A Real Time Emulation System for Ad-hoc Networks

Juan Flynn, Hitesh Tewari, Donal O'Mahony
Network and Telecommunications Research Group,
Department of Computer Science, Trinity College, Dublin 2, Ireland.
{juan.flynn, hitesh.tewari, donal.omahony} @cs.tcd.ie

Keywords: mobile, wireless, ad-hoc, real-time, emulator

Abstract
Compared to fixed networks such as the Internet, mobile ad-hoc networks (MANETs) present the routing protocol designer with many more challenges. In this paper we examine some of the tools used by designers to test their protocols. We propose that emulation of a mobile wireless network reduces the amount of time and effort in testing a routing protocol compared with either field testing or full simulation of a protocol stack. We present a system that emulates the wireless layer of a protocol stack, describe how it operates and comment on the user experience.

1. INTRODUCTION
The emerging field of ad-hoc networks has given rise to a plethora of routing protocols and architectures. Presently, there are far more protocol drafts and designs than there are implementations as testing a wireless ad-hoc network requires significant organisational skills and human resources [1]. Simulators such as NS [2] give the protocol designer a very detailed account of events that occur in a given simulated scenario. However, often this level of detail is not required and the protocol designer would favour a more hands-on simulator that would allow them to interactively engage with their protocol. Furthermore, if a protocol must be implemented twice, one time for the simulator and again for the deployment stack, this adds to the implementation workload. In this paper, we present a wireless ad-hoc network emulator that allows the protocol designer to interactively test their algorithms and implementation in place. We observe that for the ad-hoc routing protocol designer, at the initial design stage, a macro-level view of a scenario is more useful than the micro-level view offered by non-realtime simulations.

Research at Trinity College into 4th Generation Networks [3] has focussed on a scenario where the core high-speed network using evolved IP protocols serves to connect very large populations of mobile users, many of whom are equipped with wireless communicators. Users who are physically close to this infrastructure will be able to communicate not just with each other, but also to any other users via a wireless access point.

Our prototype communicators are based on commercial off-the-shelf handheld devices running the Microsoft Windows CE operating system. On these nodes, and also on the fixed nodes in our infrastructure, we have developed a generic protocol stack which allows a node to be configured, either in-advance or on-demand to deal with whatever communications environment is in force. We have developed layers to run across a variety of wireless communication links including IEEE 802.11, Infrared (IrDA), Bluetooth and our own custom built VHF radios. At the network level, we have a working implementation of Dynamic Source Routing (DSR) and other ad-hoc routing algorithms, and at the top levels, a voice telephony application as well as wireless Web browsing. Our experimental setup allows us to experiment with a real, fully functional ad-hoc network. However, at the debug and protocol design phases, we have a great need to be able to test particular mobility scenarios.

The following sections outline the common simulation tools that are available to designers of ad-hoc networking systems, the challenges faced by ad-hoc protocol designers and the design of our network emulation tool, JEmu.

2. EXISTING SIMULATION SYSTEMS
Network Simulators are frequently used by ad-hoc network protocol designers to test new routing algorithms. The IETF have a working group whose focus is to evolve mobile ad-hoc network (MANET) protocols into future Internet standards. Two simulators that are used most frequently by the IETF MANET community are GloMoSim [4] and NS. This section outlines the features of both and their suitability for use in the design and testing of routing protocols.

2.1. GloMoSim
GloMoSim is a simulation library for wireless and wired network systems developed by UCLA's Parallel Computing Laboratory as part of DARPA's Global Mobile Information Systems project. One of GloMoSim's key strengths is its use of Parsec [5], a parallel discrete event language. When simulating large numbers of nodes, GloMoSim divides the network into gridded areas and these areas are simulated as Parsec entities. When a network node sends out a message, it will at most only have to be sent out to the eight neighbouring areas. The dimensions of the gridded areas should be set such that the transmission range of a node will fit within it and its neighbours.

A simulation is set up in GloMoSim by creating an easily readable text based input file. Various parameters are entered, such as the size of the area being simulated, power range of the nodes, overall simulation times, node placement patterns, simulated bandwidth, MAC type, routing protocol, transport layer type (TCP or UDP), traffic generators and many other parameters. A visualization tool is also provided, and this allows the user to step though simulation traces.

