DHCP & OSPF

Network Infrastructures Lab Sessions

SAPIENZA
UNIVERSITÀ DI ROMA

Instructors:
- Pietro Spadaccino

Today

- Something about automatic laboratory configuration in Kathará
- DHCP
  - Dynamic IP assignment
- Recap exercise on what we have seen so far
- OSPF
  - Dynamic routing

Startup files

When you launch a kathará lab it starts from scratch. It doesn't "save" command you have given before.

To avoid entering commands every time you start a laboratory, you can create "**`x.startup`**" files in the laboratory folder of your host computer, where **`x`** is the name of a device.

- The commands contained in **`x.startup`** file (one per line) will be executed when **`x`** starts.

Files in kathara containers

In the lab folder, you can create a directory named as a device. The files inside that directory are mapped into that device at startup.

For example: you have a kathara device named `pc1`. On your host computer, you can create a directory named pc1 with a file `foo.txt` inside. When pc1 starts it will see the file in `/foo.txt`.

You can also create subdirectories: on your host computer you can create a file in `pc1/etc/foo.txt` and, when pc1 boots up, it will see a file `/etc/foo.txt`

Interface configuration in Linux environments

In real linux environments, the ifaces configuration could be carried out by the "`/etc/network/interfaces`" configuration file. In this file, you can specify the way in which ifaces are brought up and assign ip addresses, netmasks and gateway (and a lot more!).

The configuration below enables interface eth0 at boot, assigns to it the address 192.168.1.100, with its netmask, and sets its default gateway to 192.168.1.1:

```
auto eth0
iface eth0 inet static
        address 192.168.1.100
        netmask 255.255.255.0
        gateway 192.168.1.1
```

We need to create the /etc/network/interfaces file inside every host directory, in order to be mounted at boot up of the VMs

Caveat

In linux, network, as many other things, are managed by a ”daemon”. A daemon is a background process which is configured via configuration files like ”/etc/network/interfaces”.

Since the boot process of a Kathara VM first launches the networking daemon and after mounts the files and folders inside the VM folders in the lab, we **NEED** to restart the daemon via:

```
/etc/init.d/networking restart
```

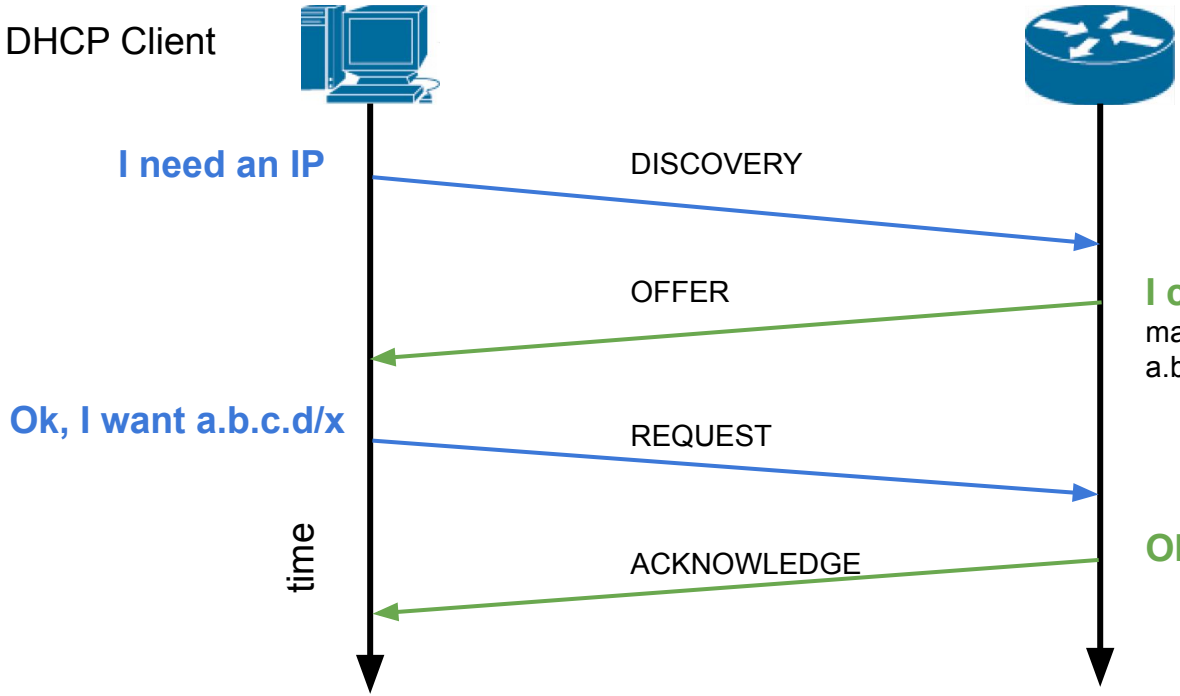
In order to do that automatically, put the command **inside the .startup script of every host.**

