Laboratory 4 Dynamic routing using OSPF. Netfilter.

OSPF Protocol

Introduction

OSPF (Open Shortest Path First) routing protocol is a dynamic routing protocol of type link-state unlike RIP and IGRP who are distance-vector type. This means that all routers know the complete topology of the network and routing decisions can be made without risk of generating loops in the network. The metric used is based on a cost that is usually available band on each link, while the RIP metric was given by the number of hops (this means that the RIP will prefer a direct path between two nodes on a low speed link instead of an indirect path, with an intermediate node, but on a Gigabit Ethernet link). Another advantage of OSPF is the larger number of hops supported (255 to 15) which allows operation in midsize / large networks.

Like RIP, OSPF protocol is based on a standard (open) so it is possible to interplay among several manufacturers of routers. The algorithm for determining the minimum distance between any two nodes (shortest path) is Dijkstra's algorithm.

RIP and OSPF are interior routing protocols, which are used within an area under a common administration. One such area is called AS (Autonomous System). Routes, protocols, routing policies and the other technical and administrative aspects are defined in a uniform manner and are under the control of AS. By comparison, there are exterior routing protocols (e.g., BGP) that are used between different AS.

In terms of operation, OSPF is significantly more complex than RIP. Routing messages exchanged between routers are called LSA (Link State Advertisements). A router sends in LSA information such as the state of its links with the other connected routers (link states). Based on LSA messages each router creates a table (link state database) containing the network topology. These tables should be the same for all routers in the network (or, as we shall see, even within a country). Applying SPF algorithm on this table, each router obtains minimum cost routes to all destinations in the table, which forms routing table (routing table).

To reduce the number of LSA in large networks the concept of area is used. An area represents a certain part of the network (Cisco recommends less than 50 routers in an area, but one can use much smaller areas, too - depending on the speed of available connections and the routers' computation power). There is an area called the Backbone Area - the area that binds all other areas, if any. It is mandatory numbered as Area 0, while the other areas can be numbered anyway. Link state information are identical only within an area. Link state information on interior routers are not transfer outside the area, but only information about edge routers (Area Border Routers), making OSPF AS scalable in large networks. RIP

Chosing DR/BDR

A router creates a table with all its directly connected neighbors called Adjacency Database. In the case of a point-to-point network such as serial networks, who have exactly 2 nodes, it is obvious that each router has only one neighbor on each such network. But there are broadcast networks (the messages are broadcasted to many nodes) such as Ethernet networks that can connect directly many nodes. To minimize the number of LSA on such a network a designated router, DR (Designated Router), and a "reserve" called BDR (Backup Designated Router) are chosen. Only the designated router is receiving / disseminating routing information from / to other routers in the other networks.

Choosing a DR / BDR is done on all broadcast networks, even if there are only two routers in such a network and even when there is one (and other nodes are PCs that do not participate in the routing process). The choice is based on two numerical values:

- router ID,
- priority – at the interface level.

If there is not a manually configured router ID, the router chooses highest IP address (number of 32-bit) of the existing router interfaces. If there are loopback interfaces, the one with the bigger IP address is chosen, even if its IP is less than the IP of a non-loopback interfaces. This is because the loopback interfaces are stable (do not become "down" due to disconnection of a cable, for example).

Priority of each interface is one by default and can be changed manually between 0 and 255. The priority is considered at the interface level because a router can connect to multiple broadcast networks if it has more interfaces; in each network, it could have a different status -in a network could be DR, while in other not.

The process of choosing the network is as follows:

- router with the highest priority becomes the DR and BDR becomes the next. Priority 0 means that the router cannot be DR.
- In case of equal priorities, the election is based on router ID.

Wildcard Mask

Even if the OSPF operation is fundamentally different of RIP, within a single zone the configuration is very similar. A notable difference is the use of wildcard mask to define networks.

A `wildcard mask` (like that used in ACL) is a 32-bit number that allows "masking" or more precisely "selection" of certain bits of an IP address that you specify the desired network. The rule is: for a bit of the address that is selected, the position in the mask must be `"0"`. So `"0"` has the meaning `"Check"` and `"1"` has the meaning `"Ignore"`.

In OSPF is noted that wildcard mask is the inverse of netmask. For example, a class C network `192.168.1.0/24` has the `wildcard mask` `255.255.255.0` while the `netmask` is `0.0.0.255`. Similarly, a network `192.168.100.128/26` with the `wildcard mask`

255.255.255.192 has the netmask 0.0.0.63 (last byte is 11,000,000 in the netmask and 00,111,111 in the wildcard mask).

