Course introduction

Network Infrastructures Lab Sessions

SAPIENZA
UNIVERSITÀ DI ROMA

Instructor:
- Pietro Spadaccino

About me

Name: Pietro Spadaccino

E-mail: pietro.spadaccino@uniroma1.it

Office: San Pietro in Vincoli - DIET - Networking Lab

Currently conducting our research on Networking Security on inter-AS protocols (BGP), IoT security focused on LoRaWAN, edge-cloud services optimization.

For personal meetings you can find me in my office, by appointment

Course Goals

- The aim of the course is to give the necessary knowledge to configure and manage LANs & WANs under Unix-like OSes.
- You will be faced with problem sets and assignments that resemble very common situations in a system administrator's everyday life.
- We will use a framework called Kathara, developed by Uniroma3, which allows to emulate a switching environment under Linux
- We will use and configure some of the most widely adopted networking tools, such as OpenSSH, OpenVPN and iptables

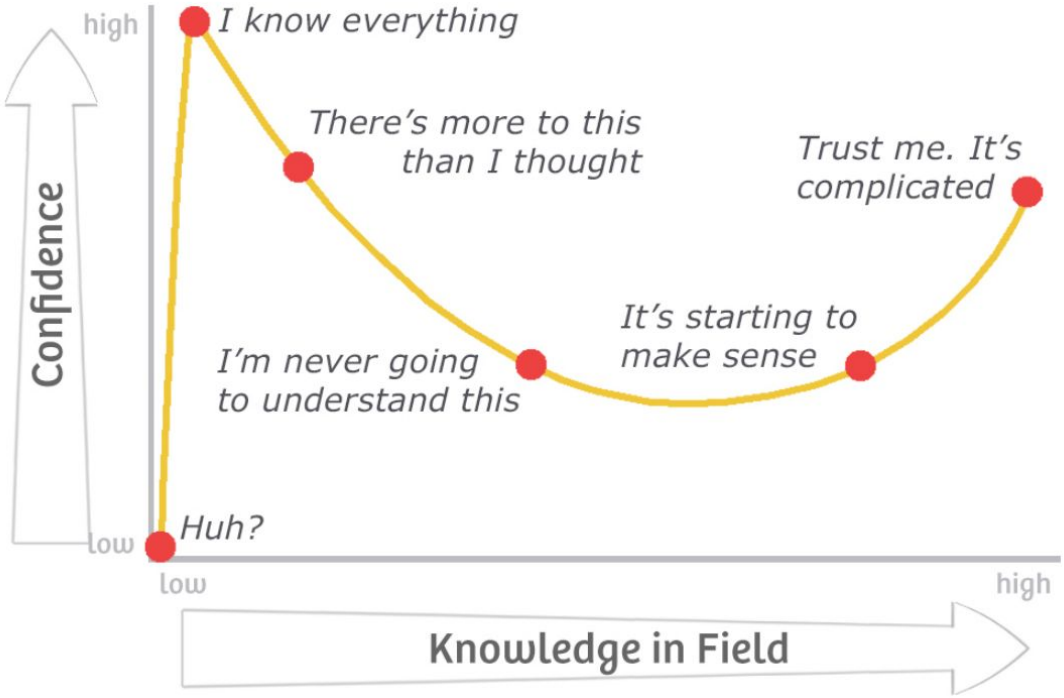
Course program (not necessarily in order)

- Kathara introduction (includes a Docker and container refresh)
- ARP and MAC addressing
- Configuration of static IP addresses & static routes
- DHCP & OSPF
- SSH
- BGP (hopefully)
- VPN
- iptables

Lab philosophy

- The lab sessions are meant to be as practical as possible
  - We will alternate between *me* showing stuff and *you* solving practical assignments on the topics previously shown
- In our opinion, this approach has two major advantages:
  - The course should not be dull for you
  - The practical assignments consist in solving, more or less, the things you'll required to solve at the exam
- Bring your computer in every lecture

Dunning-Kruger effect



Kathara

Kathara is a **network emulation tool**.