DHCP

DHCP Overview

- DHCP (Dynamic Host Configuration Protocol) is a protocol used on LANs for automatically assigning IP addresses and other informations, including default gateways and DNS
- DHCP follows a client-server architecture: the client asks for an IP and the DHCP server replies
- DHCP is everywhere, on your home Wi-Fi, in Sapienza Wi-Fi etc.

DHCP Protocol



- Works on UDP, port 67
- Uses broadcast IP

An example of the protocol is contained into the capture dhcp.pcap

DHCP in Linux

In Linux DHCP is implemented with *isc-dhcp*, which is the most used open-source DHCP implementation. It provides:

- DHCP Client (dhclient)
- DHCP Server (dhcp3-server)
- DHCP Relay (dhcrelay)

DHCP - Client configuration

To configure a network node to use DHCP as a client, just replace the static configuration in /etc/network/interfaces with:

```
iface eth0 inet dhcp
```

DHCP - Server configuration

To configure a node as a DHCP server, create a file `dhcpd.conf` in `/etc/dhcp/`:

```
default-lease-time 3600;
option domain-name-servers 8.8.8.8;

subnet 192.168.1.0 netmask 255.255.255.0 {

    range 192.168.1.50 192.168.1.100;
    option routers 192.168.1.1;
}


subnet 10.0.1.0 netmask 255.255.255.0 {

    range 10.0.1.150 10.0.1.200;
    option routers 10.0.1.1;
}
```

This configuration on the left enables the DHCP server on two subnets.

It assigns the IP addresses in the specified range to clients.

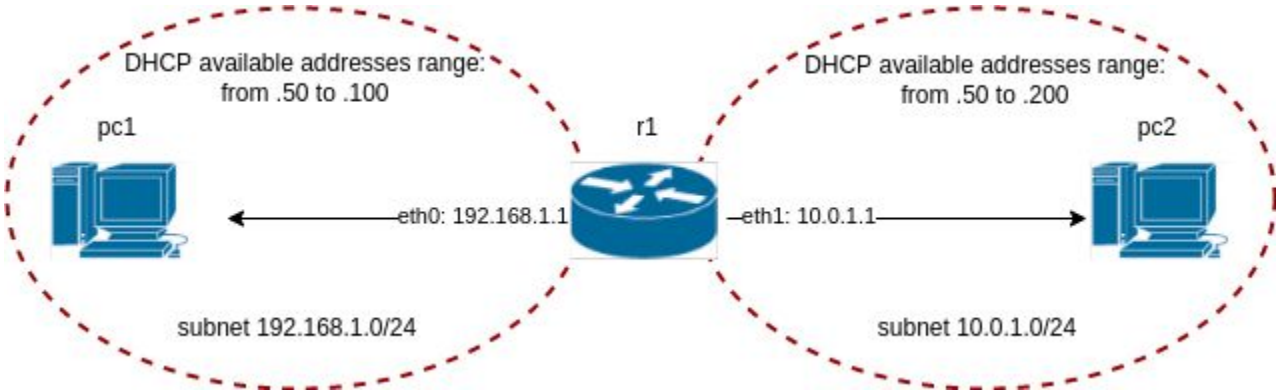
It also sets the default gateway for each subnet

Other options are possible, e.g. for setting DNS (`option domain-name-servers`), reserve IP for certain hosts etc.