Single area OSPF protocol configuration

Configuration is as follows: for each router, start OSPF routing process with a process id (number of your choice) and specify each direclty connected network, wildcard mask, and its area:

```
Router(config)# router ospf 1
Router(config-router)#network networkIP wildcard_mask area number
Router(config-router)#network networkIP wildcard_mask area number
Router(config-router)# … other directly connected networks …
Router(config-router)# exit
```

Process Id is unimportant in this case, it could be useful if starting several routing processes on the same router. OSPF, like RIP v2, supports VLSM (variable length subnet mask -subnet masks are of different lengths).

Commands to debug OSPF and to inspect the routing table:

command	*description*
show ip route	Shows the routing table.
show ip route *destination_network*	Info about a specific destination.
show ip protocols	Info about running routing protocols.
show ip ospf	Info about OSPF.
show ip ospf database	Info about OSPF routing table.
show ip ospf interface	Info about OSPF running on the specified interface.
show ip ospf neighbor	Info about directly connected neighbors.

Multi-area OSPF protocol configuration

Advantages of several areas splitting:

- Smaller routing tables;
- Less signaling information (avoid flooding the network);
- Less computing power at routers;
- The frequency of SPF (Shortest Path First) tree computation is decreased.

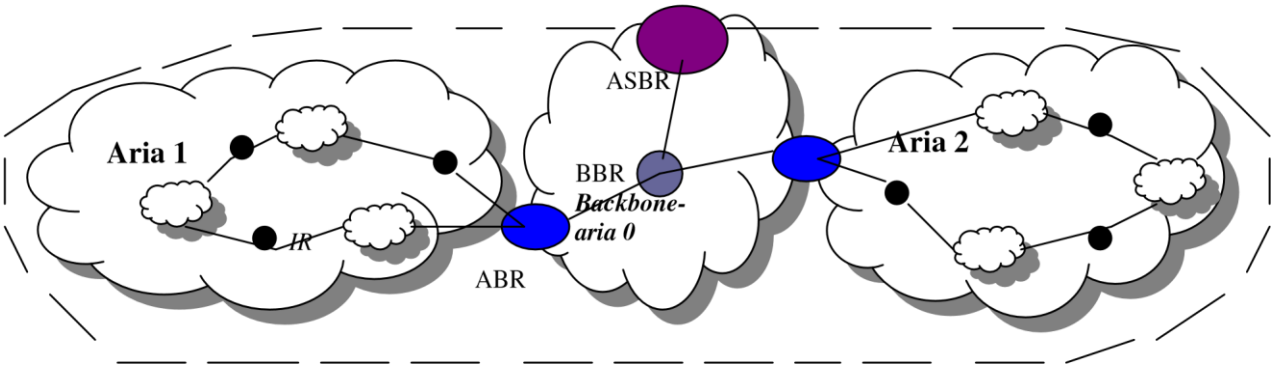

Figure 1. Multi-area splitting (Area 0 –Backbone,
Area 1 and Area 2)

IR – Interior Router

BBR – Backbone Router

BR – Border Router

ABR – Area Border Router

ASBR – Autonomous System Boundary Router

IR will run a routing process for the network inside its area; He fully knows only its area.

An area border router (which is connected to the backbone - ABR) will run a process instance to its area and one process instance for Backbone.

Router is charged with the connection with other autonomous systems, ASBR, is running a process instance for Backbone and an exterior routing protocol - EGP (External routes information are distributed within the domain by ASBR).

Note 1: Each area has:

- Its own algorithm OSPF;
- Its own database;
- Area topology is not visible externally.

Note 2: to further reduce the number of LSA messages exchanged between areas, those areas that connect through a single point in the remaining AS (have one ABR, for example area 2 in Figure 2) and does not communicate with the outside AS (so not contain ASBR) can be declared *stub areas*. Stub areas use *default routes* (0.0.0.0) to communicate with destinations outside that area. Aria N is declared manually stub with the command `area N stub` on the ABR. This has the effect that the area N will not receive LSA about external routes (and summarized routes in other areas of the same AS, in Cisco version called *totally stub*).

Configurations:

A. Setting up the router within the area is done as for single-area OSPF (section 1.5).

B. Setting up the router that connects two or more areas each other (areas belonging to the same AS) is done by configuring the interfaces to operate in corresponding areas.

