

# EMULATION ARCHITECTURE FOR AD HOC NETWORKS

A.Giovanardi

*DI, University of Ferrara, Italy*

agiovanardi@ing.unife.it

G.Mazzini

*DI, University of Ferrara, Italy*

gmazzini@ing.unife.it

**Abstract** The paper presents an emulation architecture working in the user space useful to implement and test routing protocols for ad hoc networks. The emulator interfaces with the Simple Ad hoc siMulator (SAM) [1], where many routing protocols are present. The novelty with respect to SAM is the possibility to test routing protocols with a real exchange of signaling and data packets between the nodes present in the network. With respect to a live test, the emulator works on hosts connected each other via wired links and the wireless channel is simulated.

**Keywords:** Emulation, Ad Hoc Networks, Routing Protocols.

## 1. Introduction

Ad hoc networks have become an increasingly popular technology in the past few years. In these wireless networks nodes communicate with other nodes in their range and act as forwarders of data from nodes communicating with out-of-range nodes.

Many approaches are possible to test the ad hoc network performance: simulation, live test and emulation. **Simulation** runs a model/representation of the code in a synthetic environment, by imitating the time operations executed by a real system. **Live test** runs the real code in the real environment. **Emulation** is a simulation which involves hardware or firmware components. Simulation can be very slow; the synthetic environment may poorly represent real one. Live tests may only be possible very late in development cycle; it is often difficult or too expensive to create a real test environment of any significant size; real environment tests also tend not to be reproducible. Emulation can

give a controlled, reproducible environment for running live code in a lab-environment network. Benefits of network emulation include the ability to expose experimental algorithms and protocols to live traffic loads and to introduce real packet processing times. In many cases, a large number of hosts can be considered without impact on the final cost.

Many simulators (OPNET [2], NS2 [3], GloMoSim [4] and SAM [1]) and emulators ([5] [6] [7]) have been proposed and developed to validate the proposal of new routing protocols for ad hoc networks.

Emulation environments can be classified as follows [6]:

- *Central Control Emulators*: all nodes are connected to a central emulation server. All traffic is directed via that server, which forwards the traffic to the destinations. An example is [7].
- *Simulator Combined Emulators*: also in this case stations are connected to a central server, which decides whether packets get forwarded, delayed or dropped. With respect to the preceding case, the packets do not pass through the server, which simply manage the whole emulation, by simulating radio channel, topology and mobility. An example is [5].
- *Distributed Emulators*: as opposite to the centralized approaches each station acting as mobile node is responsible for directing and forwarding traffic. An example is: [6].

The emulation architectures can be classified also regarding their easy software portability or not. Many of the proposals need the modification of the kernel code of the operating system, or to load suitable kernel modules, with a consequent not easy portability.

In this paper we propose an emulator platform falling in the *Distributed Emulators* category and easy to implement. The architecture is fully distributed: nodes exchange signaling routing packets (according to the selected routing protocol) to update topology knowledge; then, on the basis of the run-time updated routing tables, data packets are transmitted from the source to the destination with hops on intermediated nodes. Nodes are connected via wired links and the wireless channel is simulated. Our implementation is fully developed in the user space (it does not need any change to the kernel code) and uses the libpcap library [8] and the RAW sockets [9].

To easily change the routing protocol to be tested, the emulator interfaces with SAM [1], where many kinds of unicast and multicast ad hoc routing protocols (Dijkstra, Link State, Distance Vector, DSR, AODV, AMRIS, ODMRP, AMRoute) are present. To test the emulator, we have used a simple routing protocol (Link State-LS [10]) by taking into account also the channel impairment. The final performance of the selected routing protocol have been compared with those of SAM.

## 2. Emulator Architecture

The emulator has been implemented on Linux hosts connected by means of a Fast Ethernet network. The  $N$  nodes involved in emulation have routing functions, i.e., can relay packets to other nodes. Each node in the network is identified by a logical number  $n$ , where  $n \in \{1, \dots, N\}$ . Hosts are involved in many activities (not only emulation); so, a basic real traffic is always present (relative to services and applications such as NFS, DNS, DHCP, multicast and so on). The emulation traffic is composed by signaling broadcast packets (useful to know and update the topology) and by data packets.

