Investigating Routing-Protocol Characteristics with MLC

(MeshLinuxContainers: LXC, IP-tools, & co.

Outline

- System overview
  - Why: Test Protocol functionality, performance, cost, ..
  - Components: LXC, brctl, ebtables, tc, ip, top, tcpdump, wireshark,...
- Tutorial (hands on, open end...)
  - Getting started: Init, simple net topology, monitoring
  - Diving Inside (part I)
  - Network and Protocol Characteristics
    - Network: size, complexity, dynamics
    - Protocol: overhead, -convergence time, robustness
  - Quick protocol overview: OLSR, Babel, BMX6
  - Diving inside (partII)
    - Experiment I: Size vs overhead
    - Experiment II: Dynamics vs overhead & convergence

System Overview

- Why:
  - Create 100-nodes network on your laptop
  - Test Protocol functionality, performance, cost, ..
  - Reproducable, controllable, easy, but not real :-(
- What: Nothing new!
  - Scripts exploiting existing Linux technology and tools
- HOW:
  - Virtualization: LXC - LinuxContainers
  - Network Emulation:
    - brctl. ebtables, tc, vconfig
  - Monitoring
    - top, tcpdump, wireshark, ...
- Where: git clone git://qmp.cat/mlc.git

System Overview

- **LXC** - LinuxContainers: virtualization/contextualization
  - Shared filesystem from template (mother LXC)
    - Individual directories ReadWrite (etc, root, var)
    - Mount heavy directories ReadOnly (usr, lib, src)
- **Network emulation**
  - **brctl** - bridge control: connecting interfaces
    - 1 control network, 2 testing network channels
  - **ebtables** – iptables for Layer2: set virtual links
  - **tc** – traffic control: packet loss, delay, bandwidth
- **Network slicing**
  - **vconfig** – vlan tagging: create logically distinct channels for simultaneous experiments (eth0.11, eth1.12
- **Monitoring**
  - **top**, **tcpdump**, **wireshark,**, **vconfig**