For example, in a topology as shown below (where `Router1` is area 0, `Router2` is in `area 1`, and `Router0` connects `area 0` with `area 1`): `Router0` is therefore an ABR, interconnecting `areas 0` and `area 1`. We could have two OSPF processes or only one, but specifying each area that it is serving:

```
Router0(config)#router ospf 40
Router0(config-router)#network 172.16.0.0 0.0.255.255 area 1
Router0(config-router)#network 192.168.14.0 0.0.0.255 area 0
```

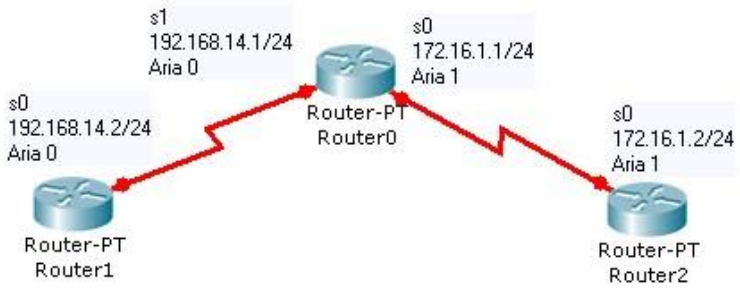

Figure 2. Topology example (Router0 is ABR)

Commands to display info about routing tables on ABR or ASBR:

command	*description*
`show ip ospf border-router`	Displays routing table of ABR.
`show ip ospf` *`process-id`*	Info about the areas a router is connected.
`show ip ospf database summary`	Displays information only about the summary LSAs.
`show ip ospf database asbr-summary`	Displays information only about the autonomous system boundary router summary LSAs.
`show ip ospf database external`	Displays information about external LSAs.
`show ip ospf database databasesummary`	Displays how many of each type of LSA for each area there are in the database, and the total.

Routes summarization (route aggregation)

Recall that the *summarization* means bundling of several routes / destinations into one (`n` routes within a specific area are perceived outside that area as one route). This requires continuous addressing space in that area - as in the case RIPv2, if subnets of the same net are randomly distributed across multiple routers, one cannot make summarization.

OSPF summarization types:

- ***External summarization*:**

Autonomous systems AS (by administrative, economic or security reasons) do not transmit internal routing information completely to other autonomous systems that are connected via an ASBR router, but only a *summary* thereof, enough to inform on each network of AS. Whenever possible, the subnets composing a network are combined into a single network.

ASBR router has the task of summarizing routes to subnets within an AS to which it is connected. Use the command:

- `ASBR(config)# router ospf 1`
- `ASBR(config-router)#`**`summary-address`** *`address mask`*

where `address` and `mask` specify the super-network (the summarized network).

If we combine the subnets `10.10.1.0/24, 10.10.2.0/24, …, 10.10.255.0/24` in the super-network `10.10.0.0/16`, the command is:

- `ASBR(config-router)#summary-address 10.10.0.0 255.255.0.0`

- ***Inter-area summarization**:*

Summarization can be used between the areas within an AS. The command is:

- `ABR(config)# router ospf 1`
- `ABR(config-router)#`**`area`** *`id`* **`range`** *`address mask`*

where `address` and `mask` specify the super-network, and `id` is the area identifier. For example, for the ABR located at the border between areas 1 and 0, if the subnets are in area 1, the summarized network is injected in area 0 with the command:

- `ABR(config-router)#`**`area`** `1` **`range`** `10.10.0.0 255.255.0.0`

Netfilter. Iptables.

Introduction [2]

The Linux kernel comes with a packet filtering framework named **netfilter**. It allows you to allow, drop and modify traffic leaving in and out of a system. A tool, **iptables** builds upon this functionality to provide a powerful firewall, which you can configure by adding rules. In addition, other programs such as fail2ban also use iptables to block attackers.

Iptables is just a command-line interface to the packet filtering functionality in netfilter. However, to keep this article simple, we won't make a distinction between iptables and netfilter in this article, and simply refer to the entire thing as "iptables".

The packet filtering mechanism provided by iptables is organized into three different kinds of structures: **tables, chains** and **targets**. Simply put, a table is something that allows you to process packets in specific ways. The default table is the filter table, although there are other tables too.

Again, these tables have chains attached to them. These chains allow you to inspect traffic at various points, such as when they just arrive on the network interface or just before they're handed over to a process. You can add rules to them match specific packets — such as TCP packets going to port 80 — and associate it with a target. A target decides the fate of a packet, such as allowing or rejecting it.

When a packet arrives (or leaves, depending on the chain), iptables matches it against rules in these chains one-by-one. When it finds a match, it jumps onto the target and performs the action associated with it. If it doesn't find a match with any of the rules, it simply does what the default policy of the chain tells it to. The default policy is also a target. By default, all chains have a default policy of allowing packets.

