YAES - a Modular Simulator for Mobile Networks

Ladislau Bölöni and Damla Turgut
Networking and Mobile Computing Laboratory
Electrical and Computer Engineering Department
University of Central Florida
{lboloni,turgut}@cpe.ucf.edu

ABSTRACT

Developing network protocols for mobile wireless systems is a complex task, and most of the existing simulator frameworks are not well suited for experimental development. The YAES simulation framework was specifically developed such that it allows the fast prototyping of networking protocols, and support real-time experimentation and refactoring. By providing a large set of abstractions and generic implementations, a number of frequently used techniques such as genetic algorithms or neural networks can be created in matter of minutes. Our experience shows that by requiring only Java programming skills which computer science and engineering students commonly possess, YAES can be a useful tool for classroom use, as well.

This paper presents the considerations behind the YAES architecture and provides a description of the system. As a case study, we present the steps necessary for running experiments on the energy efficiency behavior of the Weighted Clustering Algorithm (WCA).

Categories and Subject Descriptors: I.6.3 [Simulation and Modelling]: Applications

General Terms: Design, Measurement

Keywords: Simulation, Mobile networks

1. INTRODUCTION

Developing networking protocols for mobile wireless networks (at any level from physical to application) is a highly complex task. Formal analysis can only cover certain aspects of protocols. Realistic testbeds are costly, create problems of interference, and cannot be used in the early stages of development. Thus, simulation is the key component in the workflow of developing new protocols.

The process of developing new networking protocols is illustrated in the workflow presented in Figure 1. On the diagram, the dotted arrow denotes the relationship between the simulation code and the production code. In an ideal world, the code developed for the simulation would serve as the future production code. However, the differences between the virtual machine of the simulator and the operating system and hardware constraints of the target machines make this frequently impossible.

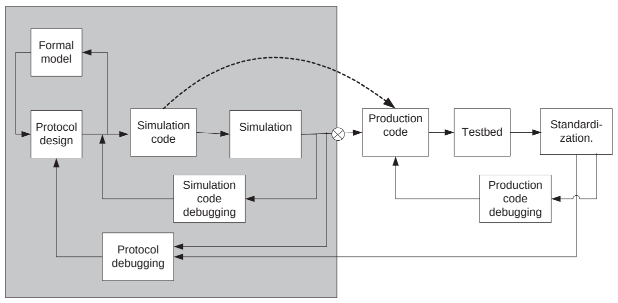

Figure 1: The development process of a wireless networking protocol

It is important to note that protocol development is not linear. There are several points in the development process where most likely several iterations are needed. Feedback from the formal analysis might change the protocol design before simulation. The simulation process requires several iterations to debug the simulation code and verify that it corresponds to the protocol specifications. Once the validity of the simulation is trusted, these results can be used to provide feedback into the protocol design. Once the simulation results are satisfactory, testbed implementations are created, with their own debugging cycle. The final goal is, always, to have a high quality protocol that is suitable for standardization and widespread use. In general, the larger a feedback loop, the more expensive it is to correct the flaws found at a certain location. Based on this observation, formal modeling appears to be the easiest way to improve the quality of the protocol design. However, formal modeling is frequently very difficult or possible only for some isolated aspects of the protocol.

From Figure 1, we can safely say that in the academic practice, most of the work happens in the shaded box, until the point marked with $\otimes$. Proceeding further from $\otimes$ involves expenses frequently out of reach for an academic laboratory, and they are also subject to market forces and regulatory decisions. The widespread availability of software radio architectures (such as PraWN [17]) will allow many researchers to implement real world testbeds, but the importance of simulation will remain still very high.

The nature of academic (both research and instructional) work puts additional requirements for the simulators. The simulator frameworks should allow to quickly try out new ideas, implement and debug the simulation code, analyse the results and feed the observations back to the protocol design. Frequently, academic research involves experimentation with approaches which are not currently technically feasible: for instance one can design routing protocols requiring complex decision making before the packets can be forwarded. For classroom use, a simulator with a fast learning curve and conforming to the necessary pre-requisites is desireable.