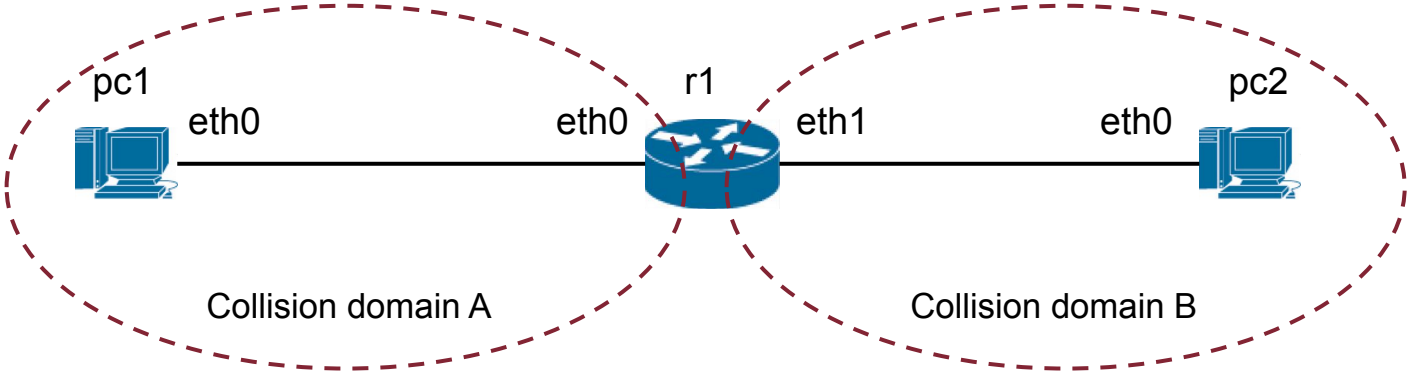
Within the Kathara environment each network node is implemented by a **docker container**, and each interconnection link is emulated by using a virtual network. Each device can be configured to have an arbitrary number of (virtual) ethernet network interfaces.

Each node can be seen as an **independent linux machine**, for you to configure from scratch.

lab_0 demo

- To create a kathara laboratory, in an empty folder create a file `lab.conf`
- Here we specify three things:
  - **node names** (must be unique)
  - **network interfaces** of the node, in the form of eth0, eth1, …
  - **collision domains** of the interfaces

To start a kathara laboratory: `kathara lstart`
To stop a kathara laboratory: `kathara lclean`



lab.conf

In order to start properly the lab, we have to create a ”lab.conf” file inside the lab 0 folder.

This file specifies the connection of the Containers interfaces to collision domains and more

```
pc1[0]=A #set eth0 of pc1 to collision domain A
r1[0]=A  #set eth0 of r1 to collision domain A

r1[1]=B  #set eth1 of r1 to collision domain B
pc2[0]=B #set eth0 of pc2 to collision domain B
```

iproute2

In order to configure interfaces, we will use a commands which are part of the iproute2 suite. These commands replace the old ones (such as ifconfig):

Legacy utility	Obsoleted by	Note
`ifconfig`	`ip addr`, `ip link`, `ip -s`	Address and link configuration
`route`	`ip route`	Routing tables
`arp`	`ip neigh`	Neighbors
`iptunnel`	`ip tunnel`	Tunnels
`nameif`	`ifrename`, `ip link set name`	Rename network interfaces
`ipmaddr`	`ip maddr`	Multicast
`netstat`	`ip -s`, `ss`, `ip route`	Show various networking statistics

The commands on the first column should be considered deprecated and no more maintained, so let’s not use them

ip link

The ip link command let us show and manage (setting up and down) network interfaces:

- `ip link show` `# shows the details of every available iface`
- `ip link set eth0 up` `# brings up eth0 interface`
- `ip link set eth0 down` `# brings down eth0 iface`

Assign IPv4 addresses to interfaces

Now that the ifaces are up, we are can assign IPv4 addresses to them.
The command to assigning IPv4 addresses to ifaces is ip address:

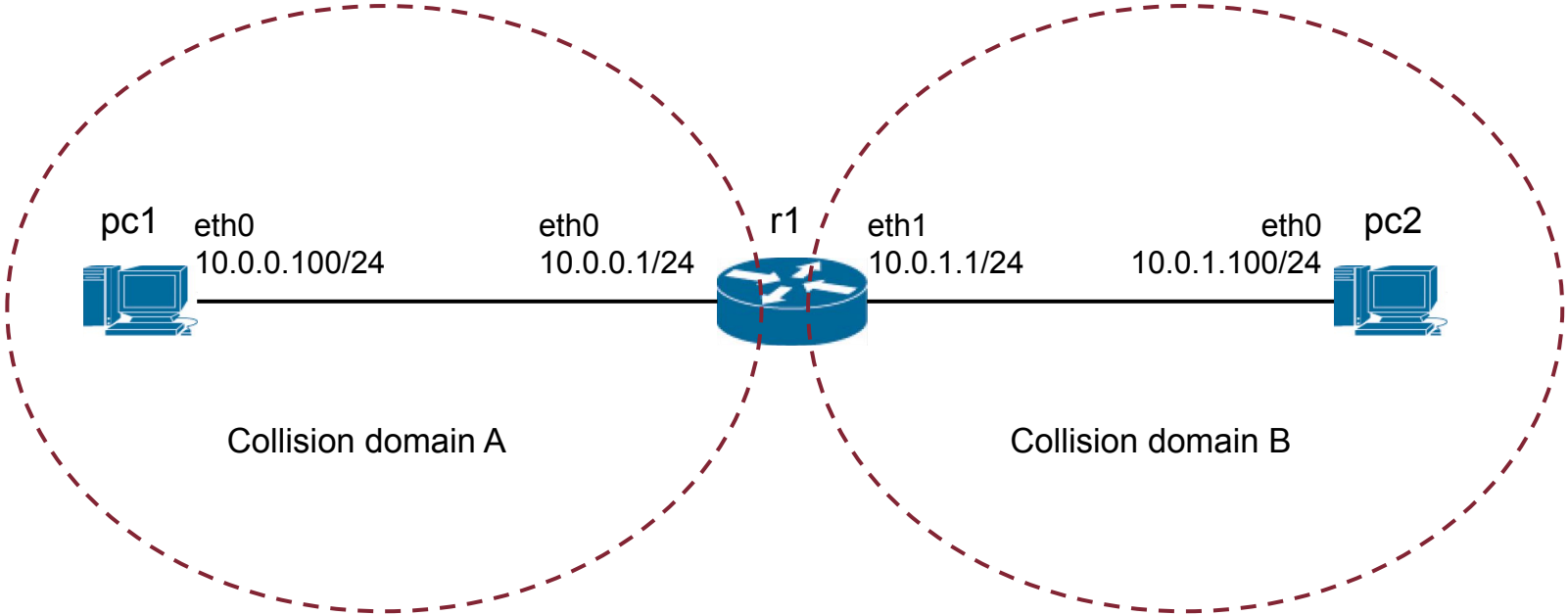
```
ip address show  # show information about addresses
                 # assigned to interfaces

ip address add a.b.c.d/m dev ethX
# assigns a.b.c.d address with
# submask m to device ethX
```

If not otherwise specified, we will assume **distinct ip addresses per network interface**

With these commands, we now assign addresses to every single host as depicted on the topology picture

lab_0


pc1
eth0
10.0.0.100/24
eth0
10.0.0.1/24
r1
eth1
10.0.1.1/24
eth0
10.0.1.100/24
pc2
Collision domain A
Collision domain B

Ping

We can test connection between two hosts using the ping command:

```
ping a.b.c.d      # sends ICMP packets to a.b.c.d
                  # waits for reply
```

It gives us information about Round Trip Time and Time To Live (useful for traceroute) for a connection between 2 hosts.

Static routes

If we try to ping pc1 → pc2, we get ”network unreachable”. **Why**?
Because pc1 does not now how to reach pc2, because he only has the route to his subnet which is 10.0.0.0/24, but pc2 is on 10.0.1.0/24.

