ROUTING PROTOCOL BASICS

Let's spice things up….

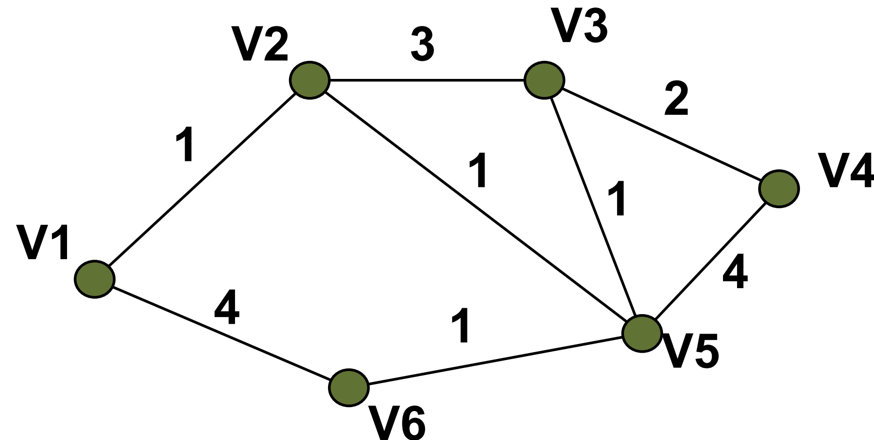
pc2
LAN D
10.3.0.0/16
eth2:
10.3.0.1
router2
eth1:
10.0.0.2
eth0:
10.2.0.1
LAN A
10.0.0.0/16
eth0:
10.7.0.2
router7
eth1:
10.8.0.1
LAN I
10.8.0.0/16
pc1
LAN H
10.7.0.0/16
eth2:
10.7.0.1
eth0:
10.0.0.1
router1
eth1:
10.1.0.1
LAN B
10.1.0.0/16
eth0:
10.1.0.2
router4
eth1:
10.5.0.1
LAN C
10.2.0.0/16
eth1:
10.2.0.2
router3
eth0:
10.4.0.2
eth0
10.4.0.1
eth2:
10.9.0.2
LAN E
10.4.0.0/16
pc3
LAN L
10.9.0.0/16
eth1:
10.9.0.1
router6
eth0:
10.6.0.2
LAN G
10.6.0.0/16
eth1:
10.6.0.1
router5
eth0:
10.5.0.2
LAN F
10.5.0.0/16
eth0:
10.5.0.3
router8
eth1:
10.10.0.1
LAN M
10.10.0.0/16
pc4

A much more complex topology…

Routing Protocols

- Larger network topologies can't rely on static routing configuration
  - Big topologies
  - What's the best path?
  - Reconfiguration?
- Routing protocols define the procedures and the messages to exchange routing information between the routers and automatically construct the routing tables
- Routing protocols make use of graph search algorithms to compute "the best paths" between all possible destinations
  - Dijkstra, Bellman-Ford
- Network topologies can be modeled as weighted oriented graphs (router, networks → nodes; links → edges; weights → link metrics)
- Construction of the routing tables is equivalent to find all the "best" paths between all nodes of the graph
  - "Best" is not always equal to "shortest"
- Let's take a look at a simple example of Dijkstra

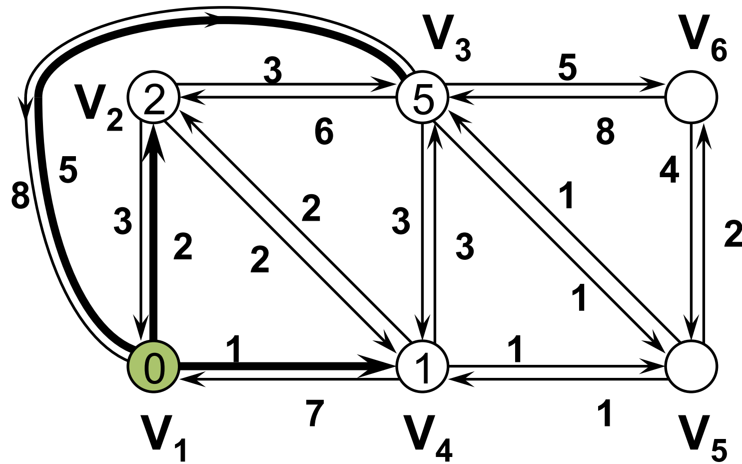
Dijkstra algorithm at a glance



- The algorithm is iterative:
  - At the Kth step we have K nodes that can be reached from the source node S with the shortest path
  - Such K nodes are in the set T={V1, V2, …, Vm}
  - At step $(K+1)^{th}$ add the node with the path with minimum weight from S that transit in the node in T
  - The algorithm ends when all nodes have been explored
- The weight of a path is the sum of the weights (costs) of all the edge forming the path

