Nomadic Communications Labs

Alessandro Villani
avillani@science.unitn.it

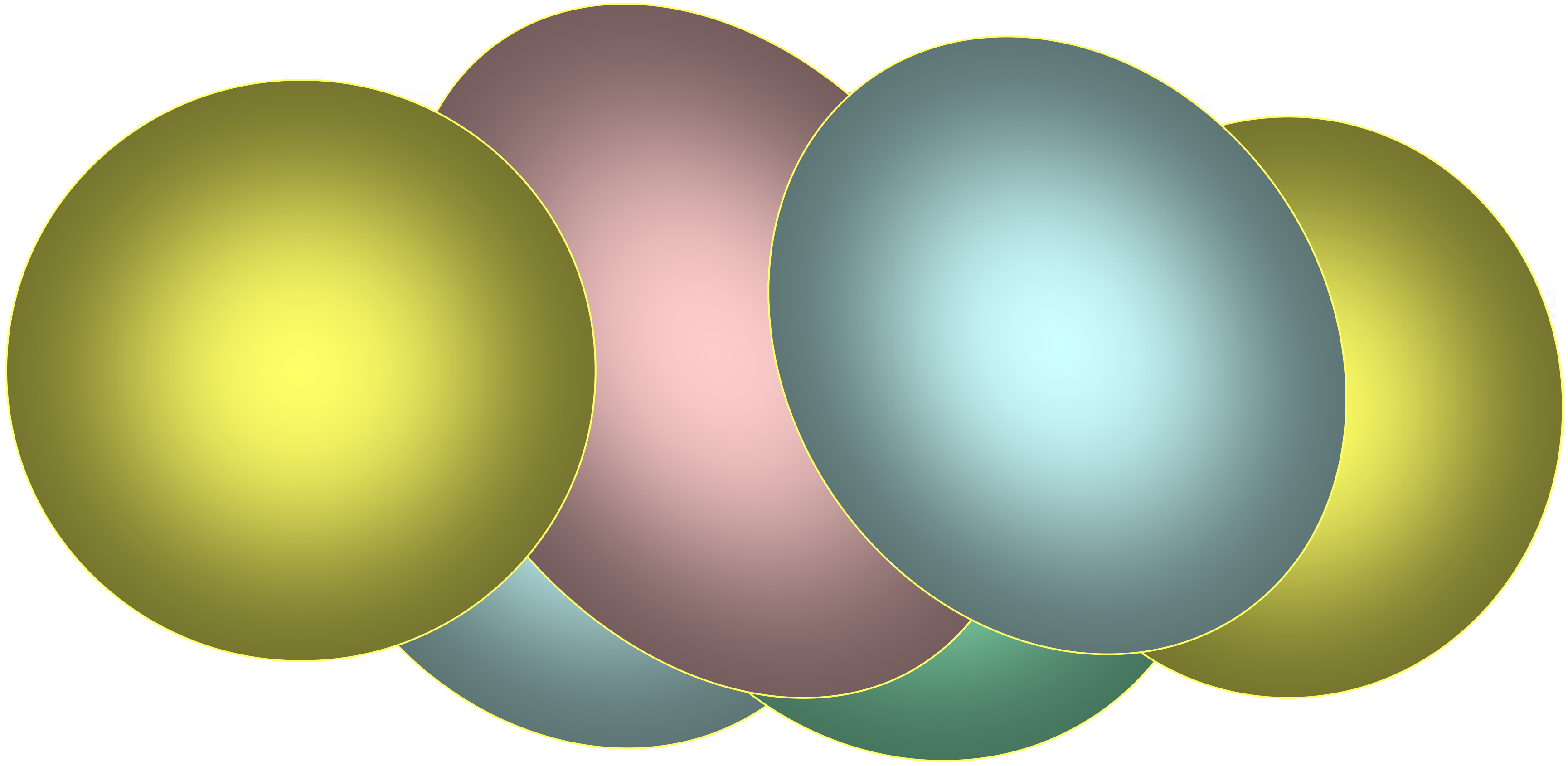
Ad-Hoc
And
Wireless Mesh Network

Routing Protocol & Mesh Network

- Wireless mesh networks bring greater flexibility, increased reliability, and improved performance over conventional wireless LANs
- The main characteristic of wireless mesh networking is the communication between nodes over multiple wireless hops on a meshed network graph