The emulation architecture interfaces with SAM, which is a discrete event simulator, composed by the following modules: *Traffic*, dealing with the traffic generation; *Channel*, simulating the radio propagation; *Mob*, dealing with the terminal mobility; *Radio*, simulating the hardware transceiver; *Mac* realizing the Data Link Layer; *Route*, implementing the routing protocols; *Statistics*, collecting statistics results.

Emulator interacts with the *Traffic*, *Channel*, *Mob* and *Route* modules. To have an emulation platform giving results only on the routing protocols without care of a particular MAC Layer, we have not considered the *Radio* and *Mac* modules used in SAM. This choice is also motivated by the presence of the MAC Layer of the wired system, which could become wireless if the hosts are equipped with wireless cards. So, we have created a suitable EMULATOR MAC which is simply performing a transparent interface between network card and Network Layer, making no particular medium access actions. At the Transport Layer, we only consider UDP.

A *Logical Management* block contains scripts able to configure the system. A file named *hosts* contains the MAC and IP addresses of all hosts which could be involved in the emulation. This file is read at the emulation start to perform all socket and network interface operations. Other scripts are able to manage the emulation simultaneously on all hosts, by also collecting and processing information stored in the output files to derive synthetic performance indexes.

An *Input* block is used to set parameters useful for the emulation: the number of nodes; the number of packet generated per host; the parameters to control the network interface; the maximum number of hops; the parameters to manage the network topology, i.e., the virtual host positions; the parameters to control the transmitted power and the propagation channel behavior; the parameters to manage the packet generation; the parameters identifying the routing protocol selected in the emulation.

A *Socket* block manages the frames generation and their insertion on the network, while a *Packet dump & Forwarding* block performs the dump and the relay of the packets. Finally, the *Output/Statistics* block collects parameters of interest to derive the final performance.

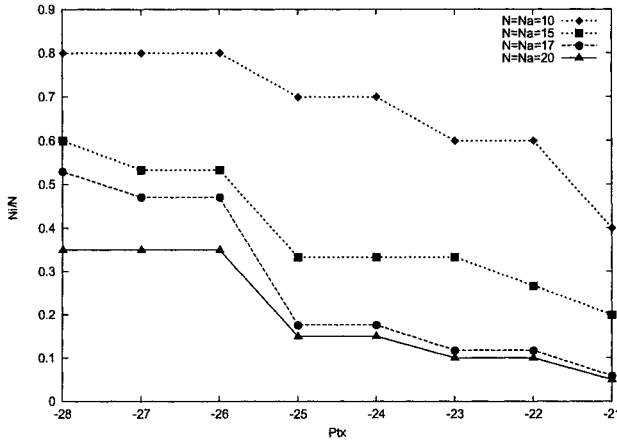


Figure 1.  $N_i/N$  as a function of  $P_{tx}$ , by varying  $N = N_a = 10, 15, 17, 20$ .

## 2.1 Implementation Details

The packets are dumped (captured at the network interface) by using the *libpcap* library [8]: *pcap\_open\_live()*, opens the network device for packet capture; *pcap\_compile()*, sets a packet filter to select the packets useful for the emulation (and to discard any other kind of packets); *pcap\_setfilter*, links the packet filter to the socket; *pcap\_loop()*, collects and processes packets; *pcap\_stats()*, collects statistics on the packet correctly dumped, discharged at the interface or by the kernel.

To insert the Ethernet frames on the network we use the RAW sockets [9], having set *ETH\_P\_ALL* as protocol (to send a not well known packet format) and having considered the socket option *IP\_HDRINCL* to make sure that the kernel knows the header is included in the data, and does not insert its own header (for example an IP header) into the packet before the payload. The broadcast routing signaling packets has been redirected to a MAC address different from the broadcast one, to avoid to flood the network with broadcast packets not interesting for the hosts not involved in the emulation.