In order to let pc1 and pc2 communicate, we have two ways:

- Define the ”default” route (gateway)
- Specify how to reach pc2 subnet from pc1 and viceversa

The difference between the two approaches will be clearer on bigger topology, but basically by defining a gateway we tell to the host ”give all the packets that are not for your subnet to the gateway, he knows how to route them”.

In our case the gateway is r1. With the second approach, we tell to the host ”give all the packets that are for a specific subnet to a specific host, he knows how to route them”.

ip route

In order to manage routes, we use the *ip route* command:

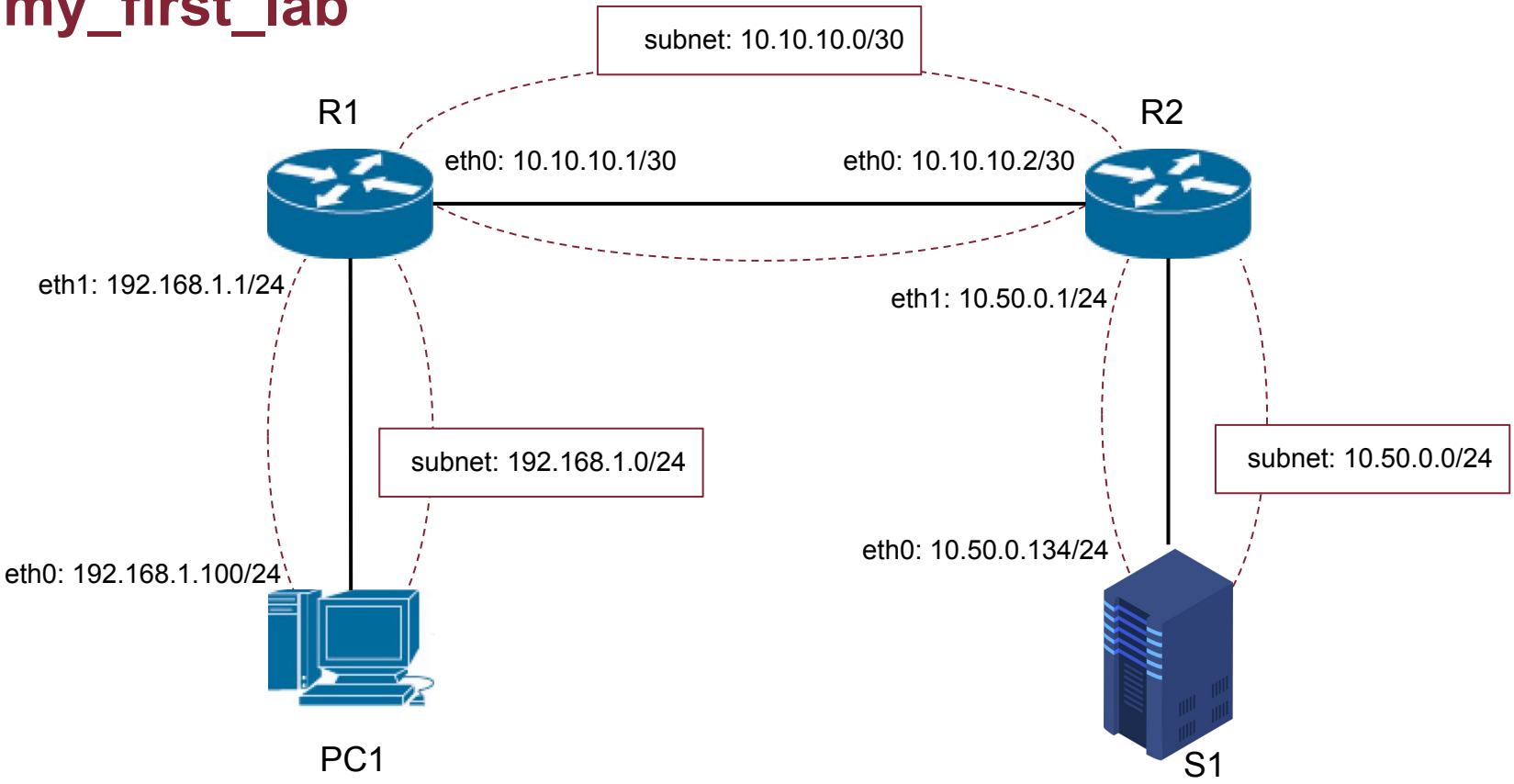
```
ip route show    # show the available routes on the host

ip route add a.b.c.d/m via next_hop_ip
# adds the route to subnet a.b.c.d/m via the next_hop_ip

ip route add default via next_hop_ip
ip route add 0.0.0.0/0 via next_hop_ip
# adds the default gateway via the next_hop_ip
```