The following points summarise GlomoSim's suitability for ad-hoc networks:

- GlomoSim is very scalable: By using Parsec, GloMoSim is very suitable for simulations involving high number of nodes due to Parsec's parallel processing capabilities. This would suit

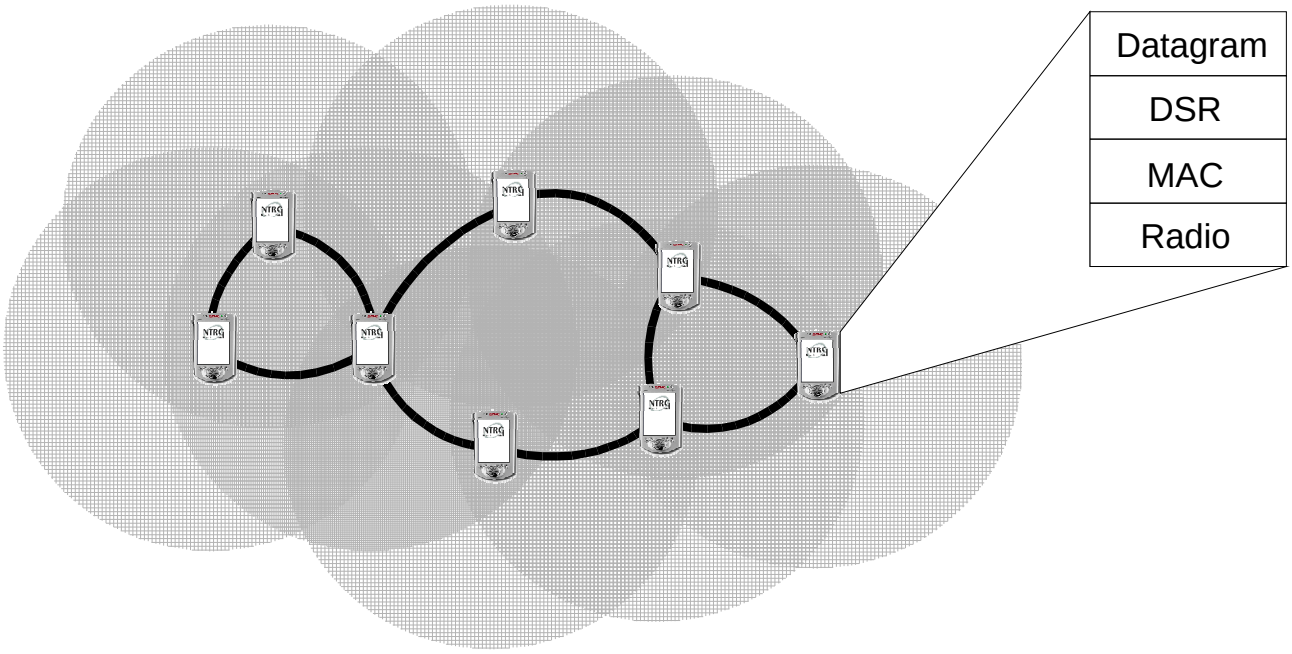

Figure 1. A mobile ad-hoc network with eight handheld nodes. The shaded region around each node indicates its transmit range, and the thick lines indicate node connectivity. Each node is running with a four layer protocol stack shown to the right.

some ad-hoc networks involving large numbers of nodes, such as Smart Dust [6].

- GlomoSim offers implementations of many different wireless layers (802.11, MACA, CSMA) and ad-hoc routing protocols (DSR, Fisheye, IP with AODV to name a few)
- GlomoSim has a Java based visualization tool that allows the user to control the execution of a simulation, view packet transmissions, view statistics and also to edit configuration files.
- GlomoSim does not directly support emulation, although a separate research effort, the Dynamic Network Emulation Backplane Project [7,8], offers plug-and-play facilities to interlink different simulation systems. By plugging GlomoSim into this common backplane, emulation facilities provided by another simulator or application could then be utilised.

