Emulating small scale MANET topologies

Guillaume Valadon*, Ryuji Wakikawa†, Hiroshi Esaki‡
Graduate School of Information Science and Technology, Tokyo University
Engineering Building 3, 403, Esaki Laboratory
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
*Email: guedou@hongo.wide.ad.jp
‡Email: hiroshi@wide.ad.jp
†Keio University, Department of Environmental Information,
5322 Endo Fujisawa Kanagawa 252-8520, Japan
Email: ryuji@sfc.wide.ad.jp

***Abstract*— While network simulators are commonly used to test MANET routing protocols, no similar tool exist to perform evaluations of implementations with real hardware. In this paper, a solution to emulate MANET topologies and to assist implementations debugging is described. It uses well known UNIX networking features to enhance and ease interoperability tests, as well as MANET demonstrations. The system was the key tool that helps to validate WIDE's project implementation of OLSR6.**

I. Introduction

In research environments, network simulators are commonly used to design and analyse the performance of new protocols. For instance, they can reproduce the behaviour of wireless interfaces, manage and move many nodes, and fake network topologies. Nevertheless, no such tools are widely available when performing real life evaluation with regular devices.

During the implementation and testing phase of a MANET routing protocol, experiments can become a real headache such as dealing with wireless interfaces. To ease the debugging part, different physical topologies need to be emulated to trigger specific protocols behaviour. In OLSR, for example, the simplest topology leading to TC messages generation is a chain of nodes paired together.

The system presented in this paper improve the way to debug and to test protocols. No configuration is performed on nodes except usual IP addresses configurations and routing daemons setups. The topology can be entirely controlled during experiments in order to focus on its results and not on the network operation.

We have two motivations to develop this emulator. The first point is to illustrate MANET to students while keeping the demonstration as real and simple as possible. The second point is to ease the validation of WIDE's project OLSR6[2] implementation. Several bugs were found and corrected using a step by step approach. The network topology is frozen when a bug is identified, and change to a new one after the code correction.

This paper is organised as follow. First, requirements of small scale MANET testbeds are given. Then in section III common testbeds setups used in research environment are described. In section IV and following, a description of the proposed system is given as well as its usage during OLSR6 debugging. Limitations of the system are then discussed to conclude on future enhancements.

II. Requirements of small scale MANET testbeds

The following list summarise important requirements discussed later in this section. They are the key elements that led to the design of the proposed solution.

- IPv6 support
- simple setup and management of nodes
- wire based to get rid of wireless troubles
- automatic topology changes
- traffic monitoring in a single entity

MANET routing protocols are designed to run on top of wireless technologies, yet during real experiments many problems often arise while trying to setup wireless interfaces to use IBSS (ad-hoc) mode. In fact, it can be really difficult to configure using cards from multiple vendors, or from different countries. In some situations like interoperability tests where wireless performance is not really needed, it is better to rely on communications using wired interfaces.

The testbed should work with off the shelf hardware in order to keep the overall cost of the platform reasonable and ease the maintenance. It also permits to validate MANET routing protocols implementations with real hardware and to use it for other kind of experiments. In the best case, it should also work with hardware commonly found in laboratories.

The management of the system must be simple, as well as the configuration and the distribution of protocol implementation's updates. The main focus of the proposed solution is to make debugging as simple as possible. Using a centralised architecture, traffic flow between nodes can be analysed in a single entity which can be the developer's laptop.

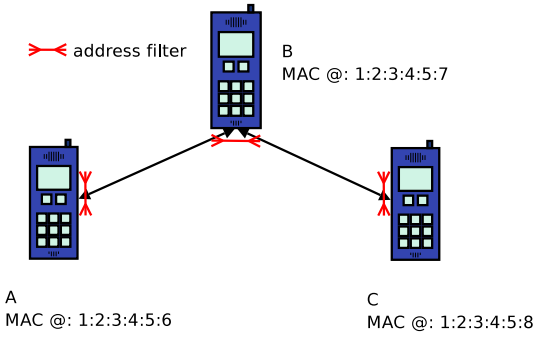

Fig. 1. Emulated network

```
# iptables -P INPUT DROP
# iptables -A INPUT -m mac -s 1:2:3:4:5:6 \
  -j ACCEPT
```

Fig. 2. Filters in node C

Topology changes caused by nodes mobility must be configured easily. The system should manage them automatically, still giving enough control to the experimenter if some specific reshape of the network is needed. Moreover, the system should be able to replay topologies change or nodes movements. This control over the topology is really important while trying to hunt and fix precise bugs. After the correction of bugs, it is essential to go back to the network's state that triggered it in order to check the reparation.