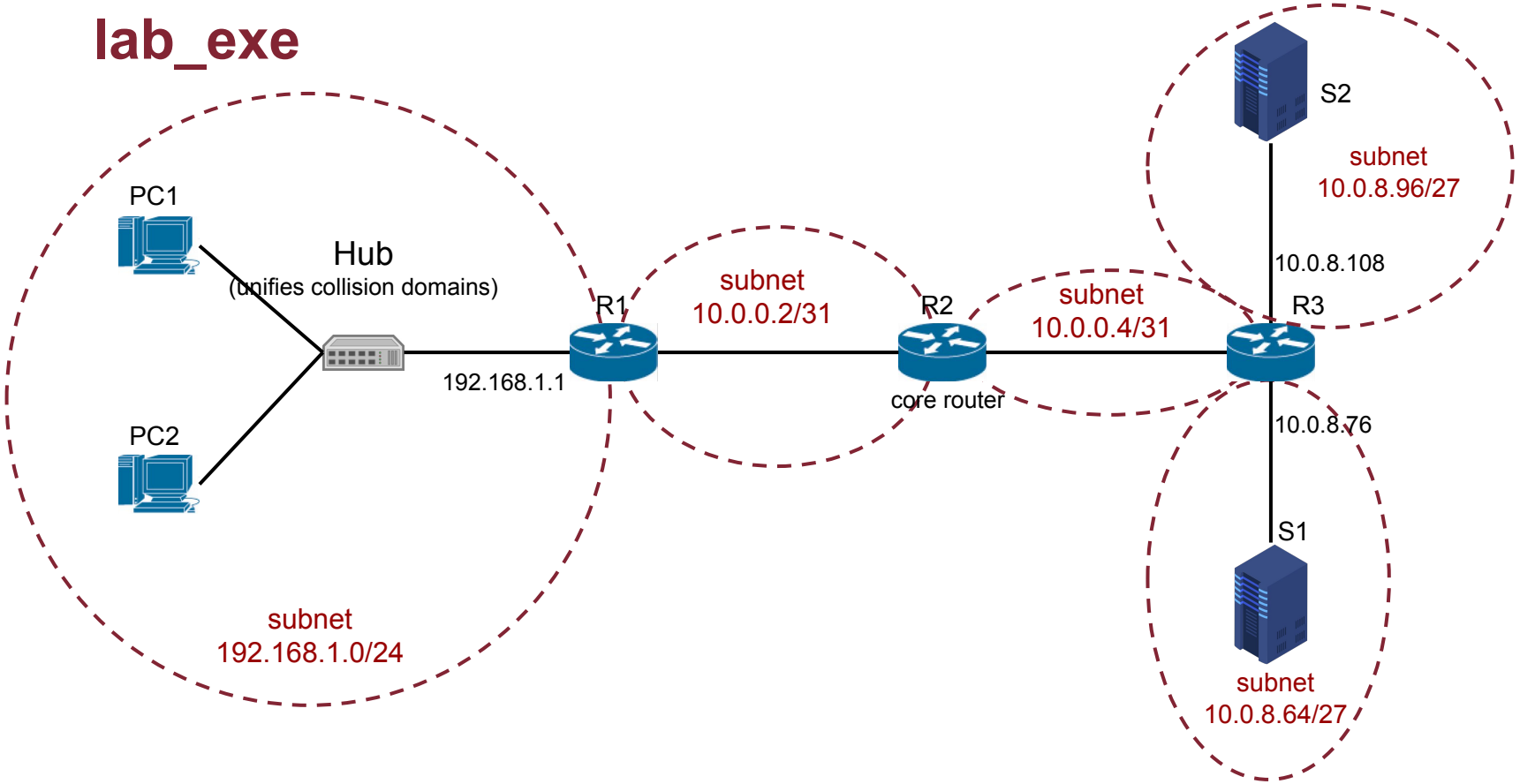
To start the DHCP server on boot add this to the startup script:

```
/etc/init.d/isc-dhcp-server start
```

lab_dhcp


DHCP available addresses range:
from .50 to .100
pc1
r1
DHCP available addresses range:
from .50 to .200
pc2
eth0: 192.168.1.1
eth1: 10.0.1.1
subnet 192.168.1.0/24
subnet 10.0.1.0/24

lab_exe


PC1
Hub
(unifies collision domains)
PC2
192.168.1.1
R1
subnet
10.0.0.2/31
R2
core router
subnet
10.0.0.4/31
R3
S2
subnet
10.0.8.96/27
10.0.8.108
10.0.8.76
S1
subnet
10.0.8.64/27
subnet
192.168.1.0/24