There are a variety of high quality simulation tools available both in the commercial and academic sides, such as OpNET [16], ns-2 [15]. In particular, ns-2 was developed in an academic setting and it is overall a high quality, freely available software, with comprehensive libraries. Both researchers and especially educators are encountering problems using it, due to a number of circumstances.

- Using ns-2 involves two languages (Tcl with object-oriented extensions and C++). In general, one needs to know Tcl to set up experiments, while they need C++ in order to develop new protocols. The interfacing between these two languages creates additional problems.
- The fact that the main simulation is run from Tcl, which is loading C++ modules, makes it very difficult for debugging. There is no easy way to run ns-2 in a C++ debugger or integrated development environment.
- The output of the ns-2 simulation are log files, which need to be parsed and post-processed in order to extract and interpret the results. This can be done in tools such as MATLAB, but this introduces the requirement to know yet another programming language.
- There is no easy way to ignore some of the components of the networking stack in order to simplify the simulation in the early stages of the development. For instance, the bit-by-bit packet structure is irrelevant for the performance of many protocols. For many application level protocols, the underlying MAC and routing protocols are not relevant and these application protocols simply rely on the existing MAC and routing protocols. All these, nevertheless needs to be specified, resulting in additional complexity.

The YAES (Yet Another Extensible Simulator) was designed such that it serves the needs of researchers and students in the early parts of protocol development.

The main design principles used are:

- Ability to specify the complete simulator flow from experiment setup to post-processing and evaluation in a single programming language. The language of choice was Java. Experience shows that the compilation of Java code in IDE's is virtually instantaneous, eliminating the need of a special scripting language.
- The YAES simulator is intended to be run inside the Eclipse IDE environment. This allows for faster development turn-around time, ability to perform run-time debugging, as well as more informative error messages. The features of the Eclipse environment allow one step refactoring for the complete simulation flow - in simulators which employ a mix of languages, this is obviously not possible.
- Significant effort was made to detect as many programming errors as possible during compilation time. The YAES core framework features defensive programming code, with extended exception handling and fully typed generic classes. Similar techniques are recommended for user applications.
- Ability to interface with external implementations of abstractions. Current experimental algorithms research frequently uses a variety of abstraction for search, learning, and planning such as neural networks, genetic algorithms, A* search, and so on. It is desirable to have abstract, generic implementations for these either in the framework or in well-integrated external libraries.

Being a new project, YAES currently lacks the library of time tested, widely used simulation code which more established frameworks provide. Nevertheless, YAES already contains a relatively extensive set of library features.

One of the drawbacks of the choice of programming language is that Java is somewhat slower than C++. However, we have found that this difference is insignificant for the types of applications considered. A more valid objection to the use of Java code is that in this way the simulation code cannot be used as production code.

The remainder of this paper is organized as follows. Related work is given in Section 2. The architecture of the YAES simulator is discussed in Section 3. Section 4 presents a case study of the use of the YAES simulator in the study of the energy efficiency of the WCA clustering algorithm. We conclude in Section 5.

2. RELATED WORK

The efforts for the simulation wireless networks are directed in various directions. There are several mature simulation frameworks. The open source Network Simulator ns-2 [15] is a high quality software, and it has a tradition to be used to simulate newly proposed protocols. Commercial offerings such as the Opnet series of simulation products [16] and the GloMoSim-derived QualNet [18] by Scalable Network Technologies. These products offer more extensive support and more friendly user interfaces, but as mature commercial products, they are not geared towards supporting early stage experimentation.

As the high accuracy simulations of mobile wireless networks are computationally expensive, a significant research direction is towards the design of parallel simulators such as SWiMNet [6] and GloMoSim [14]. One research direction is to balance the accuracy of the simulation with the computational requirements [11]. While discussing distributed simulations, we also need to note the standardization efforts of the the HLA (High Level Simulation Architecture) as specified in the IEEE 1516 standard. This specification allows the interconnection of multiple, possibly heterogeneous distributed simulation applications (known as federates).

The new generations of network simulators are considering wireless networking as a standard feature, they frequently employ a Java-based architecture and introduce improvements over various aspects of the more mature simulators.