2.2. NS

NS is one of the most widely known network simulators. It is an evolved variant of the REAL Network Simulator [9] from Cornell University and has been extended by a number of other institutions. It is a discrete event simulator and provides support for the simulation of TCP, routing and multicast protocols over wired and wireless networks. NS is a very flexible system that allows simulations of a wide variety of network scenarios from fixed wired networks to orbiting satellite networks.

Simulations in NS are written in C++ with an OTcl (Objective TCl [10]) API: C++ classes are used to implement the simulated processes (network protocols, sample application-level processes, propagation models etc.) and OTcl is used as the user interface to these classes. The user creates a text file in OTcl that describes the layout of the network and also when events are to occur, such as a node moving or an application transferring data. NS generates detailed tracefiles, which can be filtered with a pattern matching program (such as 'grep') and inspected by hand, or fed into a visualization tool.

It is possible to use NS as a network *emulator* at the network layer through the use of the Berkeley Packet Filter. Traffic recorded from the tracefiles generated by NS can be piped back on to a live network using the Berkeley Packet Filter [11], which will read in and write out frames to and from the fixed network - think of NS emulating a transit network. NS uses its own internal network layer node address scheme, so incoming IP addresses must be converted to this native format on the fly [12] before they can be handled by the simulator.

There are a few visualisation tools of interest to the ad-hoc network protocol designer. First, there's the Network Animator (NAM) which is available alongside NS. NAM is an animation tool for viewing network simulation traces. It supports topology layout and has various data inspection tools. Unfortunately, NAM is not able to represent node mobility yet [13]. A more suitable choice for the mobile ad-hoc network designer is Ad-hockey which is part of the Monarch project from CMU. Ad-hockey is a Perl/TK script that is most useful for generating mobile scenarios for NS. The user can draw paths of node movement and Ad-hockey can generate input files for NS from this.

The following points summarise NS's suitability for simulating ad-hoc networks:

- NS can offer a very detailed simulation of a wireless ad-hoc network. It has accurate implementations of the IEEE 802.11 standard, a full TCP/IP stack and there are a wide range of ad-hoc routing protocols implemented for NS. However, many NS users working on ad-hoc routing protocols dispense with the detailed radio propagation layers, opting for a simple 'null/mac' layer which is quicker to simulate**.**

- NS does offer some emulation capabilities, though only at the IP level. IP packets can be passed into and out of an NS simulation with the use of the Berkeley Packet Filter.
- NS's visualisation tools for ad-hoc mobile networks (such as Ad-hockey) lack a degree of interactivity needed for real time reconfiguration of an ad-hoc network.

Both GlomoSim and NS provide many tools for the Ad-hoc protocol designer to simulate their protocol in great detail. The next section discusses the need for network emulation as opposed to network simulation and the challenges facing an ad-hoc routing protocol designer.

3. THE NEED FOR EMULATION

There have been many protocols and algorithms proposed to address the wide range of issues that a mobile ad-hoc network presents. To aid comparison of different ad-hoc routing protocols, a framework document contatining qualtiative properties, quantitative metrics and network contexts that can be used to assess mobile ad-hoc network protocols was created and this became RFC 2501 [14]. Examples of qualtiative properties of a protocol include distrubuted operation, loop-freedom, demand based or proactive operation, security concerns, sleep modes (essential when considering portable battery powered units) and unidirectional link support. Quantitative metrics of a protocol include end-to-end throughput, percentage out of order delivery and several efficiency ratios. The network context parameters that describe the network include network size, network connectivity, link capacity, topological rate of change etc.