Dijkstra: a simple example

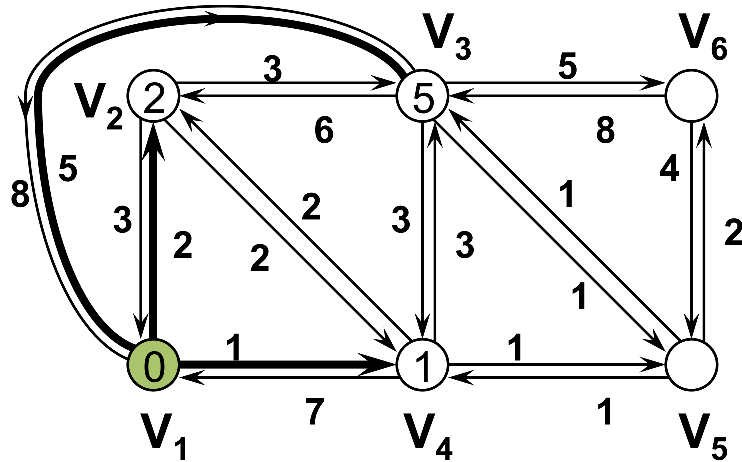
Step 1



$T_1 = \{1\}$

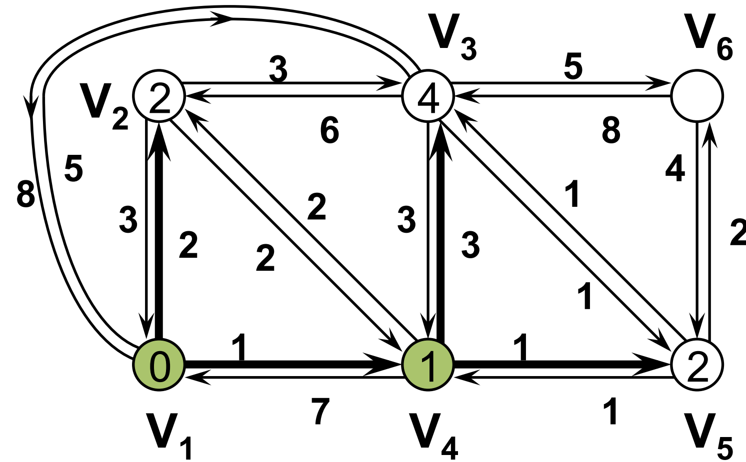
Dijkstra: a simple example

Step 1



$T_1 = \{1\}$

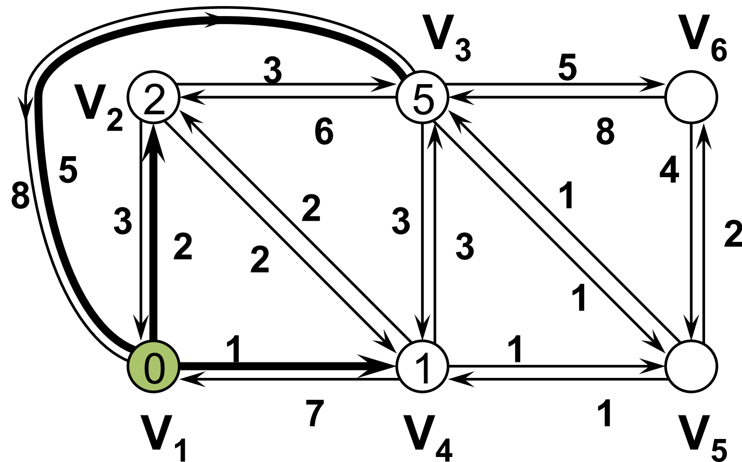
Step 2



$T_2 = \{1,4\}$

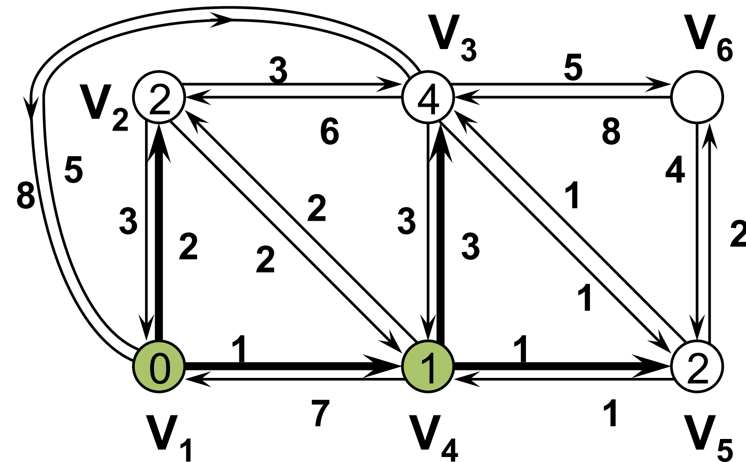
Dijkstra: a simple example

Step 1



