SANS: A Simple Ad Hoc Network Simulator

Nicolas Burri, Roger Wattenhofer, Yves Weber, Aaron Zollinger
{burri@tik.ee, wattenhofer@tik.ee, webery@student, zollinger@tik.ee}.ethz.ch
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland

Abstract: If all communication layers are affected by the specific properties of ad hoc networks, such as unreliability of wireless links or node mobility, both the network and transport communication layers deserve special attention in this context. In this paper we present a Simple Ad Hoc Network Simulator SANS particularly intended for the simulation of distributed programs and protocols on these two layers. This focused approach allows for easy and fast learning about the concepts and correct usage of SANS as a development, debugging, testing and presentation tool. Together with the fact that for the implementation of the simulator and for the simulated programs the Java programming language has been chosen, SANS lends itself to both rapid prototyping and educational purposes.

1. Introduction

For the development of distributed network algorithms and protocols the employment of network simulators is mandatory. Not only is testing and debugging of a distributed program more convenient in a simulator as opposed to direct implementation on the target platform, be it for the presence of better development, testing, or debugging tools. Also reproducibility of certain conditions and easy composition of test case suites is one of the key advantages of simulators.

A number of network simulators are available [1, 2, 3], which have in common that they aim at perfect simulation of all network layers. Obviously this incurs high conceptual complexity with respect to proper usage of such simulators. Often understanding a simulator in its whole complexity is virtually impossible, which can sometimes lead to wrong usage and thus erroneous or misleading simulation results. Moreover, sometimes only specific aspects of distributed programs or certain network layers are to be considered, where employment of a full network simulator is like shooting flies with a cannon. Above all when used for rapid prototyping or for educational purposes—such as in exercises accompanying a lecture—the complexity of network simulators can turn out to be a major drawback. The effort and time invested in understanding a full-grown simulator often stands in no reasonable relation to the yielded profit.

Against this background, we present in this paper a Simple Ad Hoc Network Simulator SANS, with the goal of providing an intuitive and easily understandable tool. Ideally, five minutes should suffice to become familiar with SANS as a development, testing and debugging tool.

One of the main reasons for the simplicity of SANS lies in the fact that SANS is particularly intended for the simulation of networking layer and transport layer protocols and algorithms. Both of these layers deserve special attention in the context of ad hoc networks: The specific dynamic behavior of wireless links as opposed to wired connections, particularly high packet loss probability, requires routing and transport layer algorithms specially designed for ad hoc networks.

SANS is suited for the simulation of programs on the networking and transport layers in that it provides an abstraction of the physical and data link layers. In particular a program running in the simulator can use a simple interface to send and receive data packets to and from its direct neighbors in the ad hoc network. Technically, this is solved by means of the UDP multicasting interface. As a side-effect, this entails that a program developed in the simulator can directly be transferred to real-life systems if the UDP multicast interface is available on the target platform.

Since SANS is intended not only for prototyping but also for educational purposes, Java has been chosen both for the implementation of the simulator itself and as the programming language of the simulated programs. This implies that the simulator is platform-independent and can be employed on all systems supporting the Java environment.

The remainder of the paper is organized as follows: Section 2 summarizes the design goals we defined for this work. Section 3 gives a short overview over the SANS system followed by a description of the network simulation

architecture in Section 4. Sections 5 and 6 go into more details explaining the implementation of management and simulation components of the simulator. Section 7 describes limitations and problems of our approach followed by a usage example in Section 8. The subsequent section discusses related work and Section 10 concludes the paper.

2. Design Goals

To reduce the overhead incurred to become familiar with correct usage of simulators, we decided to write a network simulator of our own. Our goal was to design a system which does not require more than a couple of minutes to get used to but still offers all necessary options for testing and presentation of algorithms running on wireless networks. The key features we defined for our system can be summarized as follows:

- Ease of use,
- graphical representation of the network,
- real time network simulation,
- code developed on the simulator should run on “real-world” hardware without adaptation,
- individual transmission ranges for each node,
- adjustable link properties as delay and packet loss, and
- support of a standard programming language.

In the following section we will give an overview on how we realized these design goals.