Each ad-hoc network protocol concentrates on different metrics and characteristics. So, while one protocol might perform well in a high mobility, low node, low connectivity context, it might not perform so well in a high node, low mobility context. Recent protocols such as the Zone Routing Protocol [15] or Fisheye [16] have parameters that can be tuned according to network context to improve protocol performance. Being able to invent new mechanisms in ad-hoc routing protocols to overcome the many challenges of a mobile network is as important as tuning an implemented protocol to achieve optimum efficiency. At the design stage, it's important not to get consumed in the detail of a simulation but instead to focus on the broader issues of the problem. For example, a protocol which is very efficient in one highly tested scenario might suffer from a high level problem such as looping messages or inability to detail with unidirectional

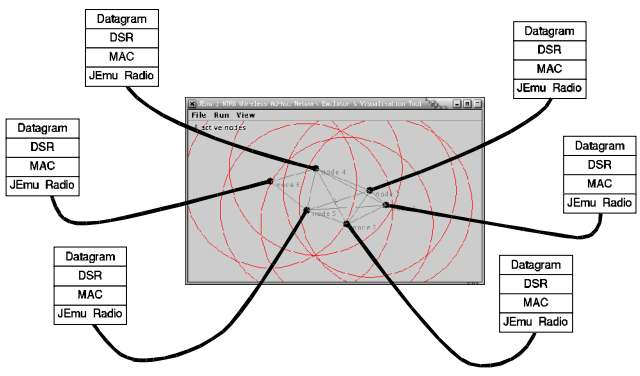

Figure 2. Six nodes connect to the JEmu Network Emulator.

nodes. [17] argues than when an ad-hoc network protocol designer is exploring a new area where many issues are unclear, the need to quickly explore a variety of alternatives can be more important that a detailed result of a specific scenario and that a more abstract simulation can also make the effects of a change in algorithm distinct, where they would be obscured by other effects in a more detailled simulation.

Instead of simulating the entire stack, which would require the duplication of a large amount of code, it is possible to simulate only the physical layer of the stack and leave the rest of the stack untouched. This way, the actual network protocols which reside higher up in the communications stack remain in place and can be tested in real time with real applications on it. And since the simulation is running in real-time with real data, it can be called an emulator.

At Trinity College, we have implemented a flexible communications stack, ad-hoc routing protocols and hardware radios that together form our mobile ad-hoc network testbed. Implementing this stack in a simulator would have been a large amount of work but compared to the person hours required to physically move about the nodes into the variety of patterns and positions needed to experiment with and validate ad-hoc network routing protocols, a simulated version would have been preferable. However, instead of simulating the whole stack, a real time simulator was made to replace only the radio components of the stack. This allowed us to continue to develop our stack codebase, and the ad-hoc routing protocols that run on it, without having to reimplement them for a simulator. Using NS with its emulation capabilities was considered, but its lack of interactive visualisation tools and 'IP datagram' level of emulation did not really lend itself suitable as a straightforward radio replacement. What was needed was an emulator that offered real time emulation of the radio environment, at a detail level of interest to ad-hoc routing protocol designers and with an interactive visualisation tool that would be designed for their needs. It is not our intention to create an accurate radio environment – such a system would be slow to run in real time and not be of immediate interest to those developing higher level protocols and applications developers higher up. (If a highly accurate radio simulation is required, the use of NS-2 or GlomoSim is still recommended.)

The key features necessary for a useful network emulator for us were:

- Ease of node placement and movement.
- Real time emulation of the radio layer.
- Integration into the NTRG stack without any changes to the layers above.
- Ability to correctly handle hidden node and exposed node situations.
- Unidirectional links and per-node transmit ranges.

To meet these needs we created a radio-replacing emulator called JEmu and the following section outlines the design of the system.

4. DESIGN OF JEMU

In a typical test scenario, a number of laptops and palmtops running the NTRG protocol stack are each connected to a small custom built VHF transceiver. The nodes are then spaced out to create the desired network topology. In order to minimise changes to the network stack, the layer of the stack that normally interfaces

	x	*y*	*tx range*
node 1	-18.2	7.2	24.0
node 2	1.0	4.2	24.0
node 3	19.6	-4.0	24.0

Table 1. Positions and transmit ranges of the nodes.

with the radio hardware is replaced with a layer that communicates to the JEmu network emulator shown in figure 2.