The JIST simulator at Cornell [3] and the SWANS wireless network simulator built on top of it presents a high memory efficiency, allowing to simulate up to a million nodes on a desktop class computer.

The J-Sim [13] project at UIUC is a Java-based open source wireless network simulation. It is built upon the autonomous component architecture (ACA) and the extensible internetworking framework (INET) of J-Sim, and provides an object-oriented definition of (i) target, sensor and sink nodes, (ii) sensor and wireless communication channels, and (iii) physical media such as seismic channels, mobility model and power model (both energy-producing and energy-consuming components).

The SENSE [8] sensor network simulator from Rensselaer is built on top of the COST general purpose event simulator. It employs the component-port model, where the simulation is set up as components which communicate with others only via inports and outports. During the design of the simulator, the creators considered three classes of users: high-level users relying solely on model repositories and network template libraries, network builders who are designing networks but not interested in extensibility, and component designers, who are interested in modifying models and building new ones from scratch.

3. THE ARCHITECTURE OF THE YAES SIMULATOR

A simulation step simulates the evolution of the described world over a specified time interval, and records its findings into simulation output parameters. A simulation output parameter is implemented as a statistical accumulator, which can automatically perform statistical calculations on the recorded values.

Although it is certainly possible to generate exhaustive logs, we found the use of real time "tally" style statistical accumulators variables offer advantages. These variables can be updated at any moment in the simulator through a command like:

```
output.update(SENSOR_ENERGY, value);
```

At the end of the simulation, we have access to values such as its final value, average value, maximum/minimum value, variance, confidence intervals, and so on. The statistical accumulators and related statistical processing is implemented based on the SSJ software package [19].

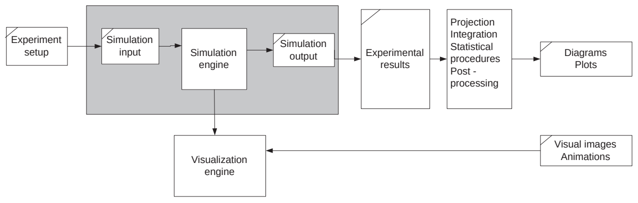

Figure 2: Data flow

In practice, many times we want to run a series of experiments with different parameters. These experiments can be specified in a pre-assembled set of simulation input parameters, which on their own, generate a set of output parameters.

In general, to study the behavior of a new protocol, we ask questions which are answerable by providing cross-sections of the outputs of the experiment sets. Examples of these questions are: how does the average (maximum, minimum,...) queue length varies when the traffic increases? how often the routing table is updated when the mobility of nodes increases?

The post-processing component of YAES allows us to provide a variety of cross-sections on the results of a series of experiments. The typical way of working is by setting up these cross-sections very early in the simulation process. This way, the complete workflow can be performed in the framework; single “run” command performs all the processing from the generation of input data to the generation of graphs and diagrams. (Obviously, the simulation itself can still take a significant amount of time).

The cross-sections are essentially tracing the evolution of one or more parameters, while varying one input parameter. This tracing happens on the previously computed result set. They can be seen as special purpose database queries. It is planned for a future release of YAES to provide the storage of the simulation results in an external SQL database.

One special cross-section operation is the statistical merging of multiple experiments with the same parameters, but with dependency on random sources. This allows us to generate statistical information such as maximum, minimum, confidence intervals accross *experiments*. The statistical samplers are providing this functionality inside a single experiment. The results can be either written in a text file or can generate graphs through the generation of plots in MATLAB.

4. CASE STUDY

In the following, we present the steps followed to develop a simulation scenario in YAES. We will put a special emphasis to the steps where new code needs to be developed, as opposed to the places where YAES library functions can be used.

The scenario chosen is a simulation study involving the influence of the parameters of the WCA clustering algorithm over the energy conservation in a sensor network. While these examples are taken from an actual study, our emphasis here on the architecture of the simulator framework and setup of the experiments.

