Computer Networks Lab

based on
Mastering Networks - An Internet Lab Manual
by Jörg Liebeherr and Magda Al Zarki

and on
Computer Networking - A Top-Down Approach
by James Kurose and Keith Ross

Adapted for
'Computernetwerken'
by Johan Bergs and Stephen Pauwels

Tim Verhaegen
Pim Van den Bosch
Group 12

May 11, 2023

Lab 4

IP Routing

What you will learn in this lab:

- How to set up static routing.
- How different network prefixes impact the routing table.
- What happens in case of a routing loop.
- How Border Gateway Protocol (BGP), and Open Shortest Path First (OSPF) update the routing tables after a change in the network topology.
- How BGP and OSPF can be combined for inter- and intra-AS routing.

4.1 Static Routing

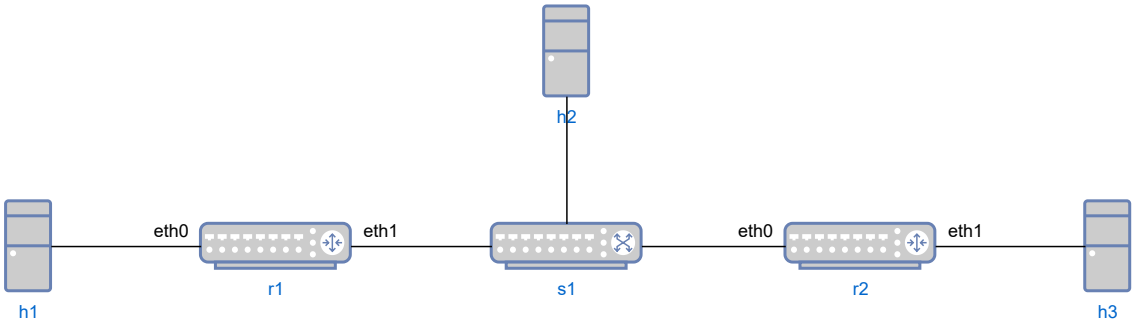


Figure 4.1: Network configuration

Host	eth0	eth1
h1	`fc00:0:0:1::1/64`	N/A
h2	`fc00:0:0:2::2/64`	N/A
h3	`fc00:0:0:3::3/64`	N/A
r1	`fc00:0:0:1::10/64`	`fc00:0:0:2::10/64`
r2	`fc00:0:0:2::11/64`	`fc00:0:0:3::10/64`

Table 4.1: Internet Protocol (IP) addresses

In this lab, we will start with the topology shown in figure 4.1. By default, when an *ipmininet router* is instantiated, it starts up an OSPF daemon so that all routes in your topology are properly set automatically. Also, each *host* is configured with a default route to one of the routers in its subnet(s).

In this exercise, we want to have a look at the routing tables without OSPF automatically updating them. You can do this by specifying some parameters in you script:

- First of all, make sure you add `from ipmininet.router.config import RouterConfig, STATIC, StaticRoute` to your script.
- Secondly, every time you create a router, add an extra parameter `config=RouterConfig` to the router definition.

This will make sure the routers do not start up an OSPF daemon automatically.

Exercise 1: Configuring static routes

1. Write a python script to implement the above topology **and make sure the OSPF daemons are not started on the routers!** Save the script to `L4-1-1.py`.
2. Start the mininet script, and issue a `pingall` command. Save the output of the `pingall` to `L4-1-1.pingall.txt`
3. On every host and router, inspect the **IPv6** routing table using either the `ip -6 route show` command. Save the routing tables to `L4-1-1.routes.before.txt` (and indicate which routing table belongs to which host).