The JEmu wireless network emulator is a stand alone application that accepts connections from JEmu clients which are usually the lowest layer in the NTRG stack. To allow multiple, independent stacks to connect to it, the emulation engine was designed around a client/server model: each stack has a radio emulation layer (that replaces the existing layer that connects to the radio hardware) as a client that connects to the emulator server. If the emulator is run on the same machine as a client stack, this connection is local, but if the emulator is run on a different machine the client can connect to the emulator remotely over a TCP/IP network. Obviously, to keep the emulation as close to real-time as possible, the connection network should have low latency and high bandwidth, such as fast ethernet.

Messages, usually some form of datagrams, are sent down into the JEmu Radio layer in a stack and this layer sends these messages on to the JEmu emulator. Messages to and from the emulator and inside the emulator are called fragments as they may only be part of a longer packet. The emulator sends fragments back to the JEmu connection layer should there be a fragment for it to receive. This closely mimics the behaviour of the radio interface layer and the radios we use. The following sequence of events is a more detailed description of what happens when a packet is sent down the stack to the JEmu layer and this sequence is illustrated in figure 4. The nodes are positioned as indicated in table 1 by the user dragging the nodes into posititon, shown in figure 3.

Step 1. A message is passed down the stack to the JEmu Radio layer.

Step 2. The JEmu Radio layer prepends this message with the source Node ID and then is sent by UDP to the emulation engine. Figure 4 shows nodes 1 and 2 sending fragments.

Step 3. The emulator receives this message and adds on a timestamp. It also looks up the Node ID in a table and stamps the X position, Y position and current transmit power of that node onto the message. This stamped message is called a fragment. In this example, the fragment from node 1 is stamped with (x=-18.2, y=7.2, tx range=24.0, timestamp = 226) and node 2 is stamped with (x=1.0, y=4.2, tx range=24.0, timestamp = 224).

Step 4. The fragment is added to the fragment pool. Another thread, working slightly behind current time, t_c, scans the pool for the fragments within a fixed time period t_e to t_e+q, where t_e is the starting time of that time period and q is length of the time period to be considered. The emulator ensures that t_e+q will lag t_c by at least q so that each time period is just slightly in the past. If the emulator becomes over burdened by an influx of data, this lag between t_e and t_c will become more noticeable.

Step 5. The emulator iterates though the list of receiving nodes and checks to see if they are able to receive any of the fragments found within the time period t_e to t_e+q. In this example, t_e is 220

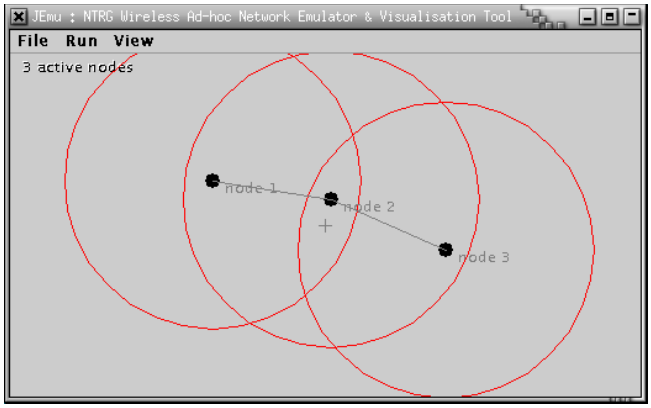

Figure 3. The three nodes are dragged into position.

and q is 10 so the two fragments with timestamps of 224 and 226 can be considered.

Step 6. If a node is able to receive a fragment then that fragment is sent to that node's JEmu Radio layer. If a node is able to receive more than one fragment then there is a collision; depending on how the user has configured the emulator, the fragments can either be scrambled, dropped or even delivered back sequentially to the JEmu Radio layer as if the collision did not occur. Finally, the