4.1 The weighted clustering algorithm

The weighted clustering algorithm (WCA) [7] is an on-demand clustering algorithm for multi-hop packet radio networks or in another words, *ad hoc* networks. Since these types of networks are dynamic in nature due to the mobility of the nodes, it is essential to keep the network topology as stable as possible. The aim is to elect the ideal number of clusterheads to cover the entire network without causing any severe degradation in the performance. The algorithm takes into consideration the ideal degree, transmission power, mobility, and battery power of mobile nodes.

Let us briefly summarize the WCA which selects the clusterheads based on the weight ,W_v, of each node v. As detailed in [7], W_v is defined as

$$W_v = w_1\Delta_v + w_2D_v + w_3M_v + w_4P_v$$

where Δ_v is the *degree-difference*, D_v is sum of the distances of the members of the clusterhead, M_v is the average speed of the nodes, and P_v is the accumulative time of a node being a clusterhead. The corresponding *weighing factors* are such that $\sum_{i=1}^{4} w_i = 1$. That node v with the minimum W_v is chosen to be the *clusterhead*. Once a node becomes a clusterhead, neither that node nor its members can participate further in the cluster election algorithm. The algorithm terminates once all the nodes either become a clusterhead or a member of a clusterhead. All the clusterheads are aware of their one-hop neighbors as well as the ordinary (non-clusterhead) nodes know their clusterheads.

The first component, Δ_v helps in efficient MAC functioning since it is desirable for a clusterhead to handle a certain number of nodes in its cluster. The motivation of D_v is to conserve energy consumption. Since more power is required to communicate to a larger distance, it makes sense to use the sum of the squares (or higher exponent) of the distances. The power required to maintain communication link from a node farther from the clusterhead increases much faster than linearly with distance (at least in the farfield region). M_v represents the mobility of the nodes. Intuitively, a node with less mobility is preferred to be a clusterhead such that the network topology will remain stable for longer periods of time. The last component, P_v, is measured as the cumulative time a node acts as a clusterhead. It is assumed that all the nodes start with the same bettery power. In that case, the battery drainage gives a direct measure of the available battery power.

Initial implementation and simulation studies for WCA were done in C++ code which was not only difficult to maintain but also to run comparisons against other existing heuristics such as Lowest_ID [1, 2, 9], Highest Degree [10, 12], node weight [4, 5], and so on. Implementing all the protocols in ns-2 would involve flushing out a number of low-level details, such as packet structure, which is not relevant from the performance of the protocols.

We have rewritten the code in the YAES simulator, concentrating only on the components which are relevant for the performance of the protocol. As a note, the relevant features might not form a continuous segment in the protocol stack. For the study of clustering protocols, we are interested in physical layer characteristics such as the geographical location and transmission range. We are also interested in the flow of messages, as it happens at the network layer. The details of the MAC protocol, however are less relevant. Although collisions, retransmissions, and so on can affect the behavior of the real world implementation, with some cautionary assumptions (e.g., low transmission frequency), they can be ignored for the simulation purposes.

4.2 Developing a simulation in YAES

Developing a simulation in YAES involves the development of the *simulation context* and the *simulation code*.

The simulation context is the description of the simulation world. It needs to be implemented as a class implementing the `IContext` interface. In practice, the context of many simulations can be assembled from pre-existing components from the YAES library. In our case, the environment is composed of a `SensorNetwork` class, which contains a set of `SensorNode` objects. As the clustering algorithms can be added as add-ons to this object, we do not need to subclass the object `SensorNode` object.

The initialization step of the context creates a desired sensor network. In this case, we create a network with a random initial distribution of the nodes which is subject to some properties specified in the simulation input: number of nodes, size of the field, and transmission range of the nodes.

The simulation code is the active part of the simulation describing the evolution of the environment in time, as well as the actions taken by the participants. For our chosen simulation, this includes the mobility model of the nodes, the energy model and, the actions taken by the nodes in the context of the WCA algorithm. While for the mobility and energy models are implementations in the YAES libraries, the code corresponding the WCA algorithm needs to be developed for the purpose of this simulation. We choose a simple random waypoint mobility model for the nodes and an energy efficiency model which considers both the power consumption necessary for processing, as well as the transmission power of the nodes.