A Mesh – Ad-hoc network

- Ad-Hoc can be meshed
  - non single broadcast channel
  - multi-hop require routing



Routing Protocol & Mesh Network

- Efficient routing protocols provide paths through the wireless mesh and react to dynamic changes in the topology, so that mesh nodes can communicate with each other even if they are not in direct wireless range
- Intermediate nodes on the path will forward the packets to the destination

Routing Protocol & Mesh Network

- IEEE created the 802.11s working group to develop a standard for mesh network
- In the meantime there are a lot of network protocol currently available. Some of them are:
  - AODV
  - OLSR
  - B.A.T.M.A.N.
  - BABEL
- OLSR is the main candidate to be included in 802.11s standard

Routing Protocol

- There are three type of routing protocols:
  - **Reactive**: we search a path between nodes when there is a data to send. No wasting of network bandwidth, best suited for network where the data path change very fast
  - **Proactive**: actively establish and maintain data path both if a data has to be sent or not. Lower latency, but require a higher number of packets to be exchanged
  - **Hybrid**: the protocol use reactive and proactive routing in different situation. The hybrid protocols are more complex to implement.

Routing Protocol

- Lo Cigno will explain the algorithms and how they works
- In our labs we will try to configure a testbed with our laptop and play with the Ad-Hoc wireless mesh network

OLSR

Routing Protocol: OLSR

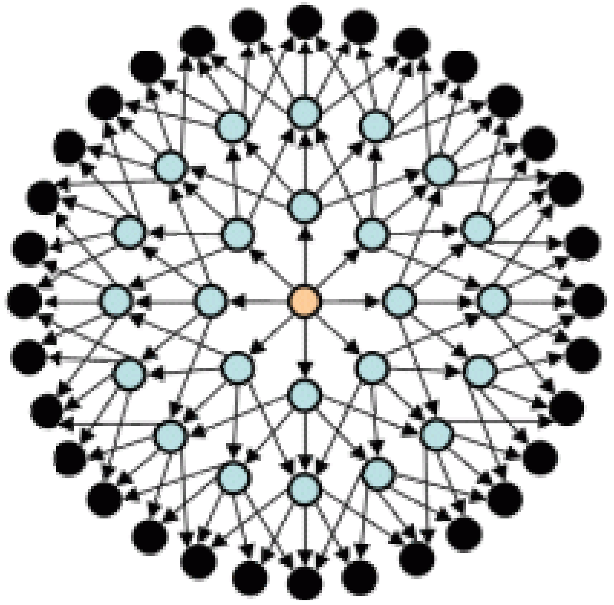
- OLSR: *Optimised Link State Routing*
- OLSR is a routing protocol for mobile ad-hoc networks
- Information are available at URL:
  - http://www.olsr.org/
- OLSR is defined in the RFC 3626:
  - http://www.ietf.org/rfc/rfc3626.txt

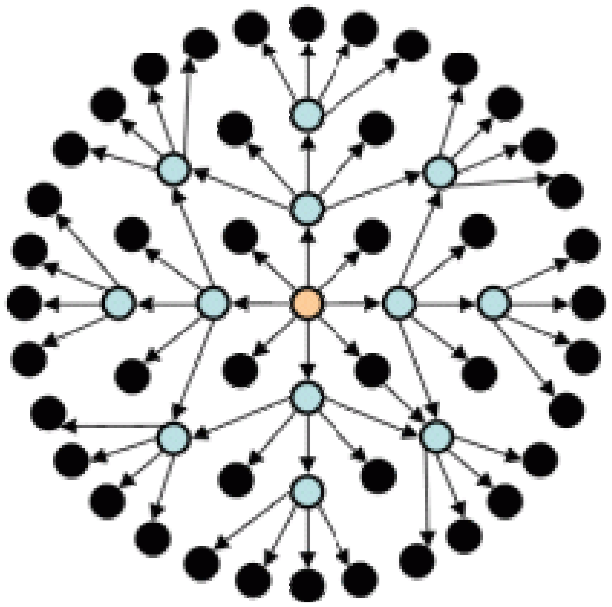
Routing Protocol: OLSR

- Proactive, link-state routing protocol
- Based on the notion of Dynamic MultiPoint Relay (MPR)
- Each node *N* selects from its neighbors an MPR(*N*) set such that all two-hop neighbors of *N* are covered by (one-hop neighbors) of MPR(*N*)
- The idea is to:
  - Reduce flooding overhead
  - Provide optimal flooding distances