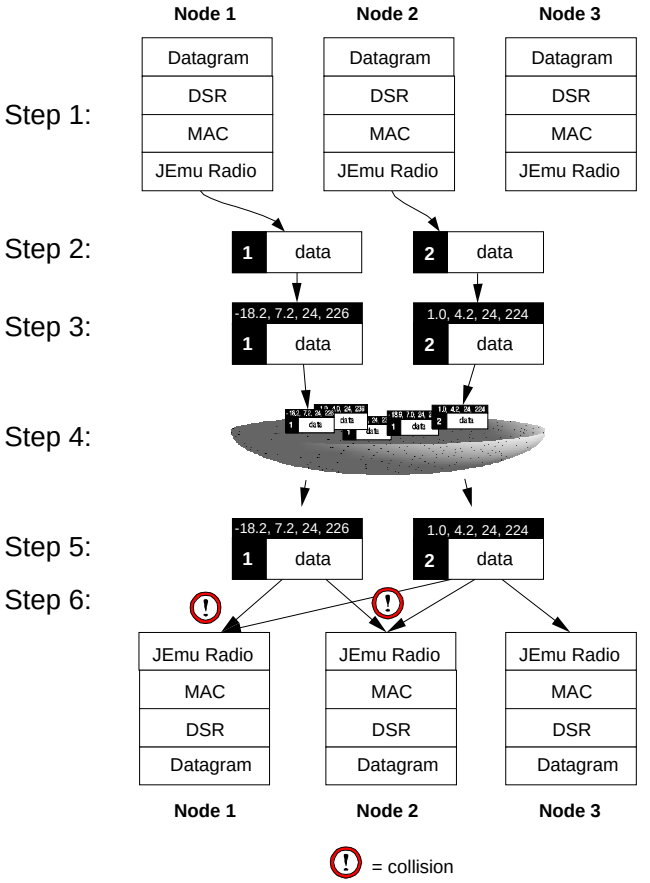

Figure 4. Sequence of events inside JEmu as nodes 1 and 2 each broadcast a fragment.

emulator expires all processed fragments (ie. with a timestamp less than t_e+q) from the pool and t_e is advanced by q. In figure 3 we can see that only node 3 can receive the tranmission from node 2. Node 1 and node 2 hear collisions as both of those nodes are within each other's transmission ranges and have transmitted a fragment at the same time (or more correctly, with the same time period). The emulator can be configured to send to the node's JEmu Radio layers both the fragments (collisions ignored), a corrupted fragment or no fragments at all.

This design allows for hidden and exposed node packet collisions and also for collisions to be 'turned off' so that a receiver sequentially hears both (or more) collided fragments. For some testing of high level routing algorithms, we have often found it useful to use the emulator with collisions turned off as at that stage of design this level of detail is not of immediate concern to us. Since the design of the emulator is receiver-oriented, in that the emulator iterates over each receiver and decides which transmissions it is able to receive for a given time segment, broadcast transmissions are handled implicity, as all transmissions by radios are broadcast. It is up to layers higher up in the stack to differentiate between broadcast, unicast or multicast transmissions (a datagram layer such as IP can do this). It is also therefore trivial to make a packet-sniffer than can be placed anywhere in the network – a receiver simply displays every packet it receives.

5. USER EXPERIENCE OF JEMU

JEmu has been designed to enable the ad-hoc routing protocol designer to create and manipulate wireless scenarios with ease. After starting the JEmu emulator, client stacks can connect to it. As a stack connects to the emulator, a node appears in the emulator's main console window; figure 5 shows a typical screenshot of the console – the user here is setting some of the visualisation options.

The user can place nodes by dragging them with the mouse. Double clicking on a node allows the user to alter a node's radio parameters, such as transmit power level. JEmu has basic automatic movement options (such as drunken walk) and explict controls for unidirectional radio behaviour.

For traffic visualisation purposes, JEmu allows the size and shape of each node to change as network traffic passes though it - as a node transmits data it gets fatter, as it receives data it gets taller. On a network with many nodes it is useful to see which nodes are being active with a quick glance at the console window; one can see waves of streching nodes as a routing protocol performs a broadcast or is establishing a path between two nodes.

6. IMPLEMENTATION & FUTURE WORK

The JEmu emulator has been implemented in Java 1.3 and has been tested under both Windows 2000 and Debian Linux. Java 1.3's HotSpot optimising bytecode compiler and virtual machine offers enough computational power to comfortably emulate a twelve node network on a 400MHz Intel Pentium II with all visualisation options turned on. The JEmu layer in the JEmu stack is written in C and compiled for a Win32 or WindowsCE platform.