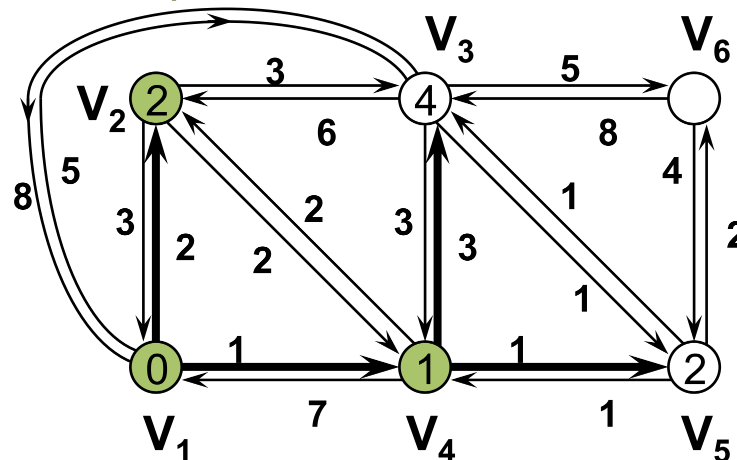
$T_1 = \{1\}$

Step 2



$T_2 = \{1,4\}$

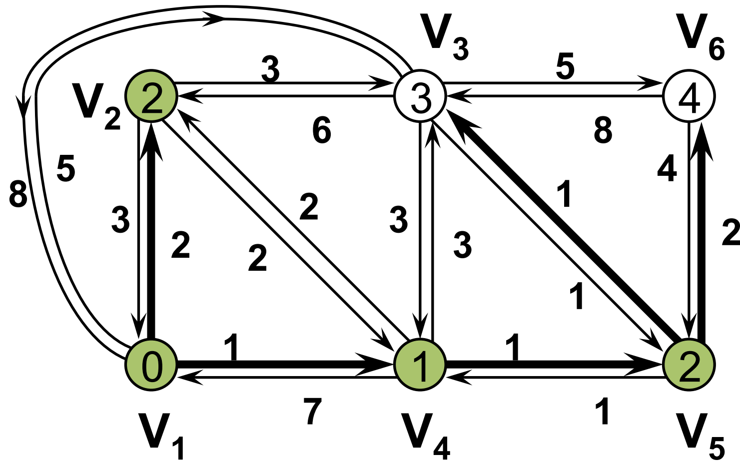
Step 3



$T_3 = \{1,2,4\}$

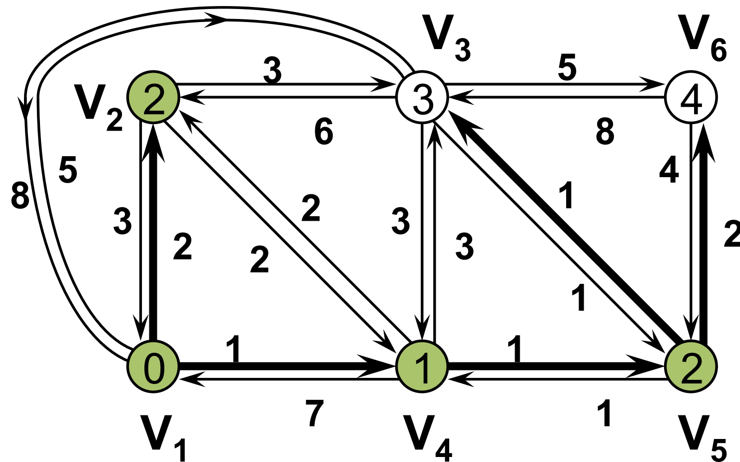
Dijkstra: a simple example

Step 4

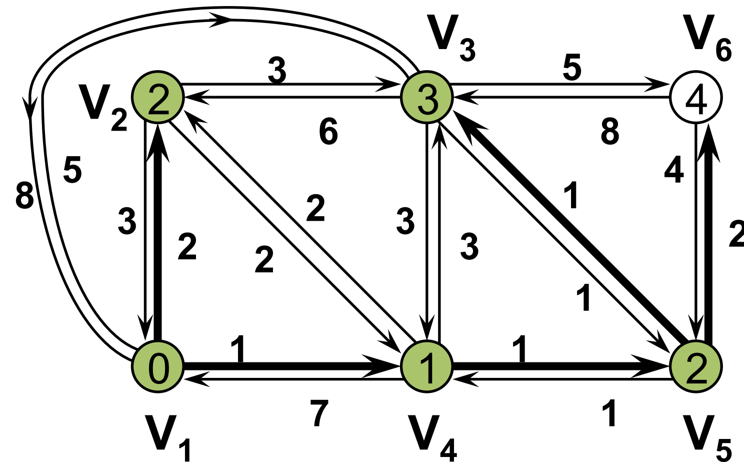


$T_4 = \{1,2,4,5\}$

Dijkstra: a simple example