As we have discussed earlier, the power consumption directly relates to the accumulated time a node acted as a clusterhead due to additional responsibilities to its member nodes. The distances between the clusterhead and its members also need to be considered since the power consumption increases with the distance. In order to account for the communication cost with the member nodes, the sum of the distances of the members of the clusterhead is also included.

In the implementation of the WCA algorithm, we do not need to discuss the implementation details at the level of MAC protocols. We will simply assume that the MAC protocol will add a constant fraction to the required communication bandwidth.

4.3 Simulation and visualization

Once the context and a visualization code is developed and assembled, we are ready to assemble an experiment and run the visualization. To run a single experiment, we need to set the input variables of the simulation, and then pass the parameters, the simulation code object, and the simulation context object to the simulation parameters.

```
SimulationInput sim = new SimulationInput();
sim.set(WCA_W1, 0.7);
sim.set(WCA_W2, 0.2);
sim.set(WCA_W3, 0.05);
sim.set(WCA_W4, 0.05);
sim.set(WCA_OPTIMAL_DEGREE, 10);
sim.set(WCA_FORCED_RECOMPUTE, WCA_FORCED_RECOMPUTE_NO);
sim.set(NODECOUNT, 20);
sim.set(TRANSMISSION_RANGE, 10.0);
sim.set(MAX_DISPLACEMENT, 5);
sim.set(RECORD_DISTANCES, RECORD_DISTANCES_NO);
sim.set(SIMULATION_CODE_CLUSTERING, SC_CLUSTERING_WCA);
sim.set(SIMULATION_CODE_MOBILITY, SC_MOBILITY_RANDOM);
sim.set(SIMULATION_CODE_ENERGY, SC_ENERGY_4THPOWER);
sim.setStartTime(0);
sim.setStopTime(1000);
SimulationOutput sop =
   Simulation.simulate(sim, Clustering.class,
       ClusteringContext.class);
```

The result of the simulation is a `SimulationOutput` object, which contains all the statistical accumulators set by the simulation code. In our case, the statistical accumulators collect information about the failure of the nodes due to the lack of power. For reasons of convenience, the simulation output encapsulates the simulation input and the final state of the simulation context as well. Thus, this object contains the complete state of the experiment for further processing.

One way for gaining insight into the simulated phenomena is to visualize the simulation. YAES provides the `VisualMap` architecture for this. If we want to visualize an object, we can add it to the visual map, together with a "painter" object (implementing the `IPainter`) interface. There are several painters provided by the library, but the users might prefer extending them or implementing custom painters. The libraries already contain painters for networks and network nodes. For the purpose of this visualization, we added code for marking the current clusterhead with a double circle, and showing the node's transmission range as a shaded area. A snapshot of the resulting visualization image is presented in Figure 3.

The visualization appears as an animation during runtime. YAES can save these snapshots as a series of images, which can be assembled into a video file or animated GIF image with programs such as Adobe Premiere. Several video files representing simulations performed with YAES are available on the YAES website [20].

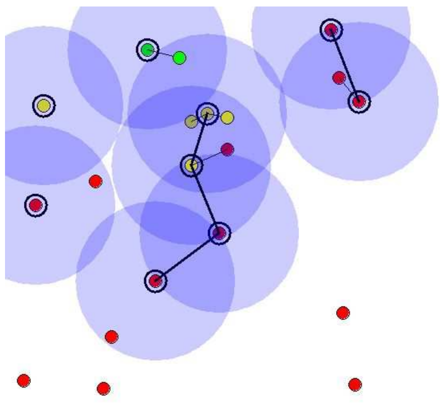

Figure 3: Visualizing the evolution of a sensor network clustered using the WCA algorithm. The larger shaded circles show the transmission range of the network nodes. The node color (green, yellow, red) indicates the power level, with the red nodes being inactive, with no power reserve left. The links show the connections between the clusterheads.

4.4 Experiment sets. Postprocessing and presentation.

The visualization of a single simulation run can allow us to trace the evolution of an algorithm. For many cases however, we are interested in the result of a potentially large number of experiments.