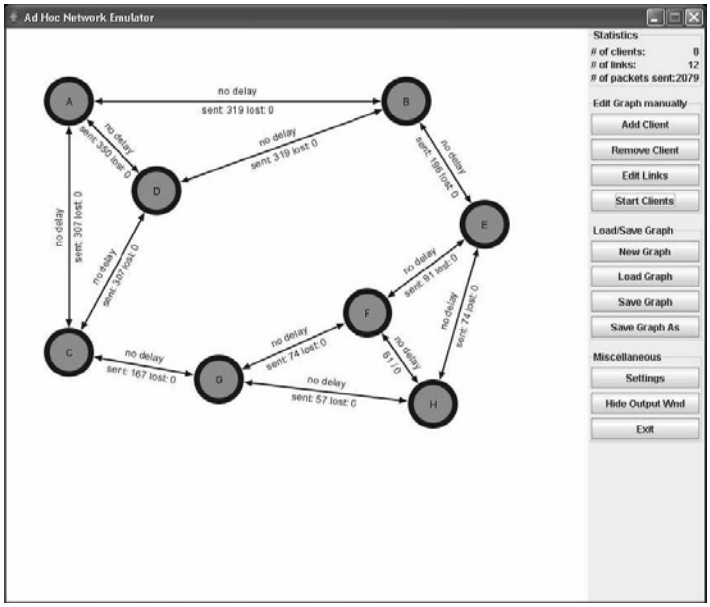

Figure 1: SANS main window showing a network consisting of eight nodes which are partially connected.

3. Overview

SANS is a Java network simulator which is designed to simplify the development and presentation of algorithms at the network and transport layers. It allows the simulation of a whole network of independent nodes which are running individual client applications on one computer without the need of a complex testbed.
As can be seen in (Fig. 1), SANS shows a schematic representation of the simulated network including nodes and possible links in its main window. Network nodes running Java client applications are displayed as circles with a node identifier in the center. Possible links (indicating that two nodes are in communication range) are symbolized as arrows. Whenever a message is sent over such a virtual link, the head of the corresponding arrow flashes for a

moment and the packet count of the link is increased. In many cases this visual feedback on packet transmission is sufficient to observe the behavior of an algorithm without the need for any further output or log file. Algorithms which are based on flooding, for example, can be watched very well since the wave of outgoing messages is clearly visible to the user. A quick look at the message count of all links also makes it possible to see if the algorithm behaves properly or not.

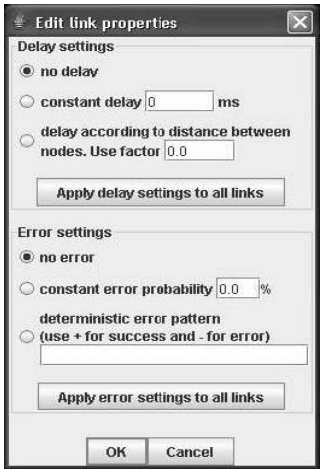

Figure 2: The configuration dialog to set the properties of a link between two nodes.

The aspect in which SANS differs from most other simulation systems is its ease of use. Setting up a network is done through simple drag and drop operations and small dialogs. A right click on the main window creates a new node at this position and a dialog opens where the user is asked to specify the client application which is to be run on this node. Similarly a connection between two nodes is added by dragging a line from one node to the other. The dialog shown in (Fig. 2) is then opened and the user is asked to set the link properties. SANS supports uni- and bidirectional links making it possible to simulate networks consisting of heterogeneous hardware with different transmission ranges. Also delay and packet loss can be set individually for each link to achieve a realistic network simulation.

Another feature which is worth mentioning is the ability of SANS to deal with changes of the network topology at runtime. Links and nodes can be added or removed while the simulation process is running, without the need for a restart. Consequently SANS can also be used to test the reaction of an algorithm to spontaneous network changes which are a common problem in wireless networks.

Finally, SANS offers a comfortable way to manage console output of the simulated nodes. A special output window contains a tabbed pane with an individual tab for each node. The user can therefore check the output of a certain node without having to filter out the output of the remaining active clients.

4. Simulation of the Physical and MAC Layers

Wireless networks differ in many aspects from wired systems. Their higher packet loss rates and more frequently changing network topologies put other requirements to routing and error-handling algorithms than traditional networks. Common routing algorithms have been found to perform poorly on such systems. Therefore the design and analysis of new algorithms optimized for wireless networks is mandatory. A similar effect of wireless links can be observed with respect to transport layer protocols.