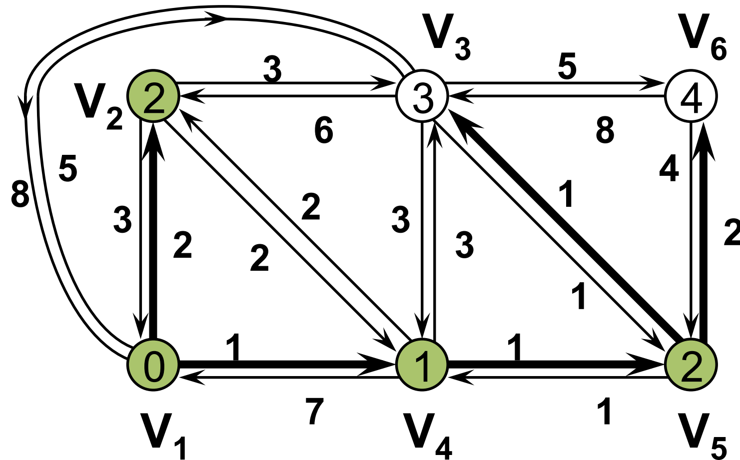
$T_4 = \{1,2,4,5\}$



$T_5 = \{1,2,3,4,5\}$

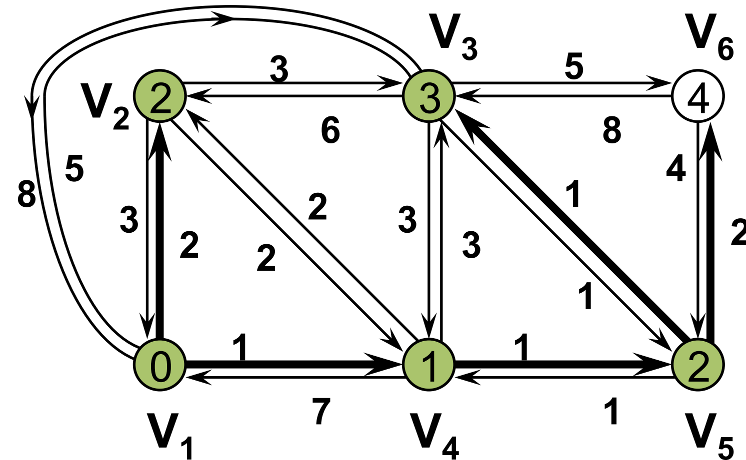
Dijkstra: a simple example

Step 4



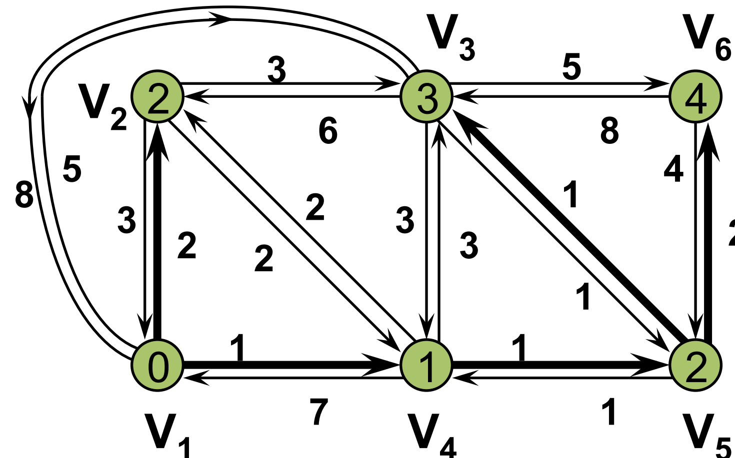
$T_4 = \{1,2,4,5\}$

Step 5



$T_5 = \{1,2,3,4,5\}$

Step 6



$T_6 = \{1,2,3,4,5,6\}$

How can Dijkstra be used in a routing protocol?