JEmu has been used to emulate 12 node networks running DSR and ZRP with end-to-end streaming traffic. On a 700MHz Intel Pentium III the CPU usage of JEmu was much less than 10% indicating that it can cope with larger network emulations. JEmu has also been used in the development of an application layer point-to-point distributed name resolution system being developed by the NTRG. Emulating a network in real time will always be bounded by available processor power. Where processor intensive higher level layers in a stack are to be used, such as CODECs or

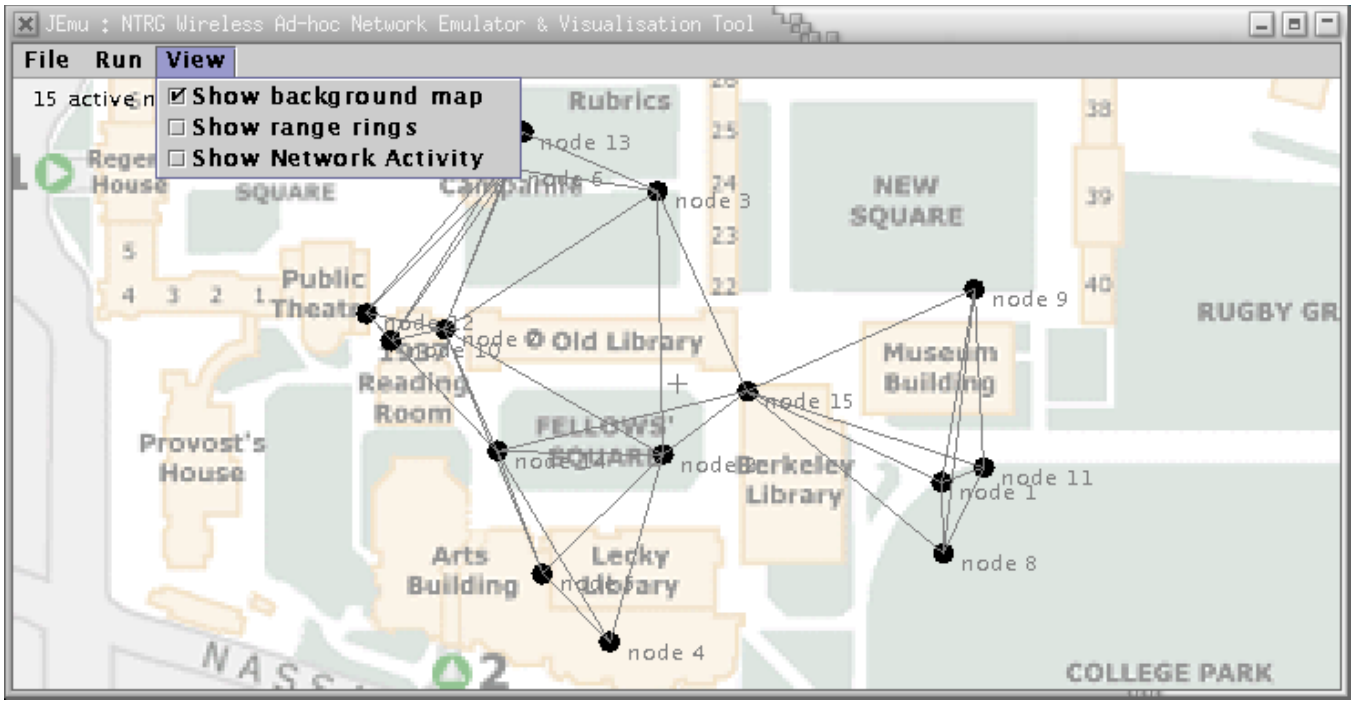

Figure 5. The JEmu console.

strong encryption, there is an advantage having JEmu nodes on individual computers instead of on the computer running the JEmu emulator.