Routing Protocol: OLSR

- As an examples:
  - Left image: standard flooding
  - Right image: only MPR nodes (light blue) forward the messages





Routing Protocol: OLSR

- Look at the configuration files:

  /etc/olsrd.conf
- Verify the configuration:
  - Change the debug level
  - Change the interface name
  - Set the IP Version you plan to use (4)

Routing Protocol: OLSR

- To run OLSRD on our laptop, define a script like the following:

```
#!/bin/sh
ifconfig eth1 down
iwconfig eth1 mode ad-hoc channel 11 essid
  "TEST-OLSR"
ifconfig eth1 up
ifconfig eth1 192.168.13.32 netmask
  255.255.255.0 broadcast 192.168.13.255
/usr/sbin/olsrd -d 9
```

- Don’t forget:
  - Use different IP addresses on all the client of your ad-hoc network

Routing Protocol: OLSR

- You should obtain something like:

```
*** olsr.org - 0.5.6-r4 ***
 Build date: 2009-06-02 00:57:55 on vernadsky
 http://www.olsr.org

Parsing file: "/etc/olsrd.conf"
 *** olsrd configuration ***
Debug Level      : 9
IpVersion        : 4
No interfaces    : ALLOWED
TOS              : 0x10
OlsrPort          : 0x2ba
RtTable          : 0xfe
RtTableDefault   : 0x00
Willingness      : 7
IPC connections  : 0
   Host 127.0.0.1
Pollrate         : 0.10
NIC ChangPollrate: 2.50
TC redundancy    : 2
MPR coverage     : 5
LQ level         : 2
LQ fish eye      : 1
LQ Dijkstra limit: 3, 3.00
LQ aging factor  : 0.100000
LQ algorithm name: default
NAT threshold    : 1.000000
```

Routing Protocol: OLSR

```
Clear screen     : no
Interfaces:
 dev: "wlan0"
   IPv4 broadcast           : 255.255.255.255
   Mode           : mesh
   IPv6 addrtype            : global
   IPv6 multicast site/glbl : ff05::15/ff0e::1
   HELLO emission/validity  : 6.00/600.00
   TC emission/validity     : 0.50/300.00
   MID emission/validity    : 10.00/300.00
   HNA emission/validity    : 10.00/300.00
   Autodetetc changes       : yes
Not using hysteresis
```

BATMAN

Routing Protocol: BATMAN

- BATMAN: *Better Approach To Mobile Ad hoc Network*
- BATMAN is a routing protocol for multi-hop ad-hoc mesh networks
- Information are available at URL:
  - http://www.open-mesh.net/
- An IETF draft of the protocol is available at URL:
  - http://datatracker.ietf.org/doc/draft-wunderlich-openmesh-manet-routing/

Routing Protocol: BATMAN

- Proactive routing protocol
- Decentralized knowledge of routing information:
  - No single node has the route to all destinations
  - Each node only maintains the general direction toward the destination and relays the data to the best next-hop neighbor

Routing Protocol: BATMAN

- To establish the general direction toward a destination:
  - Better link will provide faster and more reliable communication
  - Every node periodically sends out broadcast message (Originator Messages) to advertise its existence

Routing Protocol: BATMAN

- Look at the configuration files:

  /etc/default/batmand
- Verify the configuration:
  - Change the debug level
  - Change the interface name

Routing Protocol: BATMAN

- To run BATMAN on our laptop, define a script like the following:

```
#!/bin/sh
ifconfig eth1 down
iwconfig eth1 mode ad-hoc channel 11 essid
  "TEST-BATMAN"
ifconfig eth1 up
ifconfig eth1 192.168.13.33 netmask
  255.255.255.0 broadcast 192.168.13.255
batmand -d 4 eth1
```

- Don’t forget:
  - Use different IP addresses on all the client of your ad-hoc network