System Overview

LXC

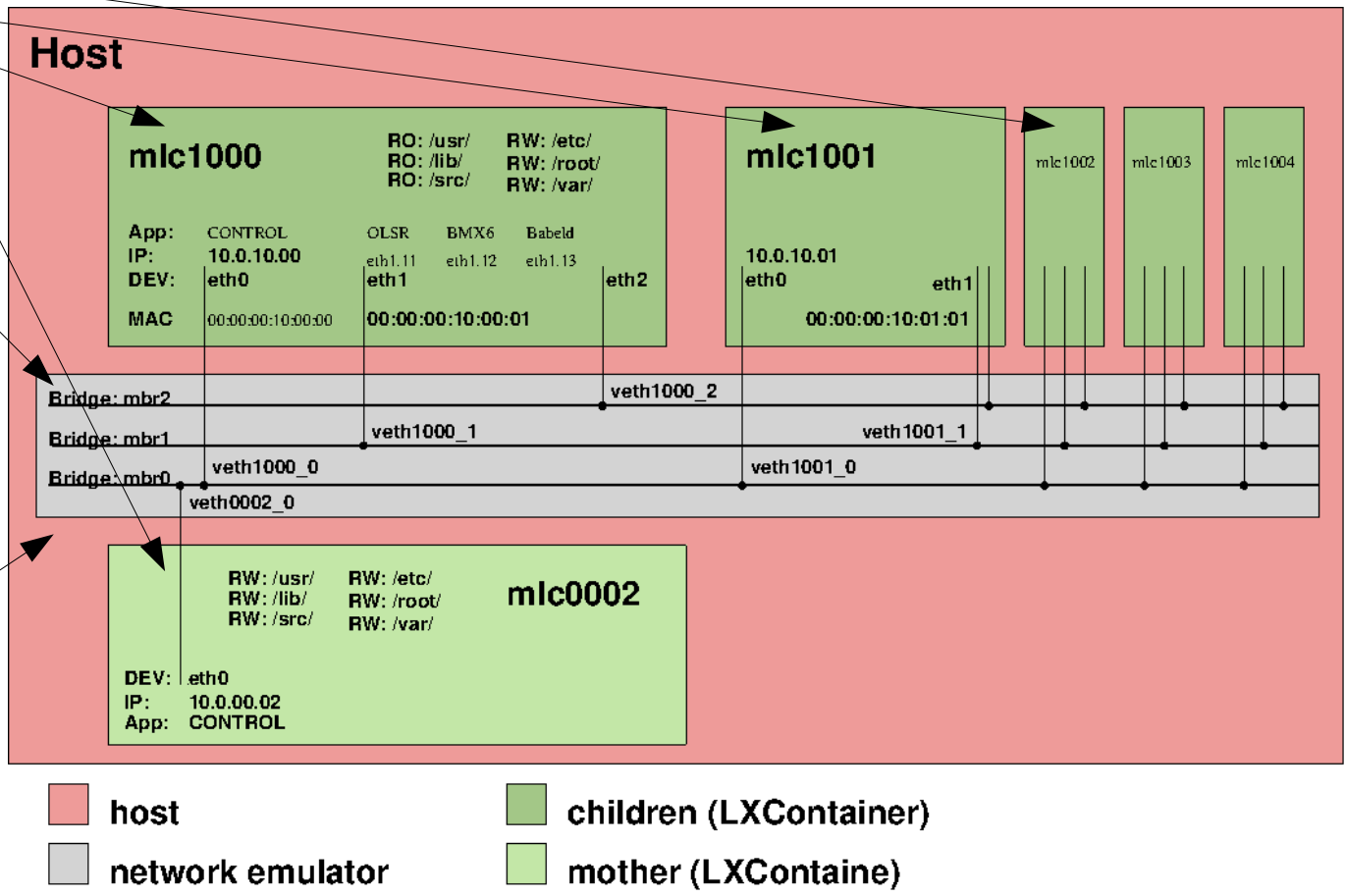
- **Childs RO/RW**
- **Mother RW/RW**

Networking

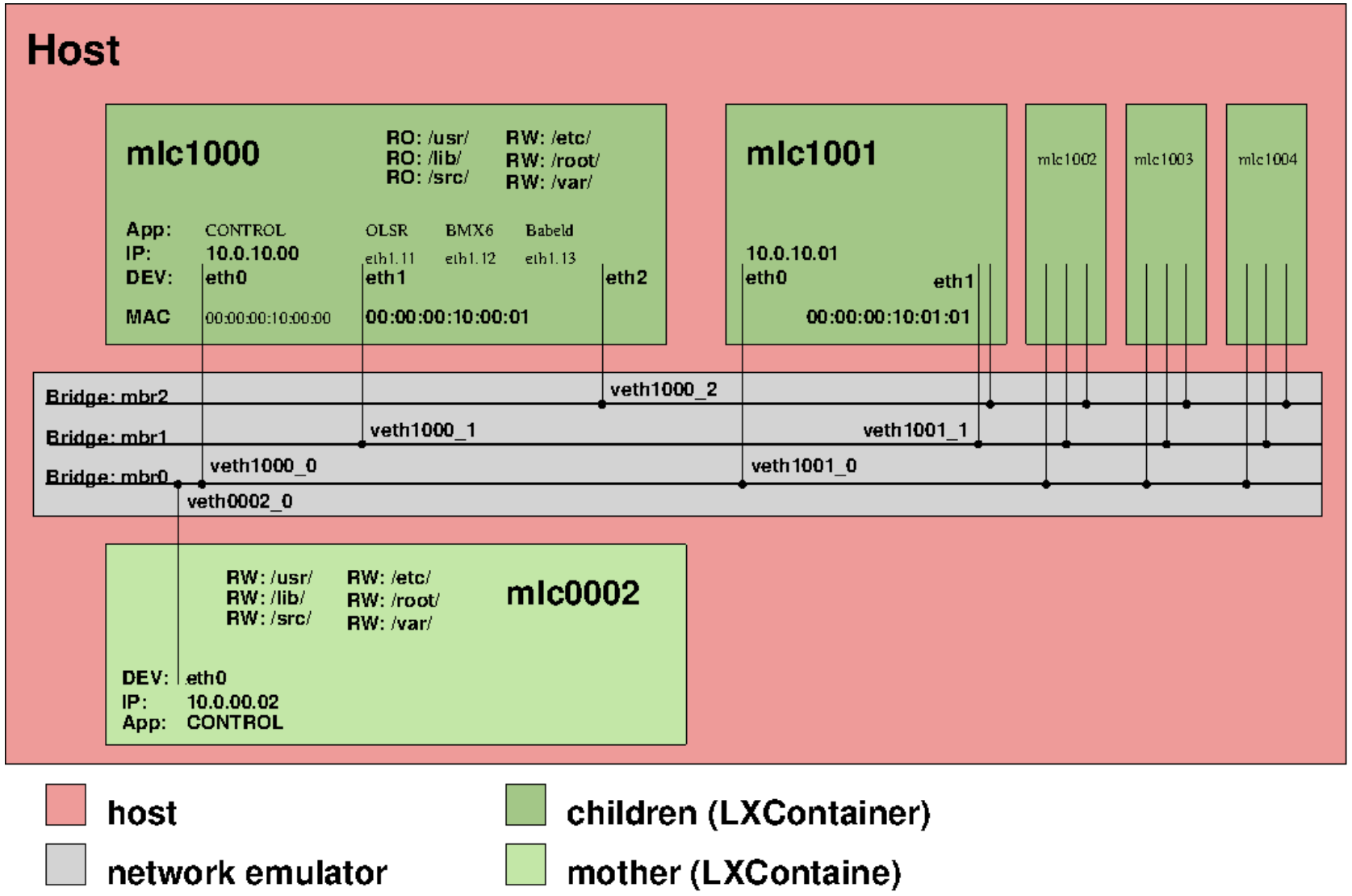
- **Brctl**
- **Ebtables**
- **Tc**

Monitoring

- **Top**
- **Tcpdump**
- **wireshark**



System Overview/Details



Network Emulation 1/2

- Default Policy: Drop ALL
- Specify link by making exception
- Handle link characteristics with tags and tc...

```
root@sid:#  mlc_link_set 1 1000 1 1001 5 7
mlc_link_set eth1 1000 veth1000_1 TQ=5 -- eth1 1001 veth1001_1 TQ=7
mlc_link: src=0:0:0:10:0:1 -> oif=veth1001_1
mlc_link: src=0:0:0:10:1:1 -> oif=veth1000_1

root@sid:# ebtables -Lnv
Bridge chain: INPUT, entries: 0, policy: ACCEPT

Bridge chain: FORWARD, entries: 6, policy: DROP
-s 0:0:0:10:0:1 -d 33:33:0:0:0:0/16 -o veth1001_1 -j mark --mark-set 0x3 --mark-target ACCEPT
-s 0:0:0:10:0:1 -d Broadcast -o veth1001_1 -j mark --mark-set 0x3 --mark-target ACCEPT
-s 0:0:0:10:0:1 -o veth1001_1 -j mark --mark-set 0x4 --mark-target ACCEPT