III. Common testbeds setups

In research environments, popular MANET testbeds involve addresses filtering. This section describe two filtering concepts achieving topology emulations. Following descriptions had been written with wired link-layer technologies in mind to simplify testbeds configurations. Regarding filters, without protocols concern, the optimal design is to deny all addresses and to selectively allow nodes to talk to each other. Filters are smaller and easier to maintain this way.

A. Filtering

1) In node filtering: This solution implies setting up access lists on every nodes involved in the network in order to emulate the topology. A good strategy is to use a master node that will update new rule sets every time a topology change is needed. Otherwise, nodes with dual IP stacks are convenient to separate the management plane from the routing experiment. IPv4 is use to update rules while the protocol test is done using IPv6. See figures 1 and 2 for an iptables[1] example using MAC addresses filtering.

The main drawback of this filtering is that it relies on operating systems filtering capabilities. Moreover, in some situations, the use of this filter is not realistic. For instance, during debugging sessions, developers will not agree to grant root access to their laptop.

[1]command allowing packet filtering on Linux

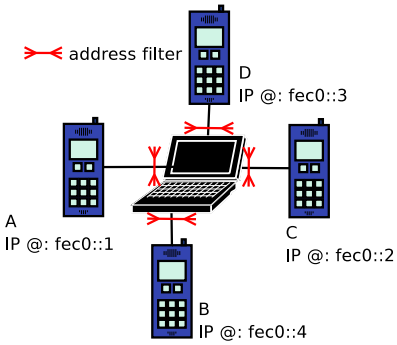

Fig. 3. Nodes are connected to the emulator using cross cables

```
in fa 0/1
switchport protected
switchport access vlan2
exit

in fa 0/2
switchport access vlan2
exit

in fa 0/3
switchport protected
switchport access vlan2
exit
```

Fig. 4. Star topology with Cisco's IOS

2) Centralised filtering: In this environment, all nodes are connected to a device that emulates network topologies and performs addresses filtering as in figure 3. Since no specific configuration are required on nodes, every operating system can get involved during the experiment.

Most of Ethernet switches can filter both IP packets and Ethernet frames. However, there is some restrictions as it is sometime impossible to filter MAC addresses on ports' egress traffic. As OLSR HELLO messages are transmitted using Ethernet broadcast, outbound filtering is inefficient. Moreover, the delay between rules input with switches' command line interface and their effective use by built-in the operating system can be quite long and inefficient while changing topology.

As star topology can easily be created using VLAN features of switches. Figure 4 shows an example of such setup using a Catalyst switch from Cisco. Considering OLSR, the emulated network is too simple as no TC message while be generated.

B. MAC or IP filtering

The biggest requirement for this testbed is to support both IPv4 and IPv6. Obviously packet filtering involving link layer addresses meet this requirement. It also gives nodes a true view of the topology by isolating neighbors traffic.

IP filters leads to more complicated filters as denying on an IP address basis will work to exchange OLSR message but will also deny other kind of communications as ICMP or UDP messages. Attention is needed to carefully setup them. During debugging sessions, as these filters can not catch non IP based protocols like ARP, output of network analysis tools is somehow misleading.

From an IPv6 perspective, it is easier not to filter on IP addresses. ICMPv6 Neighbour Solicitation messages must be filtered during OLSR simulations, while allowing Echo Request ones. Moreover, during IPv6 link layer addresses resolution, IPv6 can send packet using the unspecified addresses `"::"` making filters harder to setup.

Regarding these problems, MAC addresses filtering is an efficient solution to achieve accurate emulation of physical topology providing nodes a correct view of the network. As they only receive packets they need to, the MAC address filtering is a nice solution for debugging purpose.

IV. Proposed Solution

The topology emulator presented in this paper is designed to run on hardware commonly found in a lab. The intent is to provide researchers a cheap solution for protocols testing and evaluation in small scale testbeds. It supports both IPv4 and IPv6 using link layer topology emulation. As it is node independent, multiple devices and Operating system can be involved in the network.

A. System Overview

The proposed topology emulator is based on centralised filtering and use the following components:

1) addresses and interfaces information discovery
2) Ethernet interfaces bridging
3) topology generator
4) MAC addresses filtering

It runs on a computer where many Ethernet interfaces are available. Nodes participating in the experiment need to be plugged with a cross cable to one of the numerous network interface of the emulator as in figure 3.