There is a special subnet, called ”default”, wich is 0.0.0.0/0, that enables you to define the gateway.

my_first_lab


subnet: 10.10.10.0/30
R1
R2
eth0: 10.10.10.1/30
eth0: 10.10.10.2/30
eth1: 192.168.1.1/24
eth1: 10.50.0.1/24
subnet: 192.168.1.0/24
subnet: 10.50.0.0/24
eth0: 192.168.1.100/24
eth0: 10.50.0.134/24
PC1
S1

my_first_lab

Goal: Make every node able to reach (ping) all other nodes

Proposed exercise:

1. Create and launch a kathara lab having the topology (collision domains) as in the previous slide
2. Assign static ip addresses as shown via `ip address` commands
3. Assign necessary routes via `ip route` commands
4. Ping everybody

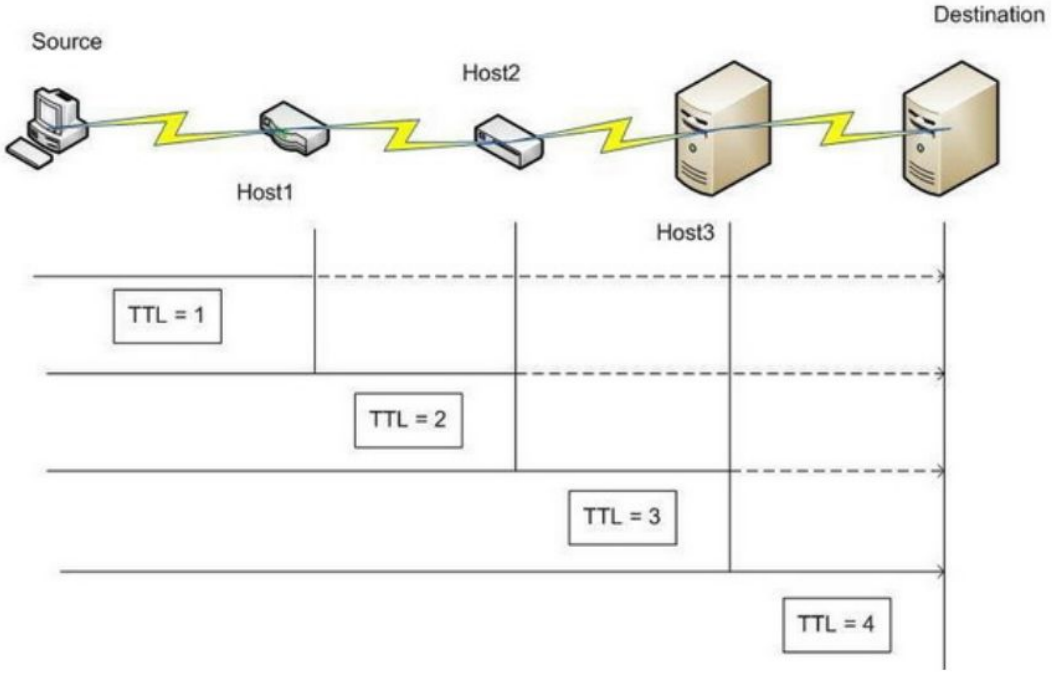
traceroute

Another useful way of testing connection between two hosts is to traceroute the connection via the traceroute command:

```
traceroute a.b.c.d    # finds the number and addresses
                      # of hops to reach a.b.c.d
```