Tasks:

1. Create the network topology
2. Configure the IPs of all the nodes using /etc/network/interfaces file:
   a. Give two address of choice to PC1 and PC2. Which is the highest number of PC that the subnet can host?
   b. For /31 subnets, the addresses are assigned with the following rule: the lower router number takes the lower IP address
   c. Assign to S1 the **highest assignable** address and to S2 the **lowest** one
3. Configure a default gateway route for S1 and S2 via /etc/network/interfaces file
4. Configure the routes of R1, R2 and R3 via the .startup scripts (everybody should be able to reach everybody else). Moverover, on the core router **R2**:
   a. *don't* use default routes
   b. Use the minimum amount of entries in the routing table
5. Use netcat on PC1 to send the string "ciao" to S1 on port 8080. Capture this communication with tcpdump on R2 and saving the result on the file "/shared/capture.pcap"

Reminder:

When delivering the homework it should be already configured!

When we run your homework, it should configure itself automatically via .startup scripts and/or configuration files, as requested

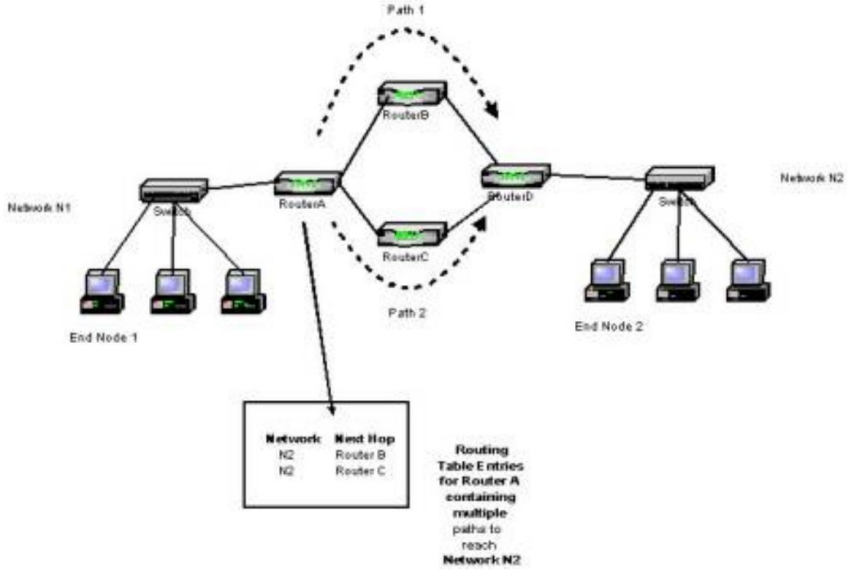
Dynamic Routing - OSPF

Where does static routing fails?

Most of the problems of static routing can be reconducted to:

- **Scalability** issues
  - Scalability because as network increases in complexity, **configuring every single router** for every single path can be a tough task, in particular if you want to take into account the different path costs in a redundant network.
- **Reliability** issues
  - Reliability because if a link goes down, the network cannot **failover** a redundant link **automatically**.
- **Heterogeneity** issues
  - Heterogeneity because the network can be composed of different router made by different vendors with different OSs, meaning that the sysadmin must be able to configure them all.

Redundant network example



Dynamic (adaptive) routing

- Once we have a routing protocol, we can **abstract** from how it is implemented and develop **platform independent** architectures
  - solves heterogeneity issues.
- A good routing protocol must **discover failures automatically** and react properly
  - solves reliability issues.
- The routing protocol must provide control messages to **all** other nodes using the same protocol to exchange informations (e.g. subnets advertisement)
  - solves scalability issues.