A major problem in designing and testing algorithms at the network and transport layers is the lack of simple interfaces to implement them without using complex simulation systems. SANS solves this problem by offering the possibility to build an overlay network on top of a UDP multicast system. Multicasts feature a similar behavior as radio communication since not only the intended destination receives a message but all nodes within the same multicast group. In SANS all nodes join a common multicast group, which leads to a system where each node could

receive every message sent through the system. This behavior maps to a network where all nodes are within each other's transmission range (or a fully connected network graph). Obviously such a model is not well suited to test multi-hop algorithms often used in wireless networks. SANS therefore restricts the message delivery to those nodes which have a direct connection to the sender in the schematic network representation, being within the virtual transmission range of the sender.

The main advantage of building an overlay network on top of a UDP multicast system is the existence of a handy interface in Java to send and receive multicasts (java.net.MulticastSocket). SANS intercepts calls to this class and handles them internally in a way transparent to the client application. Besides the convenience for the developer of using a well-known interface to send and receive messages, this approach also has the advantage that code developed within SANS runs on real hardware without any adaptation. The multicast interface is supported by all major Java Virtual Machines. Therefore the method calls intercepted by SANS when run within the simulation framework are passed through to a networking device when executed on real hardware.

(Fig. 3) shows the process performed when a message is sent in SANS. The communication is initiated by a client application calling the send() method of the Java MulticastSocket to broadcast a packet. This method call is intercepted by SANS in a way transparent to the client and instead of delivering the data to a networking device it is further processed by the simulation system. First, SANS determines which nodes are in transmission range of the sender (which means that there is a link in the schematic network representation between the sender and these nodes). In a second step, the individual link properties of each of these connections are analyzed. According to the user-defined error model, the system decides if the message reaches the receiving node or if it is dropped due to simulated packet loss. Finally, if the message is not dropped, SANS calculates the time of arrival of the packet according to the link delay and puts the data in the delivery wait queue of the receiving node. Eventually the receiving nodes call the receive() method of the MulticastSocket to poll for newly available data. Again this call is intercepted and SANS returns the next valid packet from the corresponding wait queue.

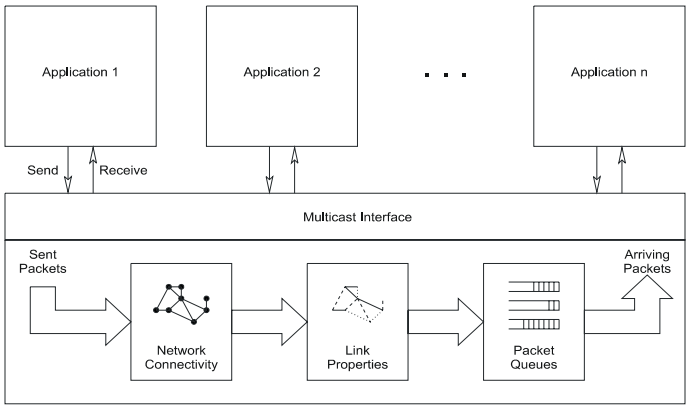

Figure 3: Architecture of the network simulation layer**.**

5. Features and Handling

In this section the most important features of SANS are described in more detail.

5.1 Network Nodes

SANS represents network nodes as red or blue circles, respectively, depending on their current state (activated or inactive). New nodes can be created by right-clicking on a free spot on the pane whereupon a new red circle is placed at this position and a dialog is opened. The user is asked to provide the name of the main class as well as the necessary classpath of the client application which is to be run on this node. Also parameters with which the client program might need to be started can be specified during this initial setup. Initialized nodes stay inactive until the user explicitly starts the simulation process and the activated nodes turn from red to blue.

5.2 Links

Many simulators use the unit disk graph model to define the topology of simulated networks. In this model a link between two nodes exists if their distance is at most one unit. SANS uses another approach, allowing the user to add edges between any two nodes by drawing a line from one to the other. This representation has two main advantages:

Readability: The resulting network graph in our system is a lot easier to read than a unit disk graph version, particularly if a lot of nodes are placed close to each other. Especially for demonstration purposes this is a clear advantage as there is no unessential information shown on the screen making it easier to follow the important details.

Flexibility: The unit disk approach assumes a constant transmission range in all directions. This is often not realistic as obstacles may disturb radio signals in real world environments. Our system is more flexible in that it allows the user to set all possible connections on a node-by-node basis, making it possible to model communication gaps.

