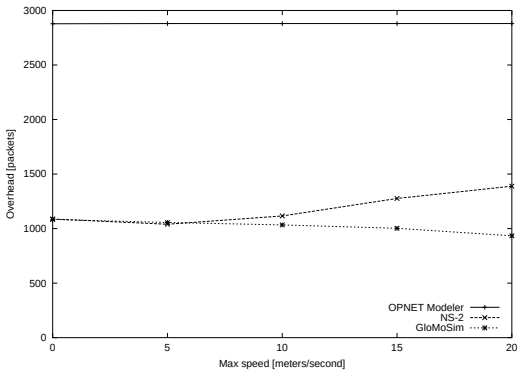

Figure 5: Overhead vs Mobility

The last scenario (Figure 6) compares the average time delay needed to flood a message throughout the whole network. This metric increases with the number of hops from the source to the destination and also whenever collisions occur. OPNET simulates a very high time delay compared to NS-2 and GloMoSim.

Power range = 200 m, Pause time = 100 s,
Broadcast rate = 4 pk/s

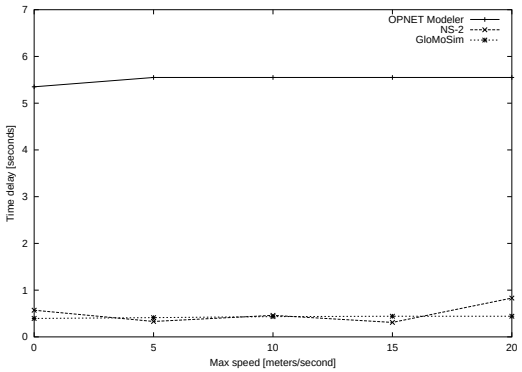

Figure 6: Time delay vs Mobility

Analysis and interpretation

The simulation results of the flooding algorithm shown in the above section demonstrate that the modelisations of the MAC protocol and of the physical layer can lead to pretty different results depending on the simulator. People of the MANET community have rarely noticed these serious divergences because it is often too time costly to become familiar with more than one simulator as long as it is not necessary.

Several reasons may help to explain these unexpected results. One major issue, from the physical layer point of view,

is that the reality (i.e. the environment and mobility model) is hard to describe and the simulators usually provide too simple and too general wireless propagation models. But even if it was possible to extract all environmental parameters from the reality, it would be anyway almost impossible to integrate them efficiently. Thus the physical layer implementation may significantly differ from one simulator to another simply because the levels of detail are not the same. Each implementation relies on several abstractions and simplifications that may lead to different results.

Then, the implementation of a new protocol itself is not straightforward and hard to transpose from one simulator to another. The deployment procedures can be pretty different (state machines, C/C++ code, layers description, graphical interface, script based, etc.) with more or less flexibility and integration facilities of existing building blocks.

Finally, through their successive releases each simulator announces the correction of newly discovered bugs especially in the lower layers such as the MAC and physical layers. It is reasonable to think that MANET simulators still contain errors or incompatibilities with respect to the IEEE 802.11 standard. These remarks tend to show that simulations, as such, do not sufficiently help to deploy and diagnose real networks (i.e. not simulated) as their results are not directly exploitable.

6. CONCLUSIONS

In this paper we have presented the results of the simulation of the flooding algorithm using three popular mobile ad-hoc network simulators. We have intentionally chosen one of the simplest algorithms in order to rely on a minimal set of building blocks (essentially MAC and physical layers). Moreover, flooding is a wide used technique used to spread informations in a whole network by wireless applications and protocols (e.g. routing).

Through intensive simulations we have noticed important divergences between the simulators and, in some cases, results that are barely comparable. The differences are not only quantitative (not the same absolute value) but also qualitative (not the same general behavior). This observation makes the simulation phase less credible as it is difficult to tell which simulator describes better the reality. Simulations do not really improve the development process.

To conclude, we think, according to our experience, that standalone simulations do not really fit the actual needs of wireless applications developers. Instead of simulations, it is probably more realistic to consider a hybrid approach in which only the lowest layers (MAC and physical) and the mobility model are simulated and all the upper layers (from transport to application) are executed on a dedicated hosts (e.g. cluster of machines) [13].

Finally, besides simulations and according to the feeling of the MANET community, there is an important lack of real experiments that prove the feasibility of wireless protocols.

7. REFERENCES

[1] The REAL network simulator. http://www.cs.cornell.edu/skeshav/real/overview.html

[2] L. Bajaj, M. Takai, R. Ahuja, K. Tang, R. Bagrodia, and M. Gerla. Glomosim: A scalable network simulation environment. Technical Report 990027, UCLA Computer Science Departmen, May 1999.

[3] T. Camp, J. Boleng, and V. Davies. A survey of mobility models for ad hoc network research. Dept. of Math and Computer Sciences, Colorado School of Mines, Golden, CO.

[4] I. W. Group. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE specification (http://standards.ieee.org/getieee802/download/802.11b-1999.pdf), Sep 1999. Work in Progress.

[5] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat and K. Lan and Y. Xu and W. Ye and D.Estrin, and R. Govindan. Effects of detail in wireless network simulation. In *Proceedings of the SCS Multiconference on Distributed Simulation*, pages 3–11, Phoenix, Arizona, USA, January 2001. USC/Information Sciences Institute, Society for Computer Simulation.

[6] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks. In *Proceedings of the sixth annual international conference on Mobile computing and networking*, pages 56–67, Boston, MA USA, 2000.

[7] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*, chapter 5, pages 195–206. Kluwer Academic Publishers, Seattle, WA, 1996.

[8] R. A. Meyer. *PARSEC User Manual*. UCLA Parralel Computing Laboratory, http://pcl.cs.ucla.edu.

[9] *The network simulator - NS-2*. http://www.isi.edu/nsnam/ns.

[10] *OPNET Modeler*. http://www.opnet.com/products/modeler/home.html.

[11] T. C. M. Project. The CMU Monarch Project's wireless and mobility extensions to ns, Aug 1998. Available from http://www.monarch.cs.cmu.edu.

[12] M. Takai, J. Martin, and R. Bagrodia. Effects of wireless physical layer modeling in mobile ad hoc networks. In *MobiHoc 2001*, 2001.

[13] Y. Zhang and W. Li. An integrated environment for testing mobile ad-hoc networks. In *The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2002.