Tables [2]

As we've mentioned previously, tables allow you to do very specific things with packets. On modern Linux distributions, there are four tables:

- The ***filter*** table: This is the default and perhaps the most widely used table. It is used to make decisions about whether a packet should be allowed to reach its destination.
- The ***mangle*** table: This table allows you to alter packet headers in various ways, such as changing TTL values.
- The ***nat*** table: This table allows you to route packets to different hosts on NAT (Network Address Translation) networks by changing the source and destination addresses of packets. It is often used to allow access to services that can't be accessed directly, because they're on a NAT network.
- The **raw** table: iptables is a stateful firewall, which means that packets are inspected with respect to their "state". (For example, a packet could be part of a new connection, or it could be part of an existing connection.) The raw table allows you to work with packets before the kernel starts tracking its state. In addition, you can also exempt certain packets from the state-tracking machinery.

Chains [2]

Now, each of these tables are composed of a few default chains. These chains allow you to filter packets at various points. The list of chains iptables provides are:

- The PREROUTING chain: Rules in this chain apply to packets as they just arrive on the network interface. This chain is present in the nat, mangle and raw tables.
- The INPUT chain: Rules in this chain apply to packets just before they're given to a local process. This chain is present in the mangle and filter tables.
- The OUTPUT chain: The rules here apply to packets just after they've been produced by a process. This chain is present in the raw, mangle, nat and filter tables.
- The FORWARD chain: The rules here apply to any packets that are routed through the current host. This chain is only present in the mangle and filter tables.
- The POSTROUTING chain: The rules in this chain apply to packets as they just leave the network interface. This chain is present in the nat and mangle tables.

Targets [2]

As we've mentioned before, chains allow you to filter traffic by adding rules to them. So for example, you could add a rule on the filter table's INPUT chain to match traffic on port 22. But what would you do after matching them? That's what targets are for — they decide the fate of a packet.

Some targets are terminating, which means that they decide the matched packet's fate immediately. The packet won't be matched against any other rules. The most commonly used terminating targets are:

- ACCEPT: This causes iptables to accept the packet.
- DROP: iptables drops the packet. To anyone trying to connect to your system, it would appear like the system didn't even exist.
- REJECT: iptables "rejects" the packet. It sends a "connection reset" packet in case of TCP, or a "destination host unreachable" packet in case of UDP or ICMP.

On the other hand, there are non-terminating targets, which keep matching other rules even if a match was found. An example of this is the built-in LOG target. When a matching packet is received, it logs about it in the kernel logs. However, iptables keeps matching it with rest of the rules too.

Sometimes, you may have a complex set of rules to execute once you've matched a packet. To simplify things, you can create a custom chain. Then, you can jump to this chain from one of the custom chains.

Examples [2]

The most common use for a firewall is to block IPs.

- `iptables -t filter -A INPUT -s 59.45.175.62 -j REJECT`

The `-t` switch specifies the table in which our rule would go into — in our case, it's the `filter` table.

The `-A` switch tells iptables to "append" it to the list of existing rules in the INPUT chain.

The `-s` switch simply sets the source IP that should be blocked.

Finally, the `-j` switch tells iptables to "reject" traffic by using the REJECT target. If you want iptables to not respond at all, you can use the DROP target instead.

- `iptables -A INPUT -s 59.45.175.62 -j REJECT`

To block a range of IP addresses one can ue the CIDR notation.

- `iptables -A INPUT -s 59.45.175.0/24 -j REJECT`

Iptables can be used to block protocols, too. For example, to block all incoming ICMP traffic, one simply need to specify the protocol.

- `iptables -A INPUT -p icmp -j DROP`

A more complex example:

- `iptables -A INPUT -p tcp -m tcp --dport 22 -s 59.45.175.0/24 -j DROP`

It blocks SSH access for an IP range. First match all TCP traffic, In order to check the destination port, you should first load the `tcp` module with `-m`. Next, you can check if the traffic is intended to the SSH destination port by using `--dport`.

Part 2: Experimental part

Build basic network topologies and network nodes configuration for a better understanding of the OSPF protocol. Learn to filter traffic using access lists.

Remark:

Do not forget to set the Netkit environment variables and check your configuration:

- `export NETKIT_HOME=~/netkit`
- `export MANPATH=:$NETKIT_HOME/man`
- `export PATH=$NETKIT_HOME/bin:$PATH`

Check your configuration:

- `./check_configuration.sh`

Study of OSPF protocol

In this part a topology consisting of five routers will be implemented as in Figure 3. OSPF protocol will be activated and configured on each router. OSPF functionalities will be investigated in several scenarios.

Network topology

The network topology is presented in Figure 3.