The system also allows the user to set three parameters for each simulated link:

Direction: SANS supports unidirectional and bidirectional links between two nodes. This feature allows simulation of networks built on heterogeneous hardware with different transmission ranges.

Delay: While simulated links can transfer packets without a measurable delay, it is often necessary to add some artificial lag to obtain a more realistic simulation. SANS therefore offers two different ways to set the minimum transmission time on a link. The user can set a fixed amount of time a packet will stay on the link before it is delivered to the receiver. Alternatively the simulator can also set a delay according to the distance between the connected nodes, resulting in an intuitive visual representation of the network delays.

Packet Loss: As wireless links tend to lose packets due to interference, it is important that a simulator supports the simulation of packet loss. SANS again offers two different ways to control this parameter. The user may either set a percentage of all transmitted packets to be randomly dropped or an error pattern may be specified (for instance drop the third, the fifth, and the tenth packet). While the first option is well suited for long time simulations with high data volume, the error pattern option can be very helpful debugging a program as the packet flow can be reproduced deterministically.

5.3 Configuration of the Simulation Framework

The configuration of SANS itself is as easy as setting up the simulated network. All options can be reached with one click and parameters can be set through dialogs. Besides various internal parameters (such as buffer sizes), which usually do not have to be changed by the user, SANS also offers some convenience functions:

Saving and Loading a Network: For debugging or presentation purposes it is often required to run the simulation of a certain network repeatedly which can be simplified by the use of previously stored networks. SANS can save any network setup including link and node parameters to a file and load it back from there as required.

Individual Console Output: SANS offers two different ways to handle the console output of client applications. Either the whole output of all clients can be sent to the system console (with an additional source identifier if required) or it can be redirected to a tabbed pane where each client has its own panel. Output to the system console can easily be stored and processed by external applications. The tabbed solution is more user-friendly as the output is clearly separated and therefore easier for the user to read.

Packet Logger: SANS contains an optional, simple packet logger recording the send and receive events of every packet transmitted. Also the contents of the packets are logged and stored as a string of hexadecimal values. The recorded data can either be printed to the system console or into the simulator output window, depending on the user requirements.

6. Internal Network Simulation

As described in Section IV the Java MulticastSocket class is used to send and receive messages in the SANS framework. In this section we discuss the approaches we have evaluated to hook into the Java system and present their individual advantages and disadvantages.

- The first approach envisioned was rewriting the MulticastSocket in a separate class, supporting the same interface as the original java.net.MulticastSocket class. This solution works well from the simulation point of view but the user has to adapt some imports to enforce the new class to be called instead of the standard library MulticastSocket class. As this was contradictory to our specification to require no adaptation of

code running on a real machine we abandoned this approach.

- The second idea was to use a custom Java class loader to solve the problem. If a client requests a new instance of the MulticastSocket, the default class loader receives a notification and creates a new instance of the default MulticastSocket class. By replacing this default class loader with a customized one, the call could be intercepted and an instance of the modified socket could be returned. This procedure works as expected until the point where the class loader needs to register the modified socket to the system. For security reasons, the Java Runtime System does not allow to register customized classes with a name starting with "java" (such as java.net.MulticastSocket). There is no proper way to circumvent this restriction and so we had to drop this approach as well.
- Eventually we found a suitable approach in the DatagramSocket class (which is the direct superclass of MulticastSocket). This class offers the static method setDatagramSocketImplFactory to set a customized factory for creating sockets. Using this method, the customized socket can be returned to the client without any changes, since the DatagramSocketImplFactory can be set by the simulator before any client is started. This allows replacing the MulticastSocket in a manner totally transparent to any simulated client program. As a nice side effect of this solution, SANS not only supports communication through multicasts but any UDP communication is simulated properly.

6.1 Separation of the Clients

SANS runs the simulation framework and all simulated client programs within one common Java Virtual Machine. As a consequence name collisions between the active clients need to be prevented to guarantee the absence of hidden communication channels (for example through static variables). The only secure way to assure such separation is to grant each node an individual namespace. As described before, Java uses class loaders to instantiate objects and to allocate memory for methods and fields of the corresponding classes. SANS loads each client in a new custom class loader which automatically generates a new namespace for the client. As a result the system has a slightly higher memory usage, since some classes are loaded multiple times but it allows the usage of static methods and fields in the client code without any interference between the different simulated applications.