The Ethernet frames are composed by a header including: the destination and source MAC addresses; the type field; the numbers identifying the source node  $n_s$  and the destination node  $n_d$ ; the Time To Live (TTL), decreased of 1 at each packet relaying; a timestamp, useful to evaluate the delivery time; the nexthop identifier; the transmission power. To discriminate into the network the packets involved in the emulation from those relative to other applications or protocols, we have identified and selected an ad hoc type field in the Ethernet frame (0x0020). TTL is included into the frame to drop a packet circulating

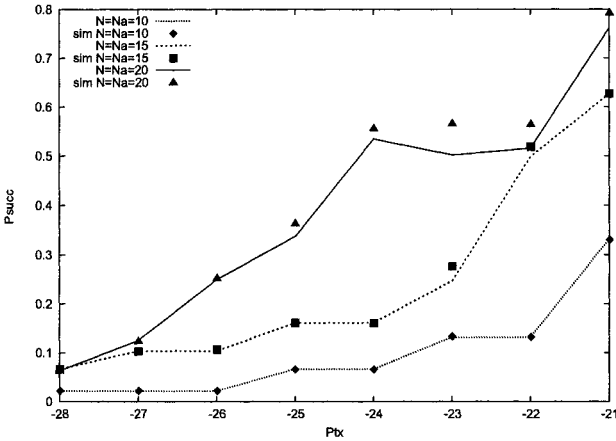


Figure 2.  $P_{succ}$  as a function of  $P_{tx}$ , by varying  $N = N_a = 10, 15, 20$ .

into the network indefinitely; furthermore, it is useful to evaluate the total number of hops experimented by a given packet.

To insert data into the frames in a machine-independent fashion, the eXternal Data Representation standard (XDR) [11] has been adopted. XDR is useful for transferring data between different computer architectures, it fits into the ISO presentation layer and support all classic C data types. The frame generation is performed trough the *Traffic* module present in SAM where Poisson, CBR, Isochronous, FTP and Video traffic can be set. By default, the number of active hosts ( $N_a$ ) is set equal to  $N$ , i.e., all hosts are generating traffic.

At the frame born, the source node  $n_s$  collects the generation time, to identify the timestamp to include in that frame. Then,  $n_s$  searches the nexthop in the path, by applying the rules corresponding to the selected routing protocol. If the nexthop exists (i.e., at least the first relaying node is reachable from the source),  $n_s$  creates the frame to send, by setting in the Ethernet header: SA= Source MAC, DA= Nexthop MAC, Type= 0x0020; and including  $n_s$ ,  $n_d$ , *TTL*, timestamp, nexthop and transmit power. After the frame creation,  $n_s$  sends the frame to the selected interface by using the RAW sockets.

The frame forwarding capability (router functionality) is started on each host and is maintained active until the end of the emulation. When a host captures a frame with type field 0x0020, it can discriminate if this frame is addressed to another node or not. In the first case it does not perform any action; in the last case it processes the frame to establish if it must be forwarded to the next hop or it has reached the final target. The first operation of the frame processing is to identify the source and destination nodes, the TTL

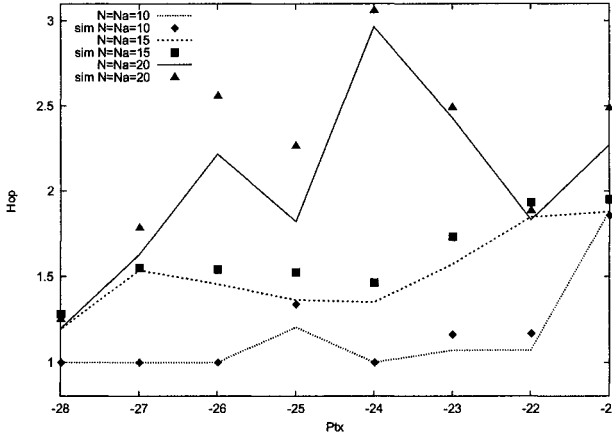


Figure 3.  $hop$  as a function of  $P_{tx}$ , by varying  $N = N_a = 10, 15, 20$ .