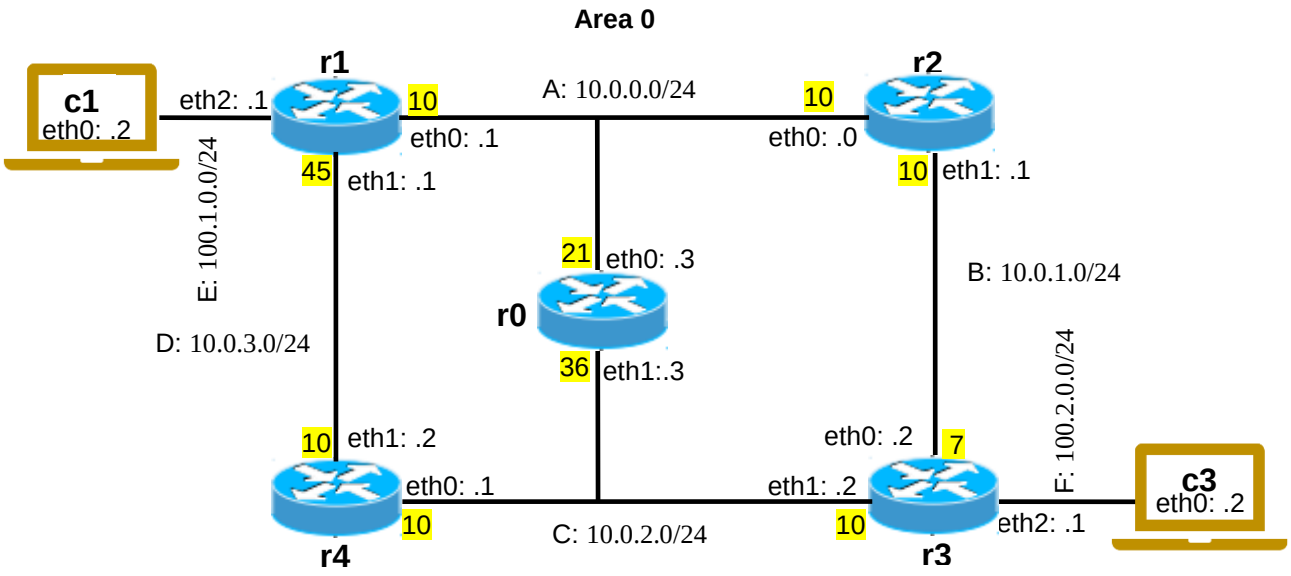

Figure 3: Network topology

Each interface is assigned an ospf cost. With yellow the cost for each interface is specified.

- `default: 10`

Tthe costs on some interfaces is modified to force OSPF choose some paths taken by traffic. To set interface costs one must enter in the configuration mode:

- `interface eth1`
- `ospf cost 45`

Building the topology lab

Inside the `netkit` folder create a new folder `lab4`. Inside the `lab4` folder create the structure of subdirectories and files associated with the topology.

- Create an empty folder for each device in the topology:`c1` folder, `c2` folder, …`r1` folder, `r2` folder, ….
- Create the `lab.conf` file where the topology is described:
  - `c1[0]=B`
  - …
  - `r0[0]=A`
  - `r0[1]=C`
  - `r1[0]=A`
  - `r1[1]=B`
  - `r1[1]=E`
  - …
- Create the `.startup` files for each device were the initial configuration is specified:
  - `c1.startup`:
    - `ifconfig eth0 100.1.0.2 netmask 255.255.255.0 up`
    - `route add default gw 100.1.0.1`
  - `c3.startup`
    - …
  - `r0.startup`
    - `ifconfig eth0 10.0.0.3 netmask 255.255.255.0 up`
    - `ifconfig eth1 10.0.3.3 netmask 255.255.255.0 up`
  - `r1.startup`
    - …
  - …
    - …

After the configuration of the IP addresses through the `startup` files, let's configure the OSPF. OSPF can be configured either after the lab is started using the `vtysh` interface or before the lab is started using configuration files. For this lab we will use the second method.

The configuration files for Zebra and OSPF must be located in the folders belonging to the network nodes: `r0` folder, `r1` folder, etc. In each router's folder, the following folders structure must be created (Figure 4):

- In `r0` folder:
  - `etc` folder
  - inside `etc` folder create `zebra` folder.
- Repeat the operation for each other router.