7. Limitations and Problems

The price SANS has to pay for its simplicity of use are some limitations and problems which are difficult to solve. The most important open problems are described in the following section.

7.1 Mobility

The user can add and remove possible links within the simulated network at runtime to simulate the mobility of nodes. SANS does not yet support scripted movement of clients, though. While it is possible to add script support to the simulation system with a reasonable amount of work it is difficult to design an intuitive user interface for this task.

7.2 Stopping Clients

Removing a client from the simulated network is done by removing the client from the network graph. That is, removing all links connected to the corresponding node and deleting the node itself. While a removed node is disconnected from the simulated network, the client application continues running. This leads to a constantly increased waste of CPU time and memory if a lot of nodes are removed in the course of a simulation process.

The only viable solution to stop a removed client would be to identify the associated threads and to stop all of them. Unfortunately the Java method Thread.stop() which could be used to stop threads is deprecated and the Sun Java Virtual Machine does not offer any way to enforce the termination of a thread any more.

7.3 System.exit()

Another problem that emerges from running several client applications in one Java Virtual Machine are calls to System.exit(). Often client applications call this method when shutting down to stop the whole virtual machine. Obviously this leads to a problem in SANS as not only the corresponding client is terminated but the whole

simulation system is shut down. To the best of our knowledge the only way to prevent a client from stopping the virtual machine would be the use of a Java Security Manager which could intercept the shutdown command. Unfortunately, installing a Security Manager leads to new problems: The client calling System.exit() might be left in an undefined state, with some threads still running, which could lead to unpredictable side effects.

7.4 Mapping a Socket to its Client

SANS currently maps sockets to clients according to the name of the thread which creates the socket. While this approach has the advantage of being transparent to the client application, it also has a major drawback: If a socket is created as a reaction to an event from the graphical user interface of a client (for instance an ActionEvent when the user clicks a button) the mapping from the socket to its client fails. The reason for this problem lies in the way Abstract Windowing Toolkit (AWT) events are handled by the Java Runtime System. All GUI events of the simulation framework as well as of the clients are processed by one common AWT event dispatcher thread. Consequently, the simulation system is unable to map a socket to a client if it is created by this thread. Currently, SANS ignores sockets which cannot be assigned to a client and displays an error message.

7.5 Scalability

Depending on the complexity of the client applications, SANS may require a lot of memory and CPU power to simulate large networks. Experiments with a demo application (a rather complex serverless instant messenger for wireless networks) have shown that SANS can handle up to a low two-digit number of clients on a 3 GHz Pentium machine. If more clients are started, the system suffers from severe slowdowns as the clients do not obtain enough CPU time any more and particularly the GUI refresh functions start to fail.

8. A Usage Example

In the course of the lecture Mobile Computing at ETH Zurich the students were given the opportunity to use SANS to design and implement a multi-hop based wireless messenger application as a long-term homework. The messenger uses a proprietary protocol including network and transport layer operations to send and receive messages. Standard 802.11b network adapters in ad hoc mode are used as communication devices. The most complex part of the application handles message forwarding, using Dynamic Source Routing [4]. If the destination of a message is not within communication range of the sender, another client can be used as a relay station. (Fig. 4) shows a simple network where node S sends a message to node D using nodes A and B as relay stations forwarding the message.

In the first two years when SANS was not available to the students this exercise was solved with difficulty as there was no easy way for the students to test and debug their work. This year we received a large number of correct solutions confirming the value of SANS as a development tool for rapid prototyping. Furthermore, our system has proven to work well on all major operating systems (Windows XP/2000, Linux, and Mac OS X) and also on various hardware devices.