One reason for multiple experiments is when we want to vary one or more parameters of the simulation. After this, we might be interested to extract *projections*, i.e., the evolution of certain output values of the simulation in function of other values (which are typically input variables). Thus, in our case, we might be interested in the evolution of the moment in time when 10% of the nodes failed, in function of the w_1, w_2, w_3 and w_4 parameters of the WCA algorithm.

Another reason for performing multiple experiments is statistical smoothing. In this case, we are repeatedly running a nondeterministic experiment to study the spread and the evolution of certain outputs. Preliminary experiments have shown that the 10% failure time has a considerable random spread in sensor network clustering experiments. Therefore, we decided to perform 20 experiments and integrate the results. The code necessary for this operation is presented below (with the repetitive parameter settings omitted).

```
List<List<SimulationOutputParameters>>
    outputsToIntegrate =
    new ArrayList<List<SimulationOutputParameters>>();
for (int i = 1; i <= 20; i++) {
  // 20 repeated experiments with identical inputs
  List<SimulationInputParameters> inputs =
       new ArrayList<SimulationInputParameters>();
  for (double w4 = 0.0; w4 <= 1.0; w4 = w4 + 0.05) {
      SimulationInput sim = new SimulationInput();
      sim.set(WCA_W4, 0.05);
      // ... set other parameters
      inputs.add(sim);
   }
   // run a set of simulations
   List<SimulationOutputParameters> outputs =
   Simulation.simulationSet(inputs,
   Clustering.class, ClusteringContext.class);
   // integrate the outputs
   outputsToIntegrate.add(outputs);
}
List<SimulationOutputParameters> outputs =
IntegratedSOP.integrateListOfSops(outputsToIntegrate);
```

We can note that YAES can execute a set of (unrelated) simulation experiments through a single command. Current work is being done towards allowing a distributed execution of this command on a Beowulf cluster. The result of a set of experiments are a list of simulation outputs. As we are running 20 identical experiment sets, the result will be a list-of-lists of outputs. This double list is integrated into a statistically smoothed output by the `integrateListOfSops` function.

What remains is to extract from this set of results the data we are interested in and present it in an appropriate format. The YAES simulator provides a way to create projections of the output sets and generates MATLAB scripts which directly plot the required graphs. In this case, having a statistical result, we are interested in plotting the average values as well as the confidence intervals of the results. This can be achieved with the following commands:

```
List<PlotLineDescription> list =
  new ArrayList<PlotLineDescription>();
list.add(new PlotLineDescription(first,
  RandomVariable.TYPE_AVG,ENERGY_FIRST_NODE_FAILED,
  RandomVariable.TYPE_AVG,
  RandomVariable.TYPE_CONFINT_RADIUS, "First failed"));
list.add(new PlotLineDescription(first,
  RandomVariable.TYPE_AVG, ENERGY_PERCENT_NODES_FAILED,
  RandomVariable.TYPE_AVG,
  RandomVariable.TYPE_CONFINT_RADIUS, "10 percent"));
SimulationHelper.generateErrorBarPlotFromPlotDescription
  ListWithTitleLabel("results.m", list, outputs,
  "Failure times", "w4", "time");}
```

The result of running this code is the "results.m" MATLAB file. This can be run directly (no MATLAB programming knowledge is required) and the resulting graph is presented in Figure 4. A visual interpretation of this file is that the increase of the w_4 parameter in the WCA algorithm slightly increases the lifetime of the network. As this parameter controls the influence of the time a node has spent as a clusterhead, this result conforms to our expectations.

5. CONCLUSIONS

This paper presented the YAES simulation architecture, a software targetted towards research and classroom simulation of mobile wireless systems. By targetting simulations which are experimental in nature and involve significant processing outside the networking stack, we argued that YAES covers a niche which is insufficiently addressed in other simulators currently available.

The authors wish to extend thanks to the developers Majid Ali Khan, Linus Luotsinen, Xin Bai and Kresimir Sivoncik. The work reported in this paper was partially supported by National Science Foundation grants MCB9527131 and DBI0296107.