- There are different routing protocols that use Dijkstra as search graph algorithm to construct the routing tables
- All the routers exchange information about all the respective neighbor routers and the relative links to them (neighbor routers = routers at 1 hop)
- After a “convergence time” all the routers know the complete topology
- Each router constructs the graph representing the network topology and then run Dijkstra with themselves as source node
- Such a routing protocol in called: Link State protocol
  – Because every node in the network has to know the entire topology (i.e.: the state of each link in the network topology)

OSPF

- Open Shortest Path First is a link state routing protocol that uses Dijkstra to construct the routing tables
- Each router advertises its identity to the neighbors
- Each router transmits a Link State Packet (LSP) containing the list of the links and their costs
- A LSP is transmitted to all the other routers (flooding mechanism)
- All the routers can locally build the entire network topology by gathering all the received LSPs

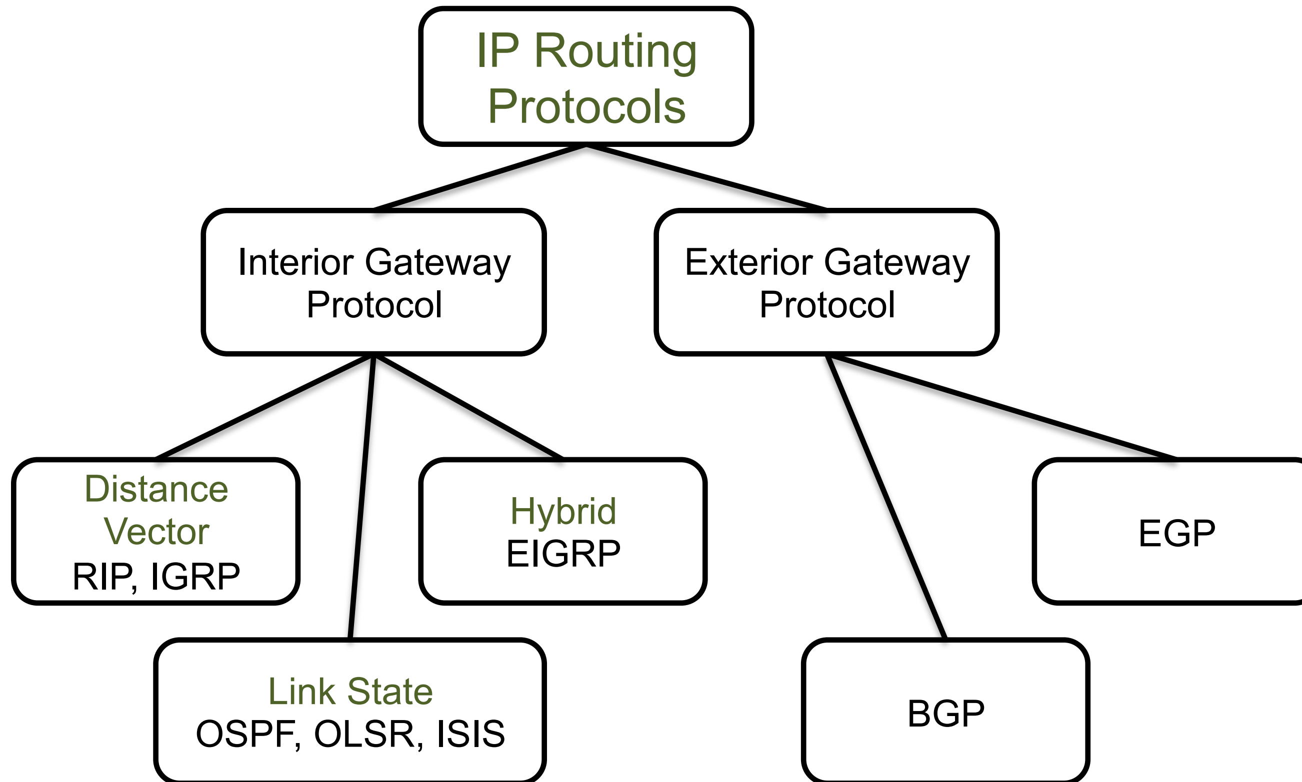
Distance Vector protocols

- A distance vector routing protocol is a protocol in which all the routers knows only the routing tables of their neighbors (the vectors of the distance to all possbile destinations)
- The next hop to the other destinations is picked up by choose the minimizing

$$\text{Cost}(i, d) = (L_i + C_{d,i})$$

- Where:
  - i is the next hop
  - L_i is the cost of the link to i
  - d the destination
  - $C_{d,i}$ is the cost of the route to d in the routing table of i

Routing Protocol classification



NOTE: Too many things about routing protocols have been ignored. We don't want to understand the routing protocols in details, we just wanted an introduction to understand what we are going to see in the following slides…