and the timestamp (representing the time of frame generation at the source node). Then, also the last node in the path which has relayed the frame is identified (it is often an intermediate node, i.e., not the source node). This permits to evaluate the channel impairment in the last link, and then, according to a received power threshold, to determine if the frame is correctly received or not. If the frame is correct and has not reached the final target, the local node starts the procedures to forward it to the next hop. If the frame is correct and has reached its last destination, the local node processes it to collect performance indexes. Otherwise the frame is dropped. The frame forwarding procedure follows these steps: TTL decrease and control, i.e.,  $TTL = TTL - 1$ ; if  $(TTL = 0) \rightarrow packet\ expired$ ; next hop evaluation (applying the rules of the selected routing protocol); frame regeneration with new TTL; packet insertion on the network. When a frame has reached  $TTL = 0$  it is dropped and an error message is returned to the output.

### 3. Numerical Results

All nodes are involved in the statistics collection: many output files are created, each characterized by a prefix identifying the logical number  $n$  of the node. As a consequence of the possible high number of hosts ( $N$ ) involved in the emulation, the number of output files to be processed could be high. So a suitable post-processing software has been developed to collect : **success probability** (correctly delivered packet fraction), **delivery time** (end-to-end time), **number of hops**, **number of isolated nodes**, **energy spent per byte**, **total energy spent** for each host and the relative average values on the network.

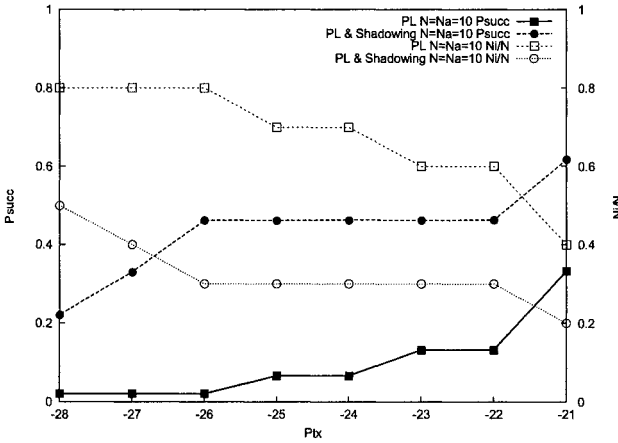


Figure 4.  $P_{succ}$  and  $N_i/N$  as functions of  $P_{tx}$ , with and without shadowing.

We have tested a simple protocol, i.e., Link State (LS) [10], by also verifying the match between emulation and SAM simulation. We consider a wireless fixed scenario with hosts located in a square room with size  $10 \times 10$ m. The propagation channel is characterized by path loss, with exponent  $\beta = 2.5$  and reference distance  $d_{ref} = 0.2$ m. In some cases shadowing has been also taken into account, with log-normal distribution and deviation  $\sigma = 6$ dB. The transmit power  $P_{tx}$  is the same for each host and for each packet sent. The minimum received power is  $P_{rx} = -76$ dBm. The traffic is Poissonian with average arrival rate  $\lambda = 10$ packet/s. 5000 packets generated per station have been considered. In the performance evaluation we have varied the number of hosts present into the network and the transmit power  $P_{tx}$ . Figures from 1 to 5 show performance indexes representing values averaged on the whole network. Figures 6 and 7 show these parameters for any single host. In figure 1, 2 and 3 we consider only path loss, while in all other Figures the cases with only path loss and with path loss and shadowing are showed.

In Figure 1 the ratio between the number of isolated nodes,  $N_i$ , and the total number of hosts present in the network,  $N$ , is reported, by varying the transmit power  $P_{tx}$  in the range  $[-28, -21]$ dBm and setting the number of hosts into the network  $N = N_a = 10, 15, 17, 20$ . The results are only relative to the emulation, since the simulator does not report this information. As expected,  $N_i/N$  decreases by increasing  $P_{tx}$  and by increasing  $N$ , i.e., by increasing the density of host in the network area.