9. Related Work

Various very powerful network simulators are available which offer full network simulation on all layers. The most common ones are NetSim [1], GloMoSim [3], and ns-2 [2]. These tools are well suited to run customized experiments which can lead to important insights on the behavior of algorithms. As mentioned in the introduction such simulators are complex systems and often require a considerable amount of time to become familiar with. The use of such simulators, however, also has an additional significant drawback: Simulated protocols are not realized as real implementations but by logical operations. As a result the algorithms need to be reimplemented before they can be run on real hardware, which may lead to unforeseen side effects. Network emulators like the APE testbed [5], JEmu [6], or EMPOWER [7] avoid this problem by using real networks consisting of several machines to evaluate a realistic implementation of an algorithm. In JEmu and APE, the topology of the underlying network is a simple star where the simulated clients are run on the peripheral nodes whereas the management component of the emulator is run on the central instance. The management component controls the communication flow between the client machines making it possible to emulate various virtual network topologies. EMPOWER follows a more

sophisticated decentralized approach to emulate the network. The benefit of the decentralization is a better scalability at the cost of a more complex network setup. As explained before, SANS simulates test networks on one computer but supports a standard interface (java.net.MulticastSocket) for communication between clients. Consequently, implementations of algorithms developed with SANS can easily be ported to run on more sophisticated emulation systems or even on the real-world target platforms for closer evaluation.

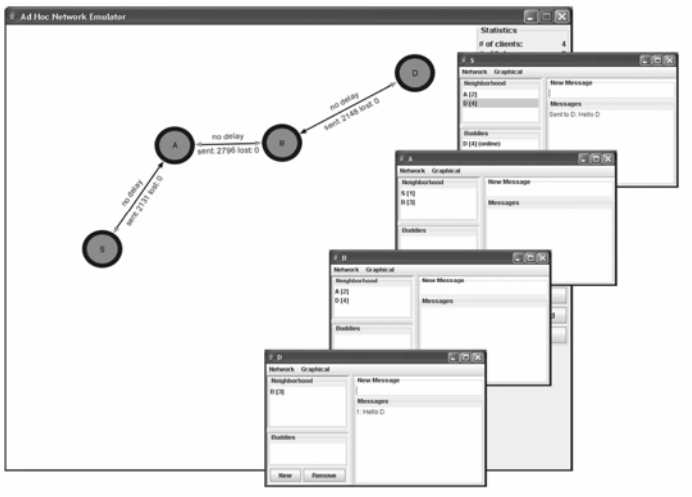

Figure 4: SANS running four instances of the serverless instant messenger.

10. Conclusion and Future Work

Developing applications for wireless networks is a difficult task as debugging is nearly impossible without the use of a testing environment. For small experiments and presentations, though, setting up one of the existing complex network simulators means a significant overhead in time and work. The SANS framework presented in this paper is a simple yet powerful alternative to those full-grown network simulators. It offers most functionality a developer may need without becoming hard to use.
Based on the feedback we have received so far we envisage to extend SANS by adding new features and by improving existing ones. Additional models for link parameters (such as packet loss depending on workload) are planned to improve the realism of simulated networks. The biggest and probably most important new feature we intend to add is a script-driven mobility model, though.

The current version of SANS can be downloaded from http://dcg.ethz.ch/projects/SANS/Simulator.jar

References

[1] A. Heybey, “The network simulator,” MIT, Technical report, September 1990.

[2] K. Fall, “NS Notes and Documentation,” *The VINT Project*, 2000. [Online]. Available: http://citeseer.nj.nec.com/fall00ns.html

[3] X. Zeng, R. Bagrodia, and M. Gerla, “GloMoSim: A Library for Parallel Simulation of Large-Scale Wireless Networks,” in *Workshop on Parallel and Distributed Simulation*, 1998, pp. 154–161. [Online]. Available: http://citeseer.ist.psu.edu/zeng98glomosim.html

[4] D. B. Johnson and D. A. Maltz, “Dynamic source routing in ad hoc wireless networks,” in *Mobile Computing*, Imielinski and Korth, Eds., vol. 353. Kluwer Academic Publishers, 1996. [Online]. Available: citeseer.ist.psu.edu/johnson96dynamic.html

[5] H. R. Lundgren, D. Lundberg, E. Nordstrm, and C. F. Tschudin, “A Large-scale Testbed for Reproducible Ad hoc Protocol Evaluations,” in *Proceedings of IEEE WCNC*, 2002.

[6] J. Flynn, H. Tewari, and D. O’Mahony, “Jemu: A real time emulation system for mobile ad hoc networks,” in *Proceedings of the First Joint IEI/IEE Symposium on Telecommunications Systems Research, Dublin, Ireland*, November 2001.

[7] L. Ni and P. Zheng, “Empower: A network emulator for wireline and wireless networks,” 2003. [Online]. Available: http://citeseer.ist.psu.edu/ni03empower.htm