Future work on JEmu includes parallelizing the main emulation engine and adding more facilities for node movement patterns and traffic visualisation tools. To speed up the emulator and allow much larger numbers of nodes to be emulated, parallelization of the core event processor is an option. Due to the way the JEmu Emulator works by iterating though each node to see what they can 'hear' in a given time period, as opposed to sequentially processing and resolving events, it is possible to unroll this loop into multiple parallel threads and simultaneously resolve which fragments can be heard by which nodes and which nodes will hear collisions. On a network with a large number of nodes it may be advantageous to run the emulator on a computer with multiple CPUs as the threads can be executed in parallel. Visualisation tools might include colouring node connector lines depending on traffic direction, or simple graphing tools to record traffic between nodes.

7. CONCLUSIONS

Being able to dynamically and interactively test an ad-hoc network's routing protocol at a high level is essential. At the initial design stages, being able to quickly try out new concepts and ideas is more important than analyzing the minutae of a protocol's performance in a given scenario. The NTRG have found that JEmu meets this need, more so than using exising simulators and user feedback from routing protocol designers within the NTRG towards JEmu has been very positive. While detailed simulations are essential for debugging and performance analysis of a protocol and the stack that it resides in, using a real stack but only emulating the the radio layer achieves realistic results in real time.

REFERENCES

[1] Maltz, D. A; J. Broch; D.B. Johnson. 1999. “Experiences Designing and Building a Multi-Hop Wireless Ad Hoc Network Testbed”, Tech. Rep. 99-116, School of Computer Science, Carnegie Mellon University, March.

[2] Fall, K; K. Varadhan. “ns Notes and Documentation”, The VINT Project, http://www.isi.edu/nsnam/ns

[3] O'Mahony, D.; L. Doyle. August 2001. “An Adaptable Node Architecture for Future Wireless Networks in Mobile Computing: Implementing Pervasive Information and Communication Technologies”, Kluwer series in Interfaces in OR/CS, Kluwer Publishing.

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

[5] R. Bagrodia; R. Meyer; M. Takai; Y. Chen; X. Zeng; J. Martin; H. Y. Song. 1998. “PARSEC: A Parallel Simulation Environment for Complex Systems”, *IEEE Computer*, October.

[6] Kahn J. M.; R. H. Katz; K. S. J Pister. 1999. “Next century challenges: mobile networking for 'Smart Dust'”, *Proceedings of the ACM/IEEE International Conference on mobile networking*, August.

[7] Xu, D.; G. Riley; M. Ammar; R. Fujimoto. 2001. “Split Protocol Stack Network Simulations Using the Dynamic Simulation Backplane”, *Proceedings of the ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, August.

[8] “The Dynamic Network Emulation Backplane Project”, Parallel and Distributed Simulation, Georgia Tech, http://www.cc.gatech.edu/computing/pads/netemulation, July 2001.

[9] Keshav, S.; 1988. “REAL: A network simulator”, S. Keshav, Computer Science Department Technical Report 88/472, UC Berkeley, December.

[10] Bogdanovich, P. “The Objective-Framework System: True Language Independence” (white paper), Tip Top Software, http://www.tiptop.com/Objective/WhitePaper.htmld/

[11] McCanne, S.; V. Jacobson. 1993. “The BSD Packet Filter: A new architecture for user-level packet capture”, *Proceedings of USENIX Winter Conference*, January.

[12] Fall. K. 1999. “Network emulation in the Vint/NS simulator”, *Proceedings of the 4th IEEE Symposium on Computers and Communications*, July.

[13] “NS-2 Frequently asked questions” at http://www.monarch.cs.cmu.edu/ns-faq/faq.html

[14] Corson, S. “Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations”, RFC2501, January 1999.

[15] Haas. Z.; M. Pearlman. 1998. “The Zone Routing Protocol (ZRP) for highly reconfigurable ad-hoc networks”, *Proceedings of ACM SIGCOMM 98*, Vancouver, August.

[16] Gerla, M.; G. Pei; et al, “Fisheye State Routing Protocol (FSR) for Ad-hoc networks”, Internet-Draft, November 2000.

[17] Heidemann, J.; N. Bulusu; J. Elson; C. Intanagonwiwat; K. Lan; Y. Xu; W. Ye; D. Estrin; R. Govindan. 2001. “Effects of Detail in Wireless Network Simulation”, *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference*, January.