In Figure 2 the average success probability,  $P_{succ}$ , i.e., the correctly delivered packet fraction, is reported by varying the transmit power  $P_{tx}$  as in Figure

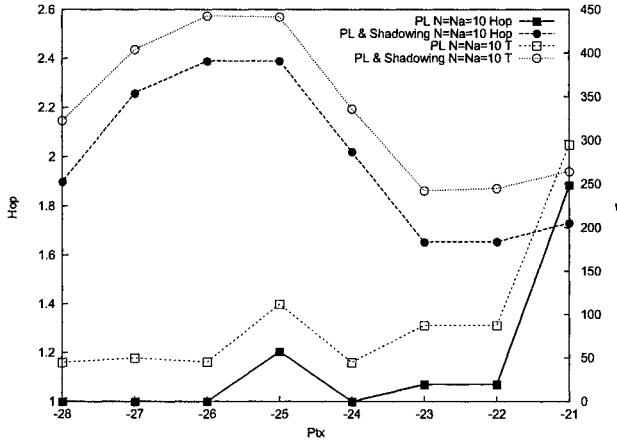


Figure 5.  $hop$  and  $T$  as functions of  $P_{tx}$ , with and without shadowing.

1 and assuming  $N = N_a = 10, 15, 20$ . In this case, simulation results are also reported. The results obtained by emulation are depicted with lines, those obtained by simulation with points. We can verify the good match. Furthermore, as expected, by increasing the transmit power, the success probability increases, since the probability to have isolated nodes decreases. By increasing the number of hosts in the network, the effect is similar to a transmit power growth, since the network becomes more dense and then nodes are closer each one (and then more reachable) with higher probability.

In Figure 3 the average number of hops to deliver a packet,  $hop$ , is reported in the same parameter condition of Figure 2. A quite good match between emulation and simulation can be verified. For high  $N$  values, by increasing  $P_{tx}$ ,  $hop$  first increases, then decreases, and this general trend is repeated more times. This can be probably explained as follows: at the begin the transmit power growth permits to eliminate zones with isolated nodes (characterized by short path, with few hops), by allowing paths longer, with higher number of hops; then, when the transmit power overcomes a given threshold, some nodes becomes directly reachable (i.e., without need of intermediate node with relaying functions) and then the number of hop decreases. On the other hand, by increasing the number of hosts in the network, the hop number increases and the effect could be similar to that of the initial transmit power growth.

In all Figures presented in the following only emulation results are shown. Furthermore,  $N = N_a = 10$  has been assumed. In Figure 4  $P_{succ}$  and  $N_i/N$  are depicted as functions of  $P_{tx}$ . The cases with and without shadowing are directly compared. We can note the same increasing and decreasing trends



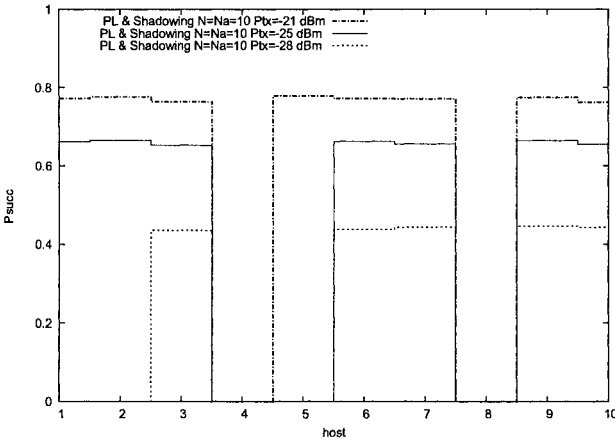


Figure 6.  $P_{succ}$  for each generic host, with shadowing.

of  $P_{succ}$  and  $N_i/N$  with  $P_{tx}$  of Figures 1 and 2. Furthermore, we can note the positive effect of shadowing which makes the network more “dense”, by mitigating the attenuation effects of the path loss. In particular, shadowing doubles the nodes in visibility and the success probability. In Figure 5 the *hop* number and the final delivery time  $T$  (in ms) are reported as functions of  $P_{tx}$ , in the same conditions of Figure 4. Shadowing allows paths more long, with higher number of hops (in some cases *hop* doubles its value). This trend is related to the effect explained above (shadowing makes the network more dense); so, a lower number of isolated nodes allows to reach farer hosts, with higher number of hops on intermediate nodes. Note that, a part a scale factor, the delivery time and *hop* trends are quite the same.