In order to setup filters, relations between IP addresses, MAC addresses and interfaces names need to be established. Giving an IP class as 192.168.0.0/24 or fec0::0/64, ARP requests or respectively Neighbours Solicitation requests are sent to every interfaces plugged to the emulator. As only on node is plug into an interface, it is straightforward to get a tuple including MAC and IP addresses as wells as interface name for each node. Figure 5 shows one example of such relations.

```
fec0::1 eth0 00:09:6b:a0:a6:71
fec0::2 eth2 00:11:24:79:8e:82
fec0::3 eth3 00:07:40:fb:ad:6a
fec0::4 eth5 00:07:40:02:42:68
```

Fig. 5. MAC, IP, interfaces relations

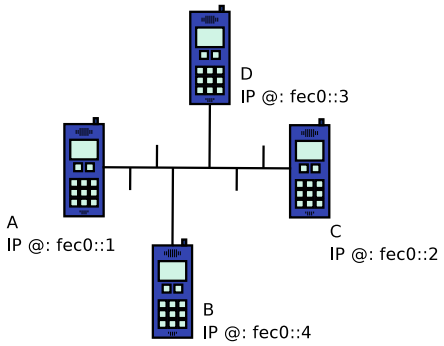

Fig. 6. Bus topology emulated with Linux bridging system

Once these relations are discovered, all nodes are connected together using an Ethernet bridge. At this stage, every nodes can talk to each other. For OLSR, every nodes are neighbors and can receive all HELLO messages generated in the network. Figure 6 shows the emulated topology corresponding to figure 3.

The emulated network is then generated based on a random network topology as seen in figure 7. All communications on the bridge are first denied, then according to the topology, Ethernet communications are allowed using MAC based filters. To ease debugging, a simple mobility model was implemented. At each iteration of the script, one node is selected randomly, one of its link will be deleted and a new one added.

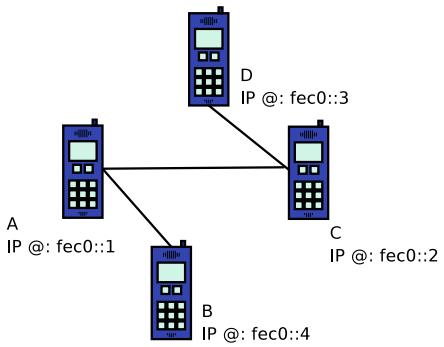

Fig. 7. A topology generated by the emulator

```
# ebtables -A FORWARD -o eth0 \
  -s 00:11:24:79:8e:82 \
  -j ACCEPT # Allow fec0::2 -> fec0::1

# ebtables -A FORWARD -o eth2 \
  -s 00:09:6b:a0:a6:71 \
  -j ACCEPT # Allow fec0::1 -> fec0::2
```

Fig. 8. An example of ebtables based MAC filters

B. Implementation

The OLSR6 debugging session was conducted with a regular IBM X31 laptop using USB Ethernet interfaces. It runs Debian testing with Linux 2.6.8. The process of gathering relations between MAC, IP addresses and interfaces names was automated thanks to scapy[4], a powerful packet manipulation tool written in python.

The Ethernet bridge is done with the *brconfig* command. A python script is used to emulate the topology, provide nodes movements and build MAC filters. Figure 8 shows an example of such filters using the *ebtables* command.

This system was designed to work on Linux 2.6 with its bridging and MAC filtering features. However, as BSD systems offers similar functionalities, the system should work with small modifications. At least OpenBSD is expected to work with *pf* (Packet Filter) and *brconfig*[2].

V. OLSR6 DEBUGGING FEEDBACK

This section briefly described how the topology emulator was used during debugging sessions of the OLSR6 implementation in zebra. The main object is to explain how bugs were found and how these sessions were performed.

At first, all developers' laptops were plugged to the emulator using cross cables. All of them had two interfaces: one to the emulator, and on to the lab's regular network. Although its is not mandatory, the later one was used to access the CVS server and synchronise bug fixes. IPv6 site local addresses were configured in the emulator network.

During the first topology test, strange entries were detected in the routing table, as well as weird IP addresses in HELLO messages. The topology in figure 9 was though frozen and the bug investigated. It appeared to be an endianess problem triggered on some architecture. We managed to discover this issue as many different OS (MAC OS X, *BSD, Linux) can be used with the topology emulator. After fixing the bug, the test was start again with the same topology to verify result of modifications.

Routing table entries were still strange. Some nodes were unreachable. In order to get a global view of the system,

[2]Ethernet bridge management command

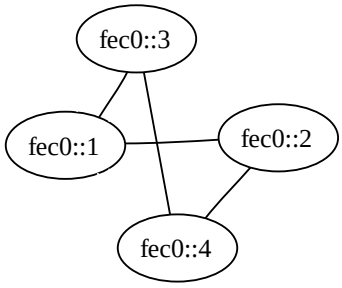

Fig. 9. First topology used during debugging

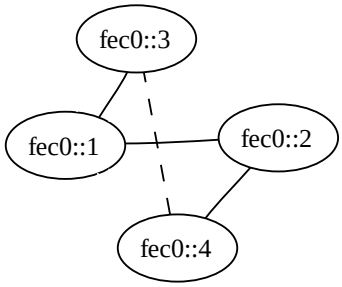

Fig. 10. After a movement, the link between fec0::3 and fec0:4 was deleted

Ethereal was launch on the bridge interface of the emulator. It appeared that no TC messages were sent by MPRs, whereas they were necessary according to the topology in figure 10. After this bug was corrected, it appears that TC messages were not forwarded by MPR.