Taking a step back - Internet organization

In the homework we have managed the assignment of IP addresses, we managed routing between nodes, …

- We have behaved like an **Autonomous System** (**AS**)

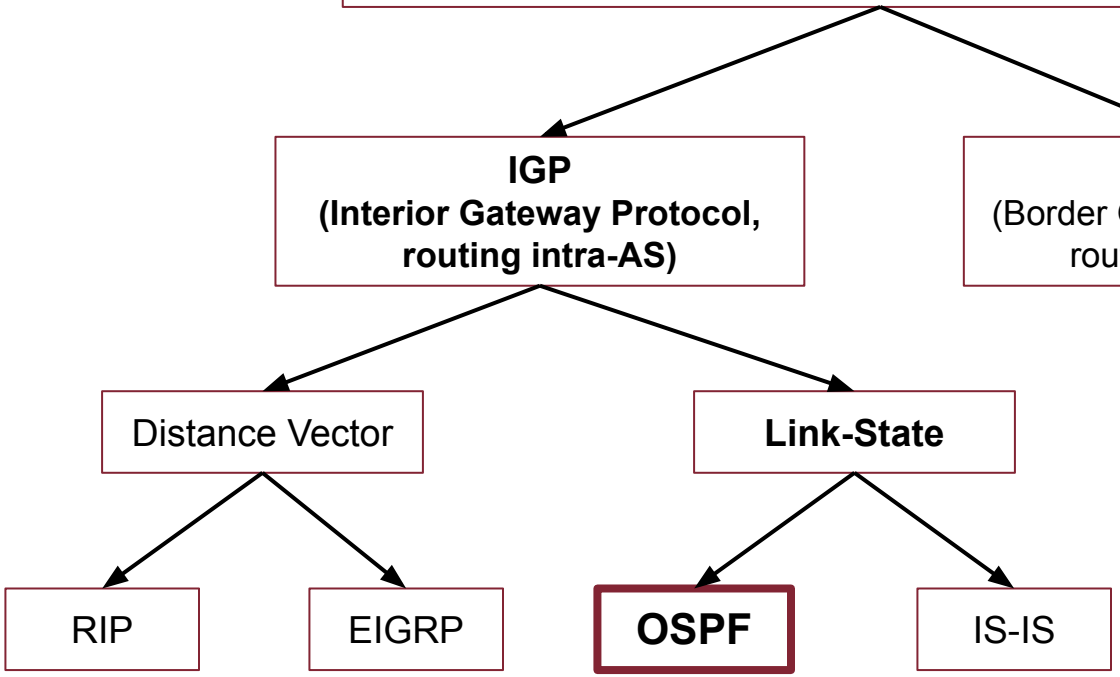
An **Autonomous System** is an entity, identified by a unique number, controlling a set of IP *prefixes* (i.e. subnets, IP addresses), that presents a common and clearly defined routing policy to the Internet.

Assignment of IP addresses is managed by the **IANA** organization (Internet Assigned Numbers Authority). It gives addresses to several Regional Internet Registries (**RIR**s), one per continent:

- AfriNIC Africa
- APNIC Asia e Oceano Pacifico
- ARIN Nord America
- LACNIC America Latina e Caraibi
- RIPE NCC Europa, Medio Oriente e Asia Centrale

Autonomous Systems receive IP addresses by RIRs and have to manage routing **intra-AS** (inside their AS) and **inter-AS** (between the AS and the rest of the Internet)

Routing protocol families



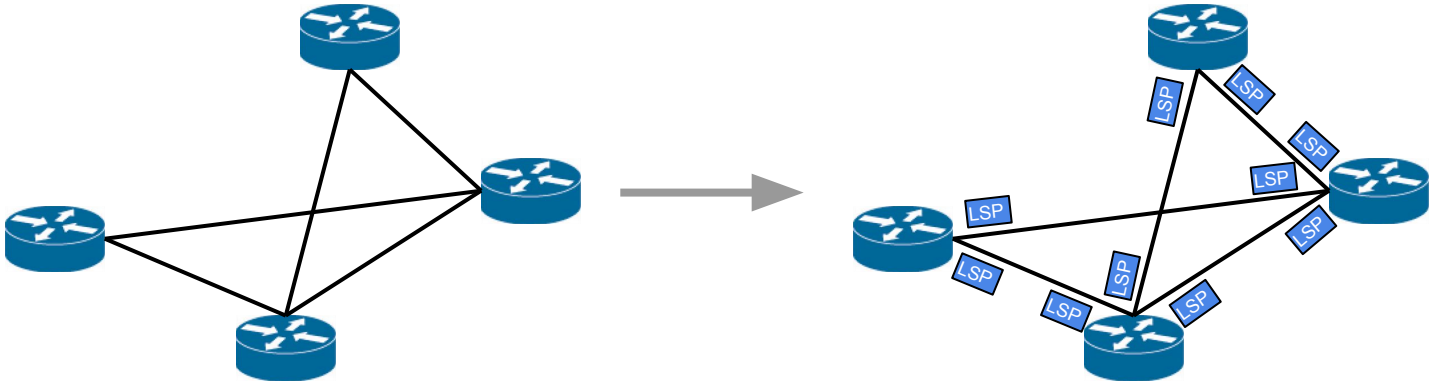
In **link-state** routing protocols, *every* node builds a map of the connectivity of the *whole* network. Each node then can *independently* calculate the best routing path