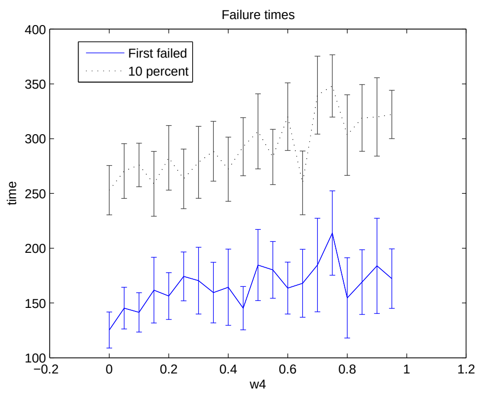

Figure 4: The influence of the w_4 parameter of the WCA clustering algorithm to the time the first failed node and the time to the moment when 10% of the nodes failed. The bars show the confidence intervals.

6. REFERENCES

[1] D. Baker and A. Ephremides. The architectural organization of a mobile radio network via a distributed algorithm. *IEEE Transactions on Communications*, 29(11):1694–1701, 1981.

[2] D. Baker and A. Ephremides. A distributed algorithm for organizing mobile radio telecommunication networks. In *Proceedings of the 2nd International Conference on Distributed Computer Systems*, pages 476–483, April 1981.

[3] R. Barr, Z. J. Haas, and R. van Renesse. Jist: an efficient approach to simulation using virtual machines. *Softw., Pract. Exper.*, 35(6):539–576, 2005.

[4] S. Basagni. Distributed and mobility-adaptive clustering for multimedia support in multi-hop wireless networks. In *Proceedings of Vehicular Technology Conference 1999-Fall*, pages 889–893, 1999.

[5] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of International Symposium on Parallel Architectures, Algorithms and Networks*, June 1999.

[6] A. Boukerche, S. K. Das, and A. Fabbri. Swimnet: a scalable parallel simulation testbed for wireless and mobile networks. *Wirel. Netw.*, 7(5):467–486, 2001.

[7] M. Chatterjee, S. Das, and D. Turgut. Wca: A weighted clustering algorithm for mobile ad hoc networks. *Journal of Cluster Computing (Special Issue on Mobile Ad hoc Networks)*, 5(2):193–204, April 2002.

[8] G. Chen, J. Branch, M. J. Pflug, L. Zhu, and B. Szymanski. SENSE: A sensor network simulator. In *Advances in Pervasive Computing and Networking*, pages 249–267. Springer, 2004.

[9] A. Ephremides, J. Wieselthier, and D. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. *Proceedings of IEEE*, 75(1):56–73, 1987.

[10] M. Gerla and J. Tsai. Multicluster, mobile, multimedia radio network. *Wireless Networks*, 1(3):255–265, 1995.

[11] Z. Ji, J. Zhou, M. Takai, and R. Bagrodia. Scalable simulation of large-scale wireless networks with bounded inaccuracies. In *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 62–69, New York, NY, USA, 2004. ACM Press.

[12] A. Parekh. Selecting routers in ad-hoc wireless networks. In *Proceedings of the SBT/IEEE International Telecommunications Symposium*, August 1994.

[13] A. Sobeih, W.-P. Chen, J. C. Hou, L.-C. Kung, N. Li, H. Lim, H.-Y. Tyan, and H. Zhang. J-Sim: A simulation and emulation environment for wireless sensor networks. *IEEE Wireless Communications Magazine, to appear*, April 2005.

[14] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *PADS '98: Proceedings of the twelfth workshop on Parallel and distributed simulation*, pages 154–161, Washington, DC, USA, 1998. IEEE Computer Society.

[15] ns-2 the network simulator. URL `http://www.isi.edu/nsnam/ns`.

[16] Opnet technologies. URL `http://www.opnet.com`.

[17] PraWN: Programmable for Radio for Wireless Nets. URL `http://prawn.cs.colorado.edu/`.

[18] Qualnet by scalable network technologies. URL `http://www.scalable-networks.com/`.

[19] Stochastic simulation for java. URL `http://www.iro.umontreal.ca/ simardr/ssj/`.

[20] YAES: Yet Another Extensible Simulator. URL `http://netmoc.cpe.ucf.edu/Yaes/Yaes.html`.