A new node movement was then launched. This time, TC messages were corrupted. The neighbours list was still reflecting the previous network state. The neighbours cache was not correctly flushed. Applying this basic step by step method, the implementation was successfully debugged. The topology was frozen after a bug's discovery and the new code tested against the same one until the problem was solved.

When this debugging session was conducted, each developer could concentrate on bug solving and compile new versions of the code on their own laptop. Different operating systems and devices were used at the same time to host the implementation.

If fewer people participate in the experiment, same operating systems and similar devices should be use to ease the management of the testbed. It is possible for a developer to access nodes from the emulator without changing the topology. Debugging and compilation of the code can be done on the emulator. Binaries are uploaded to nodes using commands such as *scp*. If public keys are properly distributed, this part is straightforward.

During this experiment, the topology emulator was tested in a realistic research environment. Whereas no specific node management component was specifically developed, no problem were encountered while upgrading or configuring nodes.

To conclude, the following list shows features which were especially useful during the debugging session:

1) freeze the topology
2) monitor all traffic on the main node
3) compile and upload from a central authority

VI. Limitations and workarounds

As explained in the previous section, the topology emulator proved its utility during debugging sessions of OLSR6 implementations. However, this environment is not designed to conduct realistic performance analysis of protocols. Bandwidth and delays values are not reasonable to reflect MANET using wireless links. It is possible to rectify this problem using *netem*[1] on Linux to emulate network properties such a delay, loss or packet rates. Although, this solution is still not thoroughly validated, it could provide a good environment to study applications behaviour in OLSR networks.

In the current version, the topology emulator only provides symmetric links. If this feature is really required during tests, it is, of course, possible to change MAC filters by hand. This can be troublesome while validating OLSR implementations as there is no convenient way to make links asymmetric.

Using real hardware to conduct experiments can sometimes be problematic if a lot of nodes are required. On Linux, emulators and virtual machines using the TUN/TAP[3] system can be used to simulate nodes. As this system provides userland applications with an Ethernet interface, this virtual nodes can be used seamlessly with the topology emulator. The proposed system was tested with five laptop and two QEMU[3] CPU emulators. User-mode Linux is also a good alternative for Linux based nodes.

A. Future work

The actual system is somehow limited because it lacks a good user interface. Actually, it is only possible to generate the topology and to move one node at each iteration. A necessary feature to conduct precise experiments is a better interface to manage links on the fly. During a simulation, it will help implementors to add and delete links, move a specific node, and enable support of asymmetric links.

As all traffic go through the topology emulator, it is possible to analyse OLSR messages and check their validity according to topology informations. In the network described by figure 7, this system can verify if HELLO messages send from node C include B and D as neighbors. It could also check if node A forwards a valid TC message issued by C. This is somehow out of the scope of the proposed topology emulator but this is a good start to add automated OLSR6 conformance and interoperability tests to the Tahi Project[5].

[3]a virtual network device for userland applications

VII. Conclusion

In this paper, a topology emulator for small scale MANET is described. It was heavily used to debug WIDE's project implementation of OLSR6. A test was conducted using twelve nodes. Two of them were emulated thanks to QEMU and others were real devices such a laptops, zaurus, or soekris boards.

The developed system supports both IPv4 and IPv6, allows easy setup of MAC addresses filters, manages topologies automatically, and provides an efficient solution to monitor nodes activity in a single location. It meet all requirements for small scale MANET testbed as described in section II. As expected, all these elements revealed their utility during experiments.

This work can be considered as a first step to perform automated interoperability tests of MANET routing protocols. As the emulator can receive all traffic and knows the topology, it could easily check if sent packets are valid. On the other hand, it could also start new thoughts for specific MANET emulators. The emergence of dedicated hardware to simulate MANET with realistic performance will really help to validate implementations as well as evaluating new applications for this kind of network.

References

[1] *netem - Network Emulator*, `http://developer.osdl.org/shemminger/netem/`
[2] R. Wakikawa and A. Tuimonen and T. Clausen, *IPv6 Support on Mobile Ad-hoc Network*, `http://www.ietf.org/internet-drafts/draft-wakikawa-manet-ipv6-support-00.txt`
[3] *QEMU processor emulator* `http://www.qemu.org`
[4] *Scapy*, `http://www.cartel-securite.fr/pbiondi/projects/scapy/`
[5] *TAHI Project*, `http://www.tahi.org/`