Routing Protocol: BATMAN

- You should obtain something like:

```
Interface activated: eth1
Using interface eth1 with address 192.168.13.32 and broadcast address 192.168.13.255
B.A.T.M.A.N. 0.3.2 (compatibility version 5)
debug level: 4
[          0] schedule_own_packet(): eth1
[          0]
[        880] Received BATMAN packet via NB: 192.168.13.33, IF: eth1 192.168.13.32 (from OG:
   192.168.13.33, via old OG: 192.168.13.33, seqno 5, tq 255, TTL 50, V 5, IDF 0)
[        880] Creating new originator: 192.168.13.33
[        880] updating last_seqno: old 0, new 5
[        880] Creating new last-hop neighbour of originator
[        880] bidirectional: orig = 192.168.13.33   neigh = 192.168.13.33   => own_bcast =  0,
   real recv =  0, local tq:   0, asym_penalty:   0, total tq:   0
[        880] schedule_forward_packet():
[        880] forwarding: tq_orig: 0, tq_avg: 0, tq_forw: 0, ttl_orig: 49, ttl_forw: 49
[        880] Forward packet: rebroadcast neighbour packet with direct link flag
[        880]
[        950]
[        960] Forwarding packet (originator 192.168.13.33, seqno 5, TQ 0, TTL 49, IDF on) on
   interface eth1
[        960]
[        960] Received BATMAN packet via NB: 192.168.13.32, IF: eth1 192.168.13.32 (from OG:
   192.168.13.33, via old OG: 192.168.13.33, seqno 5, tq 0, TTL 49, V 5, IDF 1)
[        960] Drop packet: received my own broadcast (sender: 192.168.13.32)
[        960]
[       1090]
```

Routing Protocol: BATMAN

```
[      1100] Sending own packet (originator 192.168.13.32, seqno 1, TQ 255, TTL 50, IDF off)
   on interface eth1
[      1100] schedule_own_packet(): eth1
[      1100] count own bcast (schedule_own_packet): old = 0, [      1100] new = 0
[      1100] ------------------ DEBUG ------------------
[      1100] Forward list
[      1100]     192.168.13.32 at 2171
[      1100] Originator list
[      1110]   Originator  (#/255)         Nexthop [outgoingIF]:   Potential nexthops
[      1110] No batman nodes in range ...
[      1110] ---------------------------------------------- END DEBUG
[      1110]
[      1110] Received BATMAN packet via NB: 192.168.13.32, IF: eth1 192.168.13.32 (from OG:
   192.168.13.32, via old OG: 192.168.13.32, seqno 1, tq 255, TTL 50, V 5, IDF 0)
[      1110] Drop packet: received my own broadcast (sender: 192.168.13.32)
[      1110]
[      1210] Received BATMAN packet via NB: 192.168.13.33, IF: eth1 192.168.13.32 (from OG:
   192.168.13.32, via old OG: 192.168.13.32, seqno 1, tq 0, TTL 49, V 5, IDF 1)
[      1210] count own bcast (is_my_orig): old = 0, [      1210] new = 1
[      1210] Drop packet: originator packet from myself (via neighbour)
[      1210]
[      1790] Received BATMAN packet via NB: 192.168.13.33, IF: eth1 192.168.13.32 (from OG:
   192.168.13.33, via old OG: 192.168.13.33, seqno 6, tq 255, TTL 50, V 5, IDF 0)
[      1790] updating last_seqno: old 5, new 6
[      1790] bidirectional: orig = 192.168.13.33   neigh = 192.168.13.33   => own_bcast =  1,
   real recv =  1, local tq: 255, asym_penalty:  12, total tq:  12
```

Routing Protocol: BATMAN

```
[      1790] update_originator(): Searching and updating originator entry of received packet,
[      1790] Updating existing last-hop neighbour of originator
[      1790] update_routes()
[      1790] Route to 192.168.13.33 via 192.168.13.33
[      1790] Adding new route
[      1790] Adding route to 192.168.13.33 via 0.0.0.0 (table 66 - eth1)
[      1790] schedule_forward_packet():
[      1790] forwarding: tq_orig: 12, tq_avg: 12, tq_forw: 11, ttl_orig: 49, ttl_forw: 49
[      1790] Forward packet: rebroadcast neighbour packet with direct link flag
[      1790]
[      1860]
[      1870] Forwarding packet (originator 192.168.13.33, seqno 6, TQ 10, TTL 49, IDF on) on
   interface eth1
[      1870]
[      1870] Received BATMAN packet via NB: 192.168.13.32, IF: eth1 192.168.13.32 (from OG:
   192.168.13.33, via old OG: 192.168.13.33, seqno 6, tq 10, TTL 49, V 5, IDF 1)
[      1870] Drop packet: received my own broadcast (sender: 192.168.13.32)
[      1870]
[      2170]
[      2170] ------------------ DEBUG ------------------
[      2170] Forward list
[      2170]     192.168.13.32 at 2171
[      2170] Originator list
[      2170]   Originator  (#/255)         Nexthop [outgoingIF]:   Potential nexthops
[      2170]   192.168.13.33 ( 12)   192.168.13.33 [      eth1], last_valid: 1790:
[      2170]    192.168.13.33 ( 12)
[      2170] ---------------------------------------------- END DEBUG
[      2170]
```