The directories structure and the configuration files are presented in Figure 4. The capture is made in Windows. For Linux, where Netkit is running the path to the configuration files will be:

- `# ~/netkit/lab4/r0/etc/zebra/`

In each `zebra` folder on each router two configuration files will be created: `daemons` and `ospfd.conf`. The content of each file is presented in Tables 1 and 2

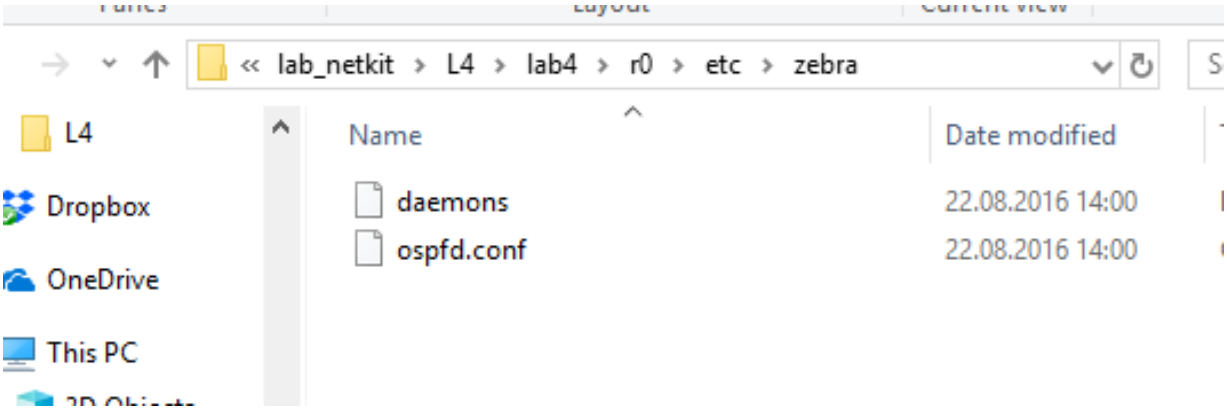

Figure 4: The directory tree and the configuration files for `zebra` and `ospf`

The daemons file is used to configure the routing daemons that will be used on the router. In this case the `zebra` process, which is the core of the routing software, and `ospfd` process, which is the module implementing the OSPF routing protocol, will be activated.

```
# This file tells the zebra package
# which daemons to start.
# Entries are in the format: <daemon>=(yes|no|priority)
# where 'yes' is equivalent to infinitely low priority, and
# lower numbers mean higher priority. Read
# /usr/doc/zebra/README.Debian for details.
# Daemons are: bgpd zebra ospfd ospf6d ripd ripngd
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
```

Table 1: Content of the daemons file

The `ospfd.conf` file contains the configuration commands for OSPF daemon. The configuration commands can be inserted through the zebra command line accessed with `vtysh`. Building the `ospfd.conf` file has the advantage that the configuration will be maintained when the lab is restarted.

```
!
hostname ospfd
password zebra
enable password zebra
!
! Configure the cost on interfaces
! Only for those interfaces that have the cost different of 10
! Default cost for exiting an interface is 10
interface eth0
ospf cost 21
interface eth1
ospf cost 36
!
!configure OSPF
router ospf
! Speak OSPF on all interfaces falling in 10.0.0.0/16
network 10.0.0.0/16 area 0.0.0.0
redistribute connected
!
log file /var/log/zebra/ospfd.log
!
```

Table 2: Content of the `ospfd.conf` file on router `r0`

Remark 1: Pay attention to update the content of the `ospfd.conf` file on each router according to the lab topology!

Remark 2: For routers `r1` and `r3`, which have the computers connected to them through some local networks that are not included in the network domain `10.0.0.0/16`, those local networks must be configured separately in ospf configuration file.

For example in `ospfd.conf` file on r1 an additional line must be included in router ospf configuration section:

- `network 100.1.0.0/24 area 0.0.0.0`

After all the lab directories and files are defined, start the lab with `lstart` command:

- **`~/netkit$`** `lstart -d ~/netkit/lab4`

Check OSPF's functionalities

- Use `ping` command to test connectivity between the lab nodes.
- Inspect routing tables on the routers with `s hip route` command. What types of routes do exist in the routing table?
- perform a `traceroute -I` from `c1` to `10.0.2.1`
  - what path is the traceroute expected to take?
  - what path are ICMP replies expected to take?
- perform a `traceroute -I` from `c1` to `10.0.3.2`
  - what path is the traceroute expected to take?
  - observe the interplay between ospf routes and directly connected networks
- Use `show ip ospf interface` to determine which the `designated router` on each network is.

- Use `show ip ospf database` on one router to determine the structure and the content of the ospf database.
- Use `show ip ospf database network` on router `r1` to determine which its perspective on the network topology is.
- Use `show ip ospf neighbor` on router `r1` to determine its neighbors.
- Capture the ospf pachets with tcpdump on a router interface and analyze it content. Which are the informations that are carried in ospf packets?

Investigate how fast ospf recovers from link fault.