Figure 6 shows  $P_{succ}$  for each generic host, with shadowing and by varying  $P_{tx} = -28, -25, -21$ dBm. As expected, by increasing  $P_{tx}$  the success probability increases for all hosts. Furthermore, it is possible to note that some hosts can not deliver packets, since they are probably isolated. The number of hosts in this condition decreases with the  $P_{tx}$  growth. The behavior depicted in this Figure allows to better understand if the average values depicted above (obtained by also considering unreachable nodes) are really representative or not. The last graph (Figure 7) depicts the number of forwarded packets for each generic host, having considered  $P_{tx} = -28, -25, -23, -21$ dBm. Note that this parameter increases when  $P_{tx}$  increases and that for any different level of transmission power the number of hosts performing routing action is limited (about the half of the hosts involved in the emulation).

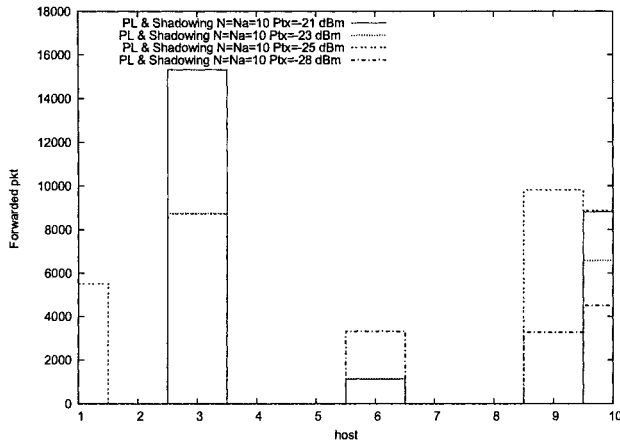


Figure 7. Number of forwarded packets for each generic host, with shadowing.

## Acknowledgments

This work is developed under MURST/MIUR Pattern Project and Regional Insebala Project.

## References

- [1] P.Bergamo, D.Maniezzo, A.Giovanardi, G.Mazzini, M.Zorzi, "Distributed Power Control for Power-aware Energy-efficient Routing in Ad Hoc Networks," in Proc. of EW2002, pp. 237-243, Florence, Italy, Feb. 2002.
- [2] "OPNET commercial tool", <http://www.opnet.com>.
- [3] "The Network Simulator NS2," <http://www.isi.edu/nsnam/ns/>.
- [4] "GloMoSim," <http://pcl.cs.ucla.edu/projects/glomosim/>.
- [5] K. Fall, "Network emulation in the VINT/NS simulator", in Proc. of IEEE Computers and Communications, 6-8 July 1999, pp. 244-250.
- [6] M. Matthes; H. Biehl; M. Lauer; O. Drobnik, "MASSIVE: An Emulation Environment for Mobile Ad-Hoc Networks" in Proc. of Wireless On-demand Network Systems and Services, (WONS 2005), 19-21 Jan. 2005, pp. 54-59.
- [7] J. Flynn, H. Tewari, D. O'Mahony, "Jemu: a Real Time Emulation System for Mobile Ad hoc Networks", in Proc. of the First Joint IEI/IEE Symposium on Telecommunications System Research, Dublin, Ireland, Nov. 2001.
- [8] S. McCanne, C. Leres and V. Jacobson. libpcap, 1994. <ftp://ftp.ee.lbl.gov/libpcap.tar.Z>
- [9] R. Stevens, "UNIX Network Programming, Volume 1: Networking APIs - Sockets and XTP", 1998, Prentice Hall PTR.
- [10] A.S. Tanenbaum, "Computer Networks", Prentice Hall, 1989.
- [11] RFC 1014 - XDR: External Data Representation standard, <http://www.faqs.org/rfcs/rfc1014.html>.