Routing Protocol: BATMAN

```
[      2180]
[      2190] Sending own packet (originator 192.168.13.32, seqno 2, TQ 255, TTL 50, IDF off)
   on interface eth1
[      2190] schedule_own_packet(): eth1
[      2190] count own bcast (schedule_own_packet): old = 1, [      2190] new = 1
[      2190]
[      2190] Received BATMAN packet via NB: 192.168.13.32, IF: eth1 192.168.13.32 (from OG:
   192.168.13.32, via old OG: 192.168.13.32, seqno 2, tq 255, TTL 50, V 5, IDF 0)
[      2190] Drop packet: received my own broadcast (sender: 192.168.13.32)
[      2190]
[      2280] Received BATMAN packet via NB: 192.168.13.33, IF: eth1 192.168.13.32 (from OG:
   192.168.13.32, via old OG: 192.168.13.32, seqno 2, tq 10, TTL 49, V 5, IDF 1)
[      2280] count own bcast (is_my_orig): old = 1, [      2280] new = 2
[      2280] Drop packet: originator packet from myself (via neighbour)
[      2280]
[      2870] Received BATMAN packet via NB: 192.168.13.33, IF: eth1 192.168.13.32 (from OG:
   192.168.13.33, via old OG: 192.168.13.33, seqno 7, tq 255, TTL 50, V 5, IDF 0)
[      2870] updating last_seqno: old 6, new 7
[      2870] bidirectional: orig = 192.168.13.33   neigh = 192.168.13.33   => own_bcast =  2,
   real recv =  2, local tq: 255, asym_penalty:  24, total tq:  24
[      2870] update_originator(): Searching and updating originator entry of received packet,
[      2870] Updating existing last-hop neighbour of originator
[      2870] update_routes()
[      2870] schedule_forward_packet():
[      2870] forwarding: tq_orig: 24, tq_avg: 18, tq_forw: 23, ttl_orig: 49, ttl_forw: 49
[      2870] Forward packet: rebroadcast neighbour packet with direct link flag
[      2870]
```

BABEL

Routing Protocol: BABEL

- BABEL is proactive routing protocol
- It is based on a loop-free Distance Vector Algorithm
- Information are available at URL:
  - http://www.pps.jussieu.fr/~jch/software/babel/
- An IETF draft of the protocol is available at URL:
  - https://datatracker.ietf.org/doc/draft-chroboczek-babel-routing-protocol/

Routing Protocol: BABEL

- Babel uses history-sensitive route selection:
  - If there are more than one route, the selected one is the already established path
- Babel execute a reactive update and force a request for routing information when it detects a link failure from one of its neighbors

Routing Protocol: BABEL

- Look at the configuration files:

  /etc/babeld.conf

- Verify the configuration, put something like:

```
interface eth1 wired false
```

Routing Protocol: BABEL

- To run BABEL on our laptop, define a script like the following:

```
#!/bin/sh
iwconfig eth1 mode ad-hoc channel 11 essid
  “TEST-BABEL"
ifconfig eth1 up
ifconfig eth1 192.168.13.32 netmask
  255.255.255.0 broadcast 192.168.13.255
babeld -d 5 eth1
```

- Don’t forget:
  - Use different IP addresses on all the client of your ad-hoc network