We will focus on Open Shortest Path First (**OSPF**) protocol

OSPF protocol

Every router running OSPF advertise its presence to **neighbors** routers and ”**floods**” the network with particular messages called Link State Packets (**LSP**) which contains information about the link (in terms of metrics and state) to which the node is connected to.

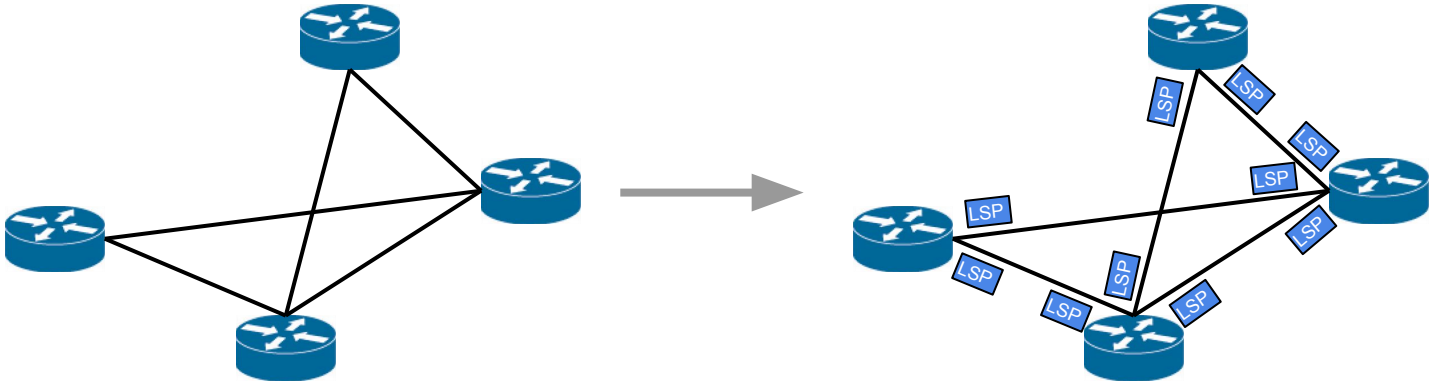
Every router collects all the received LSP in the Link-State Database (LSDB) and then constructs the network topology (a **weighted graph**) from the LSP received and, with the **Dijkstra algorithm**, builds the ”best path” for every single host. This will be the router’s **routing table**.