QUAGGA

- Quagga is an opensource routing software suite, providing implementations of OSPFv2, OSPFv3, RIP v1 and v2, RIPng and BGP-4 for Unix platforms, particularly FreeBSD, Linux, Solaris and NetBSD
- Quagga is a fork of GNU Zebra for which development was abandoned in 2003
- The Quagga architecture consists of a core daemon (zebra) that presents the Zserv API over a Unix or TCP stream to Quagga daemons:
  - ospfd for OSPFv2
  - ripd for RIPv1 and RIPv2
  - ospf6d for OSPFv3 (IPv6)
  - ripngd for RIPng (IPv6)
  - bgpd for BGPv4+
  - isisd for IS-IS
- The zebra daemon is responsible for providing an interface to the kernel IP and routing subsystems
- The per-protocol daemon are responsible for performing all protocol-specific operations as described in the relevant RFCs
- Each daemon has its own configuration file and terminal interface which can be accessed by telnet
- The `vtysh` tool is provided to configure the Quagga router from the localhost, in a unique interface

QUAGGA configuration

- Activate the daemon by putting “yes” for the protocol you need in the file `/etc/quagga/daemons`
  - For example: `zebra=yes; ospfd=yes`
  - Each daemon has its own configuration file
  - `bgpd.conf`, `ospfd.conf`, `ripd.conf, etc..`

- By default, the quagga daemons are listening only to the loopback interface 127.0.0.1. If you want to telnet a quagga daemon remotely you can, in the `/etc/quagga/debian.conf` file
  - Either indicate one or several IP addresses or remove the -A option meaning that you can telnet a daemon on any of its IP addresses
  - Examples:

```
ospfd_options=“ --daemon -A 127.0.0.1 192.168.1.104”
zebra_options=“ --daemon”
```

- To enable vtysh put in `/etc/quagga/debian.conf`
  - `vtysh_enable=yes`

QUAGGA daemons shells

- you can access the daemons by telnetting their port number because each daemon has its own configuration file and terminal interface
- The daemon name/port binding is stored in /etc/services, so you can also use the name
  - `telnet localhost ripd`
  - `telnet localhost 2008`

Anyway, as it's not very practical to configure your router by telnetting its daemons separately, vtysh has been created to configure everything in one single interface

To connect just type:

```
vtysh
```

zebra	2601
ripd	2602
ripng	2603
ospfd	2604
bgpd	2605
ospf6d	2606
isisd	2008

QUAGGA - configuration files

- Official docs
  - http://www.nongnu.org/quagga/docs
- From vtysh you can configure everything (interfaces, ip configuration, routing protocols parameters, etc..) and dump the status of your router (show routes, show interfaces, etc..)
- We are not gonna learn how to use the vtysh console, but we'll try to configure the quagga daemons via configuration files
- BTW, the configuration lines are exactly like the commands you type from the shells

Simple configuration examples

```
! -*- zebra -*-

hostname router2
password zebra
enable password zebra

interface eth0
ip address 10.2.0.1/16
link-detect
bandwidth 100000

interface eth1
ip address 10.0.0.2/16
link-detect
bandwidth 100000

interface eth2
ip address 10.3.0.1/16
link-detect
bandwidth 5000
~
```