Routing Protocol: BABEL

- You should obtain something like:

```
Adding network eth1.
Got 213:ceff:fed9:4952:8b39:c64b:2345:0 17699 1271282059 from babel-state.
Noticed ifindex change for eth1.
Noticed status change for eth1.
Netlink message: [multi] (msg -> "" 0), [multi] (msg -> "" 0),
Netlink message: [multi] (msg -> "" 0), [multi] (msg -> "found address on interface eth1(3):
   fe80::213:ceff:fed9:4952
" 1),
Netlink message: [multi] (done)

Sending hello 27317 (400) to eth1.
sending request to eth1 for any.
Noticed IPv4 change for eth1.
Sending self update to eth1.
Sending update to eth1 for any.

Checking kernel routes.
Netlink message: [multi] (msg -> "found address on interface lo(1): 127.0.0.1
" 1), [multi] (msg -> "found address on interface eth1(3): 192.168.13.32
" 1),
Netlink message: [multi] (msg -> "found address on interface lo(1): ::1
" 1), [multi] (msg -> "" 0),
Netlink message: [multi] (done)

Netlink message: [multi] (msg -> "" 0), [multi] (msg -> "" 0), [multi] (msg -> "" 0), [multi]
   (msg -> "" 0), [multi] (msg -> "" 0), [multi] (msg -> "" 0), [multi] (msg -> "" 0),
   [multi] (msg -> "" 0),
```

Routing Protocol: BABEL

```
Netlink message: [multi] (done)

Netlink message: [multi] (msg -> "Add kernel route: dest: ::ffff:192.168.13.0/120 gw: ::
   metric: 0 if: eth1 (proto: 2, type: 1)" 1), [multi] (msg -> "" 0), [multi] (msg -> "" 0),
   [multi] (msg -> "" 0), [multi] (msg -> "" 0), [multi] (msg -> "" 0), [multi] (msg -> ""
   0), [multi] (msg -> "" 0),
Netlink message: [multi] (done)

  (flushing 12 buffered bytes on eth1)
Sending hello 27318 (400) to eth1.
  (flushing 20 buffered bytes on eth1)
Sending hello 27319 (400) to eth1.
Sending self update to eth1.
Sending update to eth1 for 192.168.13.32/32.
  (flushing 1 buffered updates on eth1 (3))
sending request to eth1 for any.
  (flushing 60 buffered bytes on eth1)
Entering main loop.
Creating neighbour fe80::224:d6ff:fe71:a7e0 on eth1.
Sending hello 27320 (400) to eth1.
Sending ihu 65535 on eth1 to fe80::224:d6ff:fe71:a7e0.
Received hello 13577 (400) from fe80::224:d6ff:fe71:a7e0 on eth1.
  (flushing 24 buffered bytes on eth1)
Sending hello 27321 (400) to eth1.
Sending ihu 1023 on eth1 to fe80::224:d6ff:fe71:a7e0.
Received ihu 65535 (1200) from fe80::224:d6ff:fe71:a7e0 on eth1 for fe80::213:ceff:fed9:4952.
```

Routing Protocol: BABEL

```
My id 02:13:ce:ff:fe:d9:49:52 seqno 17700
Neighbour fe80::224:d6ff:fe71:a7e0 dev eth1 reach 8000 rxcost 1023 txcost 65535.
192.168.13.32/32 metric 0 (exported)
Received hello 13578 (400) from fe80::224:d6ff:fe71:a7e0 on eth1.
  (flushing 24 buffered bytes on eth1)
Sending hello 27322 (400) to eth1.
Sending ihu 511 on eth1 to fe80::224:d6ff:fe71:a7e0.
sending unicast request to fe80::224:d6ff:fe71:a7e0 for any.
Sending unicast ihu 511 on eth1 to fe80::224:d6ff:fe71:a7e0.
Received ihu 1023 (1200) from fe80::224:d6ff:fe71:a7e0 on eth1 for fe80::213:ceff:fed9:4952.
Received ihu 1023 (1200) from fe80::224:d6ff:fe71:a7e0 on eth1 for fe80::213:ceff:fed9:4952.
Received nh 192.168.13.33 (1) from fe80::224:d6ff:fe71:a7e0 on eth1.
Received router-id 02:24:d6:ff:fe:71:a7:e0 from fe80::224:d6ff:fe71:a7e0 on eth1.
Received update for 192.168.13.33/32 from fe80::224:d6ff:fe71:a7e0 on eth1.
kernel_route: add 192.168.13.33/128 metric 0 dev 3 nexthop 192.168.13.33
Netlink message: (ACK)
Sending update to eth1 for 192.168.13.33/32.

My id 02:13:ce:ff:fe:d9:49:52 seqno 17700
Neighbour fe80::224:d6ff:fe71:a7e0 dev eth1 reach c000 rxcost 511 txcost 1023.
192.168.13.32/32 metric 0 (exported)
192.168.13.33/32 metric 2042 refmetric 0 id 02:24:d6:ff:fe:71:a7:e0 seqno 46328 age 0 via
   eth1 neigh fe80::224:d6ff:fe71:a7e0 nexthop 192.168.13.33 (installed)
```