- This command exploit the fact that every packet has a TTL field which is X when the packet leaves the interface.
- Every hop decrease by one the TTL field. When the TTL reaches 0, the host that expired the TTL sends back to the sender an ICMP error packet.
- The packet contains the IP address of the host who expired the TTL. By sending packets with TTL starting from 1, tracert can discover the path taken from the packets to reach a particular destination.

traceroute



netcat

Netcat allows us to send data over TCP and UDP protocols. In order to use that, we must specify the protocol we ant to use, the port we want to use and if we want to send or ”listen” on that particular port:

```
nc ip_addr port_num   # connects via TCP to ip_addr
                      # trough port_num via TCP

nc -u ip_addr port_num    # Send UDP datagrams
                          # to ip_addr trough port_num

nc -l -p port_num         # listen for TCP connections on port_num

nc -u -l -p port_num      # listen for UDP datagrams on port_num
```

tcpdump

Linux provides us tools to generate, capture and inspect packets. We can create TCP/UDP packets via the nc (netcat) command. This packets generated can be ”captured” via the tcpdump command and stored to a ”pcap” (packet capture) file. This file can be later open to be graphically inspected with the wireshark tool.

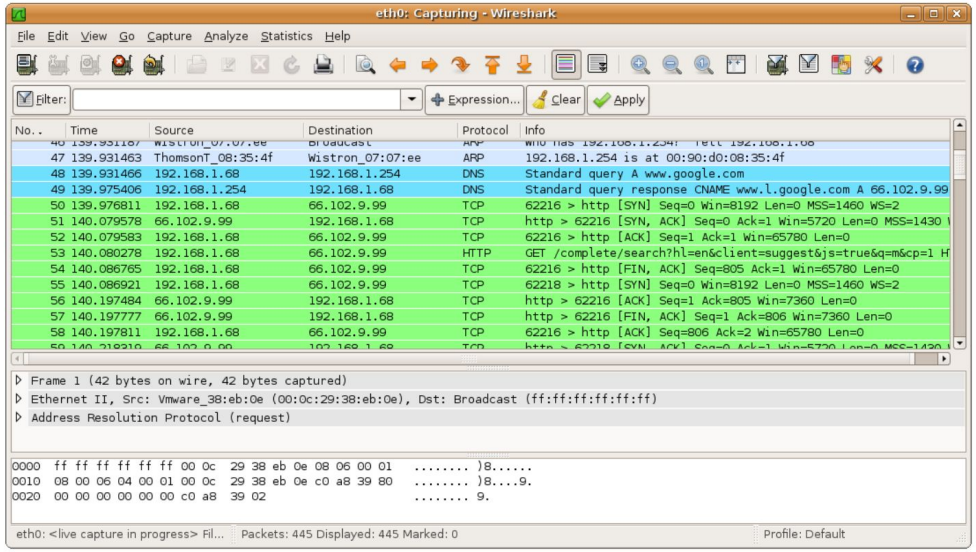
tcpdump allows us to capture packets on a physical interface.

```
tcpdump -i iface_name -w filename.pcap
# capture packets on iface_name and save captured packets on filename.pcap
```

Has a lot more configuration parameters → check on the manual.

Wireshark

After capturing packets with tcpdump, we can open the pcap file with wireshark to view the packets and their contents.



my_first_sniffer

Goal: Familiarize with tcpdump, wireshark, netcat (a.k.a. your best friends when debugging)

Proposed exercise:

1. Launch the lab you have created previously
2. Launch a netcat listener in TCP mode on port 8080 on S1 and connect to it from PC1. Try to send some data in both directions
    a. Launch tcpdump on R1 to sniff packets of the connection. You can dump the capture in the file `/shared/my_capture.pcap` and open it with wireshark on your computer
3. Do again the previous step but using UDP mode instead of TCP. Which are the differences in the exchanged packets during:
    a. Connection establishment?
    b. Data exchange?
    c. Connection closing?
4. Kill the netcat server and try to connect to it anyways. Capture the traffic with wireshark, which error packet do you observe?