[...]
Bridge chain: OUTPUT, entries: 0, policy: ACCEPT
```

Network Emulation 2/2

- Default Policy: Drop ALL
- Specify link by making exception
- Handle link characteristics with tags and tc...

```
root@sid:# mlc_qdisc_prepare
root@sid:# tc qdisc
[...]
qdisc prio 1: dev veth1000_1 root refcnt 2 bands 16 priomap  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
qdisc netem 831f: dev veth1000_1 parent 1:3 limit 1000 delay 99us
qdisc netem 8320: dev veth1000_1 parent 1:4 limit 1000 delay 99us
qdisc netem 8321: dev veth1000_1 parent 1:5 limit 1000 delay 200us loss 2%
qdisc netem 8322: dev veth1000_1 parent 1:6 limit 1000 delay 400us
qdisc netem 8323: dev veth1000_1 parent 1:7 limit 1000 delay 299us loss 5%
qdisc netem 8324: dev veth1000_1 parent 1:8 limit 1000 delay 1.6ms
[...]
```

MLC Command Overview (1/4)

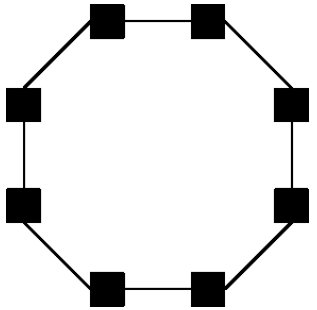
- **. ./mlc-vars.sh; ./mlc-init-host.sh** – init system