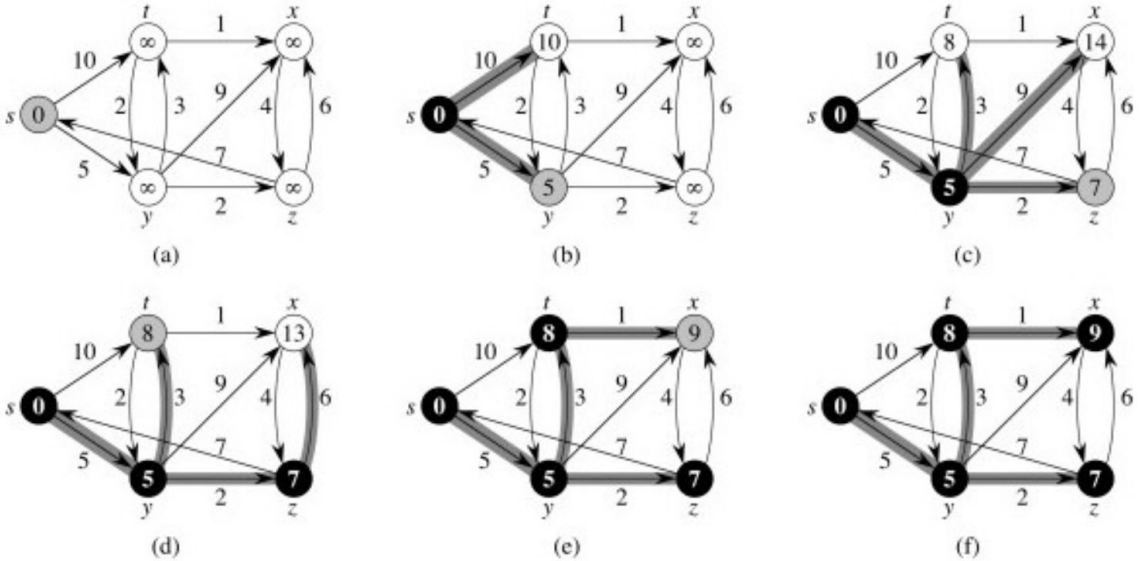
OSPF protocol

In other words, OSPF operates by:

1. Make all routers have the **same information** in their LSDB, by flooding the network with link state packets
2. **Run** Dijkstra algorithm, based on the network topology in the LSDB
3. **Populate** the routing table with the result of the algorithm



Dijkstra algorithm example



OSPF in Linux

In one of the available implementation on Linux, OSPF is included in a broader suite called **quagga**. The daemon which implements OSPF is called **zebra**. In order to have OSPF up and running, we must first configure the zebra daemon and then the OSPFv2 daemon.

One can access the OSPF console by typing "`telnet localhost ospfd`"

We can use the file `/etc/quagga/daemons` to configure the which daemons we want to run. Example configurations can be:

```
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
isisd=no
ldpd=no
```

In order to start quagga, we have to run the command `/etc/init.d/quagga restart` which can be placed in startup commands

File /etc/quagga/zebra.conf

`zebra.conf` file configures on which interface and addresses enable OSPF

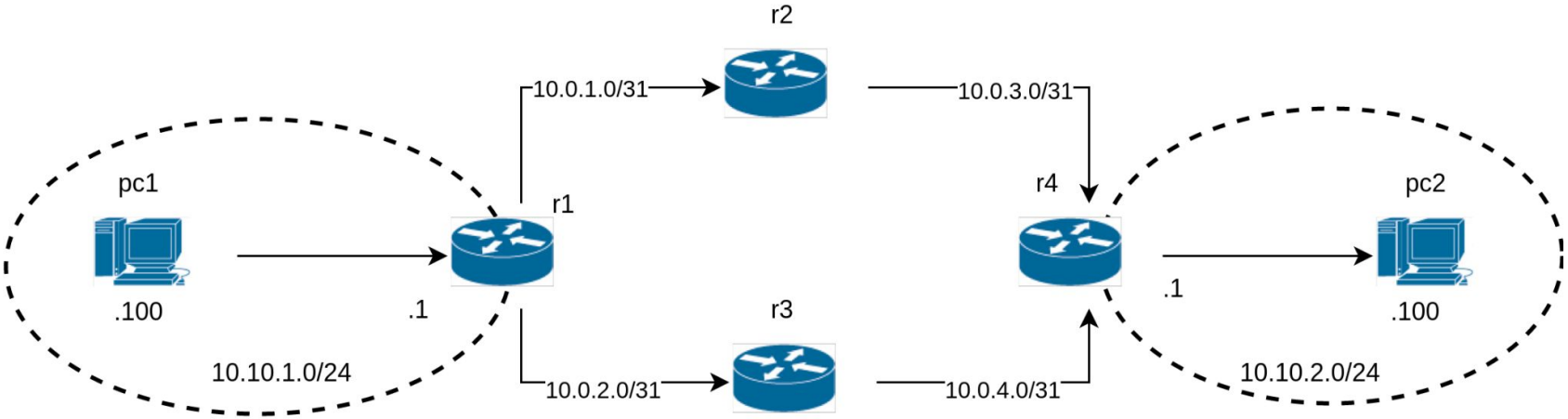
```
! -*- zebra -*-
!
! zebra sample configuration file
!
! $Id: zebra.conf.sample,v 1.1.1.1 2002/12/13 20:15:30 paul Exp $
!
hostname r1
password zebra
enable password zebra

interface eth0
ip address 1.0.10.2/31
link-detect

interface eth1
ip address 1.0.10.9/31
link-detect

interface eth2
ip address 1.0.10.13/31
link-detect

interface eth3
ip address 1.0.10.19/31
link-detect
```