- Start `tcpdump` on one interface of the router r0.
- To determine how fast the route is recovered start a `ping` command between two nodes that are on the path affected by by fault link.
- Bring down interface `eth0` on router `r1` using `ifconfig`.
  - Observe that the change is immediately propagated by the router inside lsa packets
  - Inspect the routing table with `show ip ospf route` command on router `r1`. Observe that the routing table is updated immediately.
- Interface `eth0` on router `r1` is the designated interface for the network A.
  - Inspect the the link state databases on the routers (`show ip ospf database network`)
    - Observe that the information is immediately flushed from the link state databases and eventually reannounced when a designated router is re-elected

- Determine the new routes established using `traceroute` command. Try to catch the routes that are affected by the link fault.

- Bring the interface `eth0` on router `r1` up to bring the topology at the initial state.

- Bring down interface `eth0` on router `r2` using `ifconfig`. `Eth0` on router `r2` is a normal interface
  - Inspect the the link state databases on the routers (`show ip ospf database network`)
    - in this case ospf waits expiry of the RouterDeadInterval timer (default: 40s) before removing the adjacency from the database

- Determine the new routes established using `traceroute` command. Try to catch the routes that are affected by the link fault.

OSPF multiarea

In this section a network topology with multiple OSPF areas will be implemented. The topology is given in Figure 5. Two stub networks are added to the OSPF area 0 given in Figure 3. They are defined as new OSPF areas, `area1` and `area 2`.

- different interfaces of the same router can be assigned to different areas
- each router interface, network, router adjacency.is associated with a single area