The Report

The report

- Setup an Ad-Hoc network with 2/3/4/… laptops
- Test at least two of the Multi-Hop routing protocol
- Test the throughput using netperf/iperf and using ping to verify the number of hop
  - Try to setup a testbed with 1, 2, 3, … hops
  - Verify the bandwidth for all the possible couple of destination (1, 2, 3, … hops)

The report

- Optional:
    - Evaluate the ratio between 1 hop and 2/3/… hops throughput obtained in the previous test.
    - Run the previous test changing the rate of the wireless card involved into the test.
    - Evaluate again the ratio between 1 hop and 2/3/… hops throughput.
    - There is any difference?

The report

- Optional:
  - Run a 2 hops test with just 3 laptops
  - Run the same test as before, using 4 laptops (you have two laptops available ad intermediate node)
  - There is any difference in the performance? How many times the routes changes during the second test?

The report

- Optional:
  - In a 2/3 hop scenario stop one of the node involved in the test and verify how long it takes to find the new route
- Optional:
  - Implement the same topology don’t using iptables but moving the laptops around the buildng. There is any changes in the throughput?

The report

- An interesting starting point:
  - M. Abolhasan, B. Hagelstein, J. C.-P. Wang. Real-world performance of current proactive multi-hop mesh protocols

The report

- When you assign an IP address to the wireless interface, Linux insert a default route for the corresponding network:

```
mylaptop> sudo ifconfig eth3 192.168.10.100
mylaptop> route -n
Kernel IP routing table
Destination     Gateway   Genmask          Flags Metric Ref   Use Iface
192.168.10.0    0.0.0.0   255.255.255.0    U     0      0       0 eth3
```

- After the configuration of the wireless, remove this route, to assure the correct behavior of the routing protocols:

```
mylaptop> sudo route del -net 192.168.10.0 netmask 255.255.255.0
```

The report

- Given 3 nodes: A, B, C
- Using OSLR, we want to force A to communicate with C going through B
- On laptop A:

  iptable –A INPUT –s $x_c.y_c.z_c.w_c$ –p UDP --source-port 698 –j DROP

- On laptop C:

  iptable –A INPUT –s $x_a.y_a.z_a.w_a$ –p UDP --source-port 698 –j DROP

- Other solution:

  iptable –A INPUT –m mac --mac-source $A^cB^c:C^cD^c:E^cF^c:G^cH^c:I^cL^c:M^cN^C$ –p UDP --source-port 698 –j DROP

  iptable –A INPUT –m mac --mac-source $A^aB^a:C^aD^a:E^aF^a:G^aH^a:I^aL^a:M^aN^a$ –p UDP --source-port 698 –j DROP

The report

- Given 3 nodes: A, B, C
- Using BATMAN, we want to force A to communicate with C going through B
- On laptop A:

  iptable -A INPUT -s $x_c.y_c.z_c.w_c$ -p UDP --source-port 1966 -j DROP
- On laptop C:

  iptable -A INPUT -s $x_a.y_a.z_a.w_a$ -p UDP --source-port 1966 -j DROP
- Other solution:

  iptable -A INPUT -m mac --mac-source $A^cB^c:C^cD^c:E^cF^c:G^cH^c:I^cL^c:M^cN^C$ -p UDP --source-port 1966 -j DROP

  iptable -A INPUT -m mac --mac-source $A^aB^a:C^aD^a:E^aF^a:G^aH^a:I^aL^a:M^aN^a$ -p UDP --source-port 1966 -j DROP

The report

- To drop/reject packets (so we force the use of a multi-hop path):

  iptable -A INPUT -m mac --mac-source $A^1A^0:B^1B^0:C^1C^0:D^1D^0:E^1E^0:F^1F^0$ -j DROP

- To accept packets:

  iptable -A INPUT -m mac --mac-source $A^1A^0:B^1B^0:C^1C^0:D^1D^0:E^1E^0:F^1F^0$ -j ACCEPT

- To clear the iptables:

  ```
  iptable -F
  ```