File /etc/quagga/ospfd.conf

`ospfd.conf` file configures various OSPF parameters, including status packet interval (in seconds) and the OSPF area of subnets

```
hostname r1
password zebra

interface eth0
ospf hello-interval 2

interface eth1
ospf hello-interval 2

interface eth2
ospf hello-interval 2

interface eth3
ospf hello-interval 2

router ospf
network 1.0.10.18/31 area 0.0.0.0
network 1.0.10.2/31 area 0.0.0.0
network 1.0.10.8/31 area 0.0.0.0
network 1.0.10.12/31 area 0.0.0.0
```

lab_ospf_small

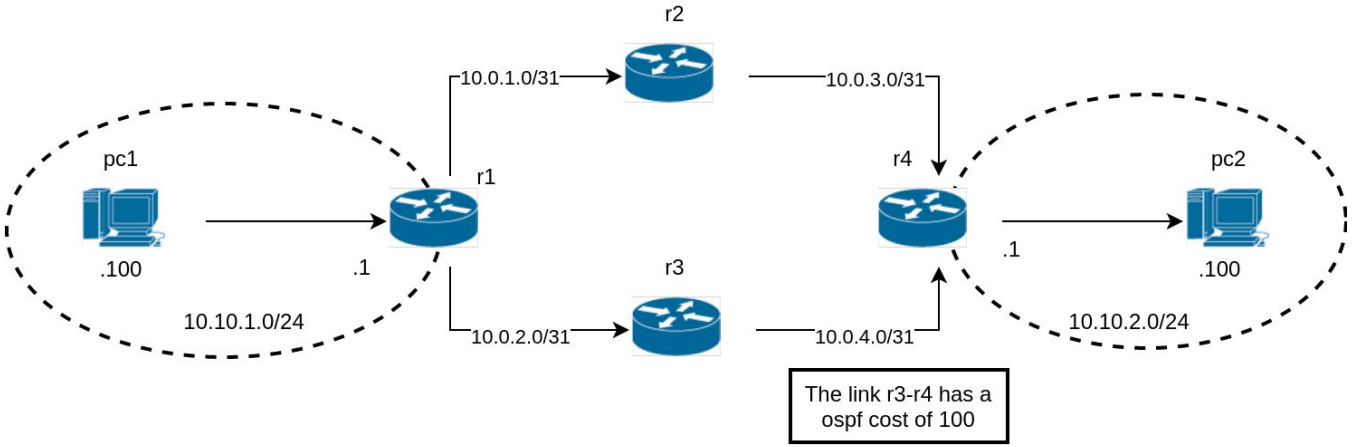


lab_ospf_small_manual-cost

OSPF daemon allows us to specify **cost** for each physical **link** on a router. The cost is a number greater than zero that has no physical meaning, it's just relative to the other links cost: a path with a **lower cost** respect to another will win the race and become part of a router's routing table. The default in our labs is **10.**

The cost of a link could be manually specified, in `ospfd.conf` configuration file.

One can observe the cost of each route given by OSPF by logging in with "`telnet localhost ospfd`", password "`zebra`", and then "`show ip ospf route`".



OSPF areas

Areas are used to **logically divide** different routing domains.

The routers belonging to different areas **do not know** the **entire** network topology of the other area, but use the Area Border Router as **gateway**.

- Routers which connects two or more areas are named Area Border Router (**ABR**)

Areas are identified by octets, like ip addresses. The area 0.0.0.0 is called the backbone area.

In lab_ospf_areas we have as ABR: r1, r2, r3

lab_ospf_areas