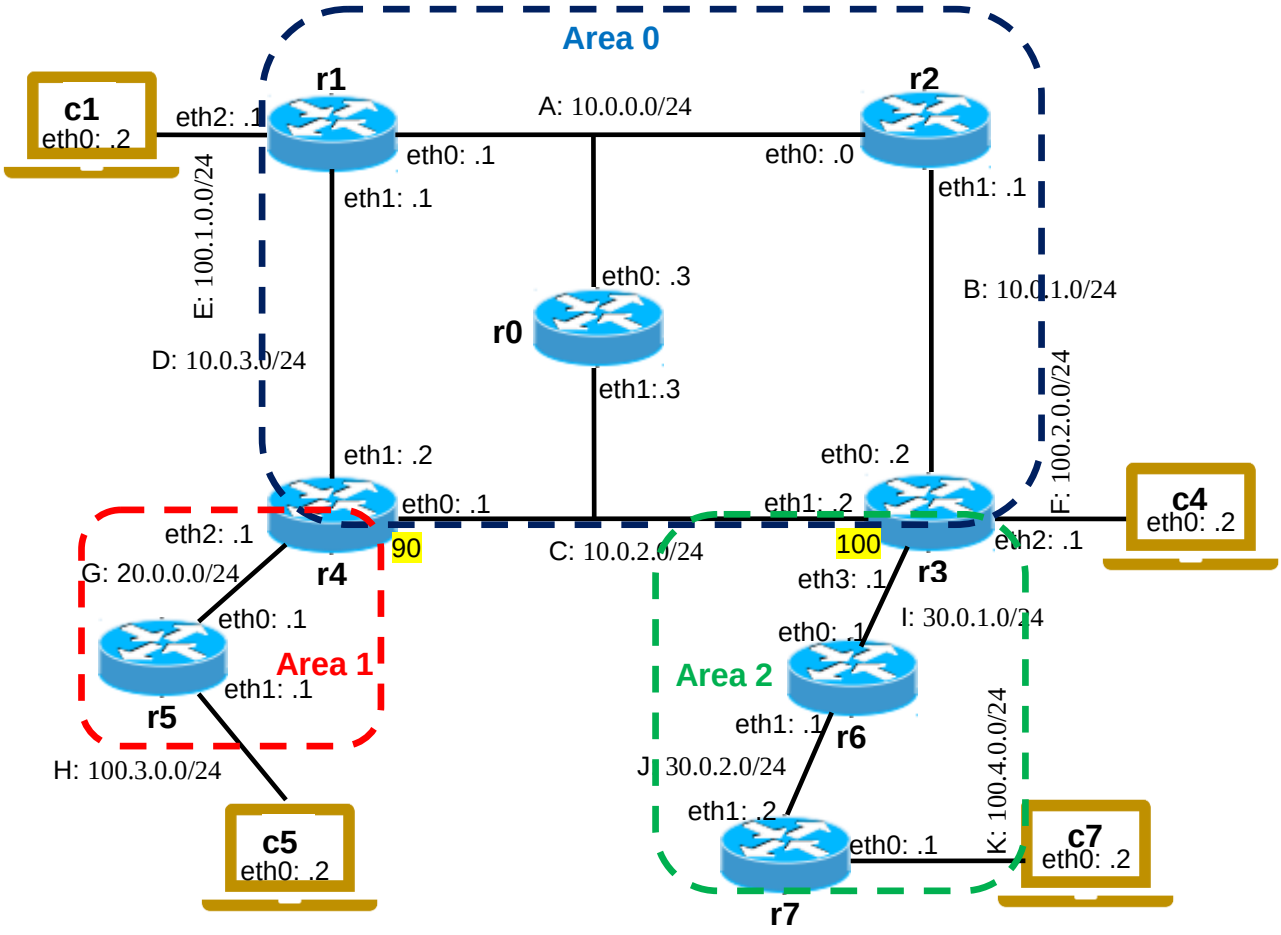

Figure 5: Network topology for OSPF multiarea

- Complete the topology defined for OSPF single area with the additional network components.
- Create the `startup` files and insert the initial configuration for the new devices.
- Create the `ospfd.conf` and `zebra.conf` fles on each new router.
- Start the lab with the new topology.

The configuration for OSPF multiarea on router `r4` is given in Table 3.

```
!
hostname ospfd
password zebra
enable password zebra
!
interface eth0
ospf cost 90
!
router ospf
! Speak OSPF on all interfaces falling in the listed subnets
network 10.0.0.0/16 area 0
network 20.0.0.0/24 area 1
area 1 stub
redistribute connected
!
log file /var/log/zebra/ospfd.log
!
```

Table 3: `ospfd.conf` file on router `r4`

The default costs are 10. Only non default costs are specified for the topology presented in Figure 5.

The configuration for OSPF multiarea on router `r6` is given in Table 4.

```
!
hostname ospfd
password zebra
enable password zebra
!
router ospf
! Speak OSPF on all interfaces falling in the listed subnets
network 30.0.0.0/16 area 2
area 2 stub
redistribute connected
!
log file /var/log/zebra/ospfd.log
!
```

Table 4: `ospfd.conf` file on router `r4`

Evaluation of OSPF functionalities

- Use `show ip ospf database network` to check that routers know detailed topology information only about their own area.
- check what routers know about the outside of the area, using the `show ip ospf database summary` command
  - check the Metric values, that show how far away the destination is from the advertising abr
- check that routers in stub areas are offered a default route, whereas routers in the backbone are not
  - also check what Metric is assigned to the default route
- Start `tcpdump` on one interfaces of router `r0`. Shut down an interface on a router in `area 0` and visualize the OSPF control packets that are exchanged as a reaction to the topology change.
- Repeat the scenario, but in this case the interface that will be shut down is `eth0` on router `r6`.

Study of Iptables

Traffic filtering with Iptables will be configured in this section.

Basic traffic filtering

In this section IP traffic originated from an IP address will be rejected. The traffic filter will be installed on `C4` machine. It will reject the packets coming from `C1` machine.

- On `C1` machine `ping` the `C4` machine
  - `C1:~# ping 100.2.0.2`
- Does `ping` work?

On `C3` build a script that will install an iptrables rule to reject the traffic coming from `C1`.

The script will be named `filter-script.sh` and will contain the lines:

- `#!/bin/sh`
- `iptables -A INPUT -s 100.1.0.2 -j REJECT`

To edit the file use nano editor (or other available editors):

- `C3:~#nano filter-script.sh`

After the file is edited and saved make it executable with the command:

- `C3:~#chmod a+rx filter-script.sh`

Before running the script inspect the iptables with the command:

- `C3:~#iptables -L -v`

Run the script with the command:

- `C3:~#./filter-script.sh`

Check that the filter is working:

- On `C1` machine `ping` the `C4` machine

    - C1:~# ping 100.2.0.2
- Does ping work? Why?

Remove the iptables rules using the script remove_rules.sh. Edit the script using nano editor:

- C3:~#nano remove_rules.sh

Insert the following lines into remove_rules.sh script:

- #!/bin/bash
- iptables -F FORWARD
- iptables -nL

After the file is edited and saved make it executable with the command:

- C3:~#chmod a+rx remove_rules.sh

Run the script with the command:

- C3:~#./remove_rules.sh

Filtering protocols

More complex filtering rules can be implemented with iptables. For example, protocols' packets can be filtered using iptables. In this section the ICMP packets will be filtered on C7 machine.

Implement a filtering rule to filter on C7 the ICMP packets coming from networks 100.3.0.0/24 and 100.2.0.0/24.

- Check that the filtering rule is working using ping command from C4 or C5 and C7.

Implement a filtering rule to block ssh access from network 100.3.0.0/24 on C4

- Check that the filtering rule is working.

References:

1. http://wiki.netkit.org/netkit-labs/netkit-labs_advanced-topics/netkit-labs_ospf/netkit-labs_ospf.pdf
2. https://www.booleanworld.com/depth-guide-iptables-linux-firewall/