- **mlc_loop (handle many nodes/childs)**
  - -c: create node, -b boot, -s stop, -d destroy
    - $ mlc_loop -i 1000 -a 1015 -c
  - -e: execute inside node
    - $ mlc_loop -a 1015 -e “tcpdump -ieth0.12 -n100 > trace&”
  - -u: update node (with new configuration)

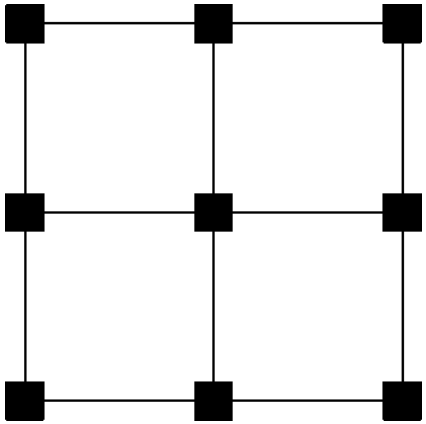
- **mlc_ls** - list running nodes
- **mlc_veth_obtain** - update dev database
- **mlc_qdisc_prepare** - set link categories (tc/qdisc)

MLC Command Overview (2/4)

- **mlc_configure_line**



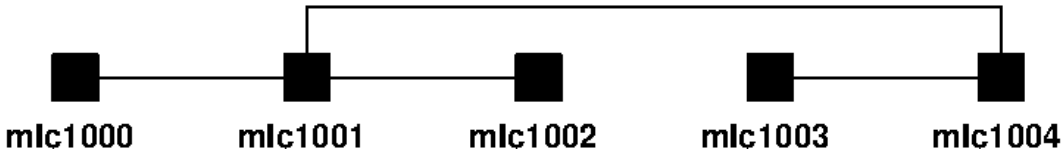
- **mlc_configure_grid**

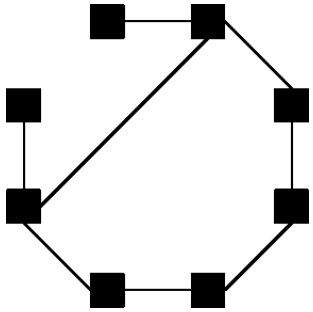


- **mlc_link_set**
- **mlc_mac_set (link with real world)**
- **mlc_net_flush**

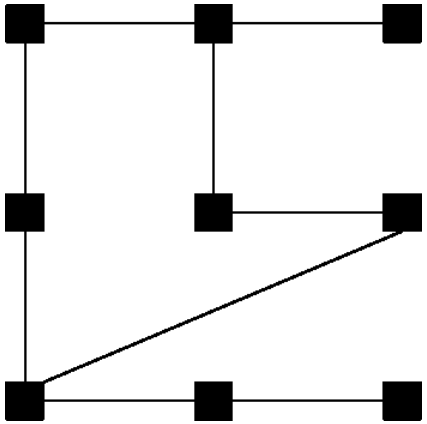
MLC Command Overview (3/4)

- mlc_configure_line





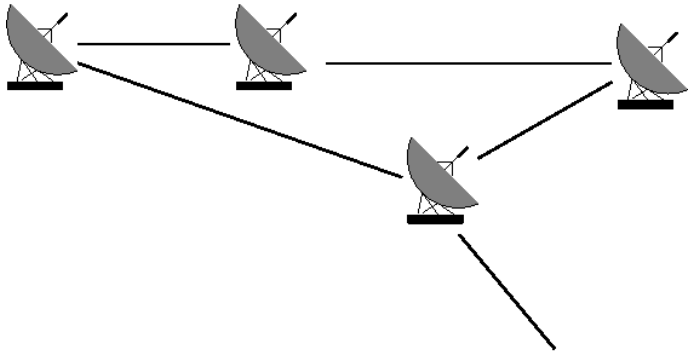
- mlc_configure_grid



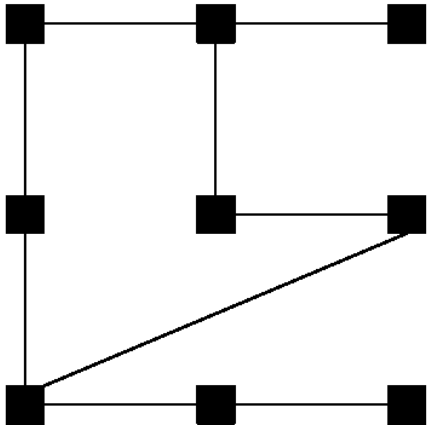
- **mlc_link_set**
- mlc_mac_set (link with real world)
- mlc_net_flush