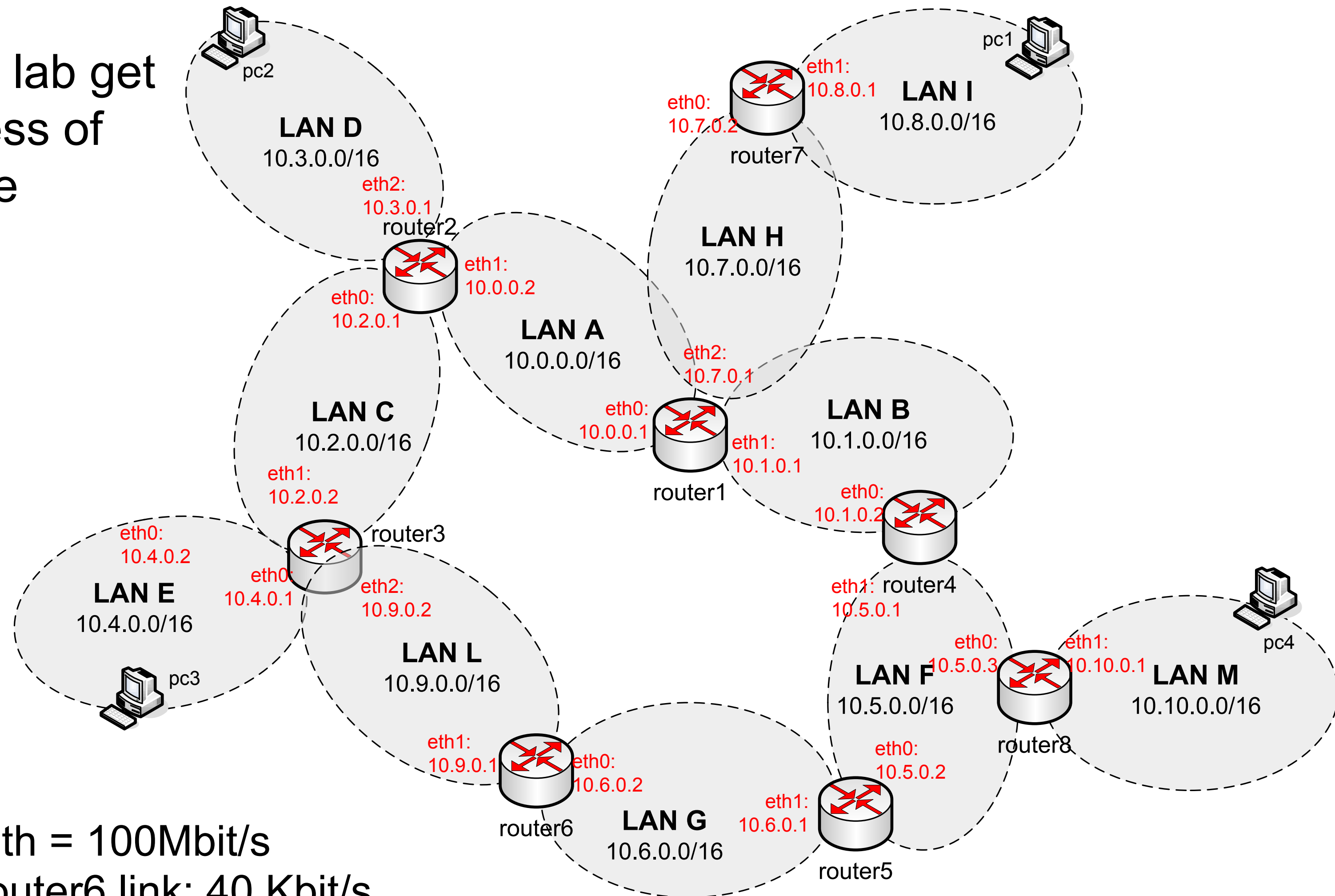
zebra.conf

```
hostname router2
password zebra
log file /var/log/quagga/ospfd.log

interface eth0
ospf hello-interval 2

interface eth1
ospf hello-interval 2

interface eth2
ospf hello-interval 2

router ospf
network 10.0.0.0/16 area 0.0.0.0
network 10.2.0.0/16 area 0.0.0.0
network 10.3.0.0/16 area 0.0.0.0
```

ospfd.conf

Lab3-quagga

pcs{1-4} in the lab get the .100 address of their respective subnets



Default bandwidth = 100Mbit/s
Router3 ←→ Router6 link: 40 Kbit/s

Lab3-quagga

router3

```
<none>

###################################################

--- Netkit phase 2 initialization terminated ---


router3 login: root (automatic login)
router3:~# ip r
10.2.0.0/16 dev eth1  proto kernel  scope link  src 10.2.0.2
10.4.0.0/16 dev eth0  proto kernel  scope link  src 10.4.0.1
10.9.0.0/16 dev eth2  proto kernel  scope link  src 10.9.0.2
router3:~# ip r
10.2.0.0/16 dev eth1  proto kernel  scope link  src 10.2.0.2
10.3.0.0/16 via 10.2.0.1 dev eth1  proto zebra  metric 20
10.0.0.0/16 via 10.2.0.1 dev eth1  proto zebra  metric 20
10.1.0.0/16 via 10.2.0.1 dev eth1  proto zebra  metric 30
10.6.0.0/16 via 10.2.0.1 dev eth1  proto zebra  metric 50
10.7.0.0/16 via 10.2.0.1 dev eth1  proto zebra  metric 30
10.4.0.0/16 dev eth0  proto kernel  scope link  src 10.4.0.1
10.5.0.0/16 via 10.2.0.1 dev eth1  proto zebra  metric 40
10.8.0.0/16 via 10.2.0.1 dev eth1  proto zebra  metric 40
10.9.0.0/16 dev eth2  proto kernel  scope link  src 10.9.0.2
router3:~#
```

Intuitively, for router3 the next hop to 10.5.0.0/16 is router2 (3 hops) and not router6 (2 hops)….

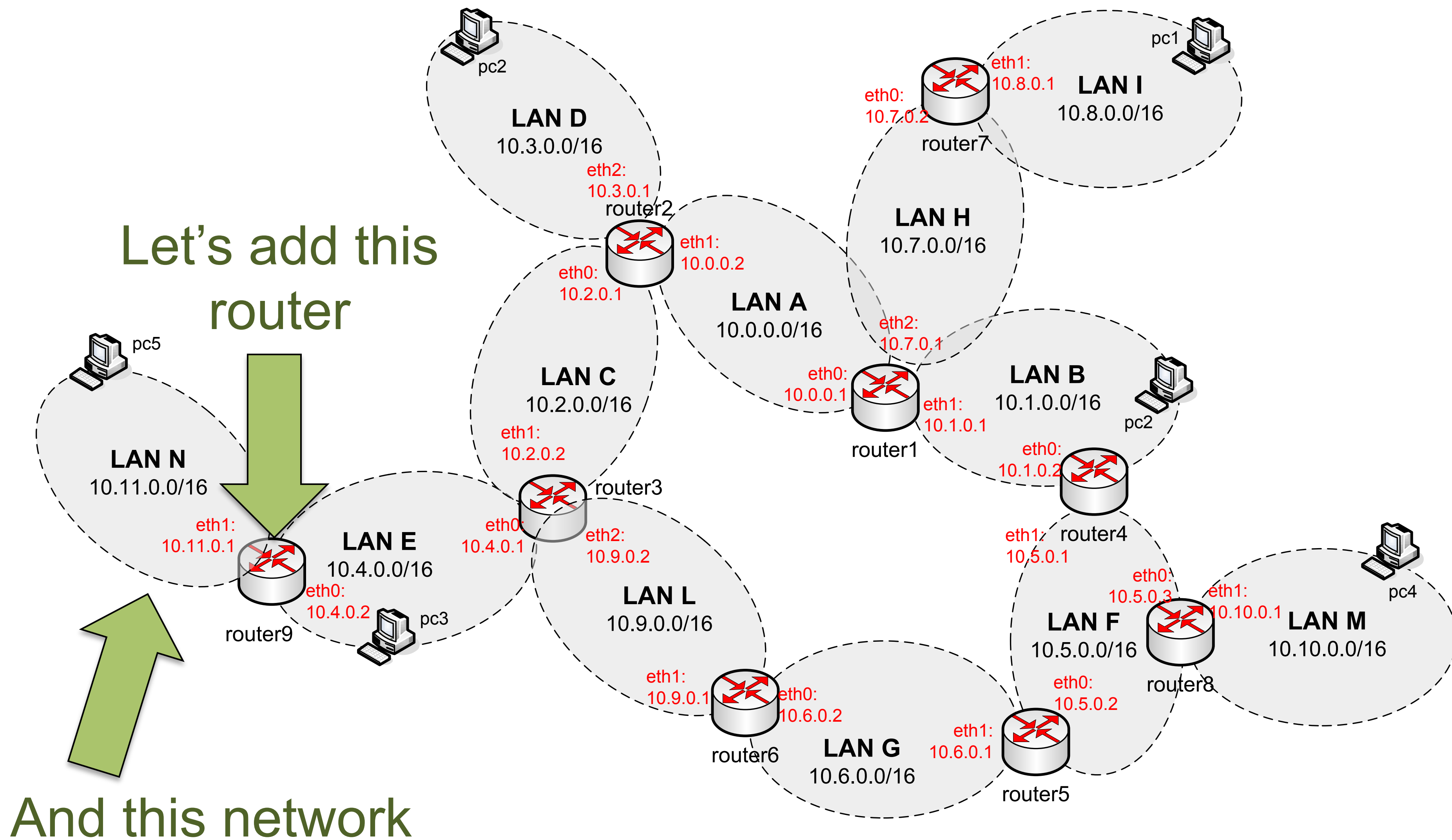
Lab3-quagga

pc3

```
Lab directory (host): /home/marlon/Labs/Lab3-quagga
Version: <none>
Author:  <none>
Email:   <none>
Web:     <none>
Description:
<none>

##################################################

--- Netkit phase 2 initialization terminated ---


pc3 login: root (automatic login)
pc3:~# traceroute 10.10.0.100
traceroute to 10.10.0.100 (10.10.0.100), 64 hops max, 40 byte packets
 1  10.4.0.1 (10.4.0.1)  5 ms  0 ms  0 ms
 2  10.2.0.1 (10.2.0.1)  10 ms  0 ms  0 ms
 3  10.0.0.1 (10.0.0.1)  2 ms  0 ms  0 ms
 4  10.1.0.2 (10.1.0.2)  0 ms  0 ms  0 ms
 5  10.5.0.3 (10.5.0.3)  0 ms  0 ms  0 ms
 6  10.10.0.100 (10.10.0.100)  37 ms  0 ms  0 ms
pc3:~#
```

…and, clearly, pc3 → pc4 path is: router3 → router2 → router1 → router4 → router8

Exercise



Exercise (cont.d)

- Let’s do it together starting from Lab3-quagga
- Add router9
- Enable ospfd daemon
- Change file permission as needed
- Configure zebra
- Configure ospfd
- Start quagga