MLC Command Overview (4/4)

- mlc_configure_line



- mlc_configure_grid



- mlc_link_set
- **mlc_mac_set (link with real world)**
  - **$ mlc_mac_set 1 1002 eth0 00:18:84:1a:07:74 3**
- **mlc_net_flush**

Getting Started (init, simple topology)

```
$ ./mlc-init-host.sh        # setup network emulation, boot mother template
$ ./.mlc-vars.sh            # source the mlc functions
$ mlc_net_flush             # flush (old network topology)
$ ip link                   # show new network environment
$ mlc_loop -a 1015 -c       # create 16 nodes
$ mlc_loop -a 1015 -b       # boot the 16 nodes
$ mlc_qdisc_prepare         # init link characteristics for new nodes
```

```
$ ssh root@10.0.10.00                  # ssh to node 1000 via control network
mlc1000$ ip -4 addr show dev eth1.12
  591: eth1.12@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 ...    inet
10.201.10.0/11 brd 10.223.255.255 scope global eth1.12
mlc1000$ ping -Rn 10.201.10.01         # ping node 1001 via eth1.12 → FAILURE
```

```
$ mlc_link_set 1 1000 1 1001 5 7       # configure link via eth1 mlc1000<->mlc1001
                                       # with slightly asymmetric channel characteristics
```

```
mlc1000$ ping -Rn 10.201.10.01         # ping node 1001 again → SUCCESS
```

Getting Started (ring topology, routing daemon)

```
$ mlc_configure_line                    # get some help on command
$ mlc_configure_line 1 5 5 1015 5 5     # configure ring topology,  symmetric links, 2%brc loss,
                                        # 0.2ms brc delay, 0.4ms unicast delay
```

```
mlc1000$ ping -Rn 10.201.10.15          # ping left one-hop neighbor 1015 → SUCCESS
mlc1000$ ping -Rn 10.201.10.02          # ping 1002 → FAILURE, NO ROUTE!, Keep on trying...
 From 10.201.10.0 icmp_seq=1 Destination Host Unreachable
 [...]
```

```
$ mlc_loop -a 1015 -e “bmx6 configFile=0 ipVersion=4 dev=eth2.12”       # start a routing daemon
```

```
 From 10.201.10.0 icmp_seq=33 Destination Host Unreachable
 64 bytes from 10.201.10.2: icmp_req=34 ttl=63 time=1.85 ms
 RR:          10.201.10.0
 10.201.10.1
 10.201.10.2
 [..]
 64 bytes from 10.201.10.2: icmp_req=35 ttl=63 time=1.82 ms  (same route)
 64 bytes from 10.201.10.2: icmp_req=36 ttl=63 time=1.84 ms(same route)
```

```
$ mlc_link_set 1 1000 1 1001 0 0        # disable link between node 1000<->1001. Break above path!!
```

```
 64 bytes from 10.201.10.2: icmp_req=48 ttl=51 time=63.4 ms
 RR:          10.201.10.0
 10.201.10.15
 10.201.10.14
 [..]
 64 bytes from 10.201.10.2: icmp_req=49 ttl=51 time=12.4 ms  (same route)
 64 bytes from 10.201.10.2: icmp_req=50 ttl=51 time=12.3 ms  (same route)
```

- **12 sec** (icmp_req 48-36=12) **to fix 14-hops route**.

Network and Protocol Characteristics

- Network size, complexity, dynamics
- Protocol overhead, convergence time, robustness
-

Quick Network/Protocol Review

- **Mesh-RouTting vs Internet Routing**
  - NB discovery, link metric valuation, and propagation
  - dynamic network (nodes/links go up and down)
- **OLSR** (Link State)
  - Global view
  - Periodic updates & timeouts
- **BABEL** (Distant Vector)
  - Local view
  - Dynamic updates (on-demand)
- **BMX6** (Distant Vector)
  - Local view
  - Periodic updates & timeouts
  - Statefull compression of RouteUpdates (hashes & IIDs)

Diving Inside (Experiment I):

- Experiment I: Size vs overhead
- Experiment II: Dynamics vs overhead & convergence
-

Further Experiments ?

Questions ?