L4-1-1 Using the output of the routing tables and of the `pingall` command, explain which pings succeed and which pings fail and why. /1

```
mininet> pingall
*** Ping: testing reachability over IPv4 and IPv6
h1 --IPv6--> X X
h2 --IPv6--> X h3
h3 --IPv6--> X h2
*** Results: 66% dropped (2/6 received)
```

traces/L4-1-1.pingall.txt

```
h1:

fc00:0:0:1::/64 dev h1-eth0 proto kernel metric 256 pref medium
fe80::/64 dev h1-eth0 proto kernel metric 256 pref medium
default via fc00:0:0:1::10 dev h1-eth0 metric 1024 pref mediu

h2:

fc00:0:0:2::/64 dev h2-eth0 proto kernel metric 256 pref medium
fe80::/64 dev h2-eth0 proto kernel metric 256 pref medium
default via fc00:0:0:2::11 dev h2-eth0 metric 1024 pref medium

h3:

fc00:0:0:3::/64 dev h3-eth0 proto kernel metric 256 pref medium
fe80::/64 dev h3-eth0 proto kernel metric 256 pref medium
default via fc00:0:0:3::10 dev h3-eth0 metric 1024 pref mediu

r1:

fc00:0:0:1::/64 dev r1-eth0 proto kernel metric 256 pref medium
fc00:0:0:2::/64 dev r1-eth1 proto kernel metric 256 pref medium
fe80::/64 dev r1-eth1 proto kernel metric 256 pref medium
fe80::/64 dev r1-eth0 proto kernel metric 256 pref medium

r2:

fc00:0:0:2::/64 dev r2-eth1 proto kernel metric 256 pref medium
fc00:0:0:3::/64 dev r2-eth0 proto kernel metric 256 pref medium
fe80::/64 dev r2-eth0 proto kernel metric 256 pref medium
fe80::/64 dev r2-eth1 proto kernel metric 256 pref medium
```

traces/L4-1-1.routes.before.txt

We initially have no routing tables set up in any of our routers. However, using the neighbour discovery protocol, hosts may still learn of routers that can forward their packets. Host h_2 in particular learns of router R_2 to send its packets to. h_1 learns of router r_1, and h_3 learns of router r_2.

There are 6 pings:

1. $H_1 \mapsto H_2$ (FAILED): H1 is able to send a ping request to H2, but H2 has no route for this IP address and forwards the response to the router it knows, R_2. This router's routing table is empty and no direct neighbours are the destination, the router thus drops the packet
2. $H_1 \mapsto H_3$ (FAILED): H1 sends the packet to R1, but since R1 has no route to this destination, it sends back the ICMPv6 message "Destination unreachable: No route"

3. $H_2 \mapsto H_1$ (FAILED): H2 has no route for this packet, so it forwards it to the router it knows, R2. R2 then drops the packet, it sends back the ICMPv6 message "Destination unreachable: No route"
4. $H_2 \mapsto H_3$ (SUCCEEDED): H2 sends the packet to R2, which has a route to H3
5. $H_3 \mapsto H_1$ (FAILED): H3 sends the packet to R2, but since R2 has no route to this destination, it sends back the ICMPv6 message "Destination unreachable: No route"
6. $H_3 \mapsto H_2$ (SUCCEEDED): H3 sends the packet to R2, which has a route to H2. Because R2 is used rather than R1, the response also gets back to H3

L4-1-2 Which hosts or routers need to be configured with one or more extra routes? Indicate **which routes** need to be added. /1

---

1. R1 needs a route for addresses within the fc00:0:0:3::/64 subnet
2. R2 needs a route for addresses within the fc00:0:0:1::/64 subnet

---

L4-1-3 Are there hosts or routers that do not require extra routes? Why (not)? /1

H2 does not require any additional routes if it can just reach one of the two routers. H1 and H2 also do not require any additional routes if they can reach their respective routers.

Add the required routes to the hosts and/or routers using the `ip route -6 add` command (look up the exact syntax required). When you are done, verify with `pingall` that all hosts can now reach each other. Save the output of the routing tables again to `L4-1-4.routes.after.txt`.

L4-1-4 Include the output of all routing tables: /1

```
R1 routes:

fc00:0:0:1::/64 dev r1-eth0 proto kernel metric 256 pref medium
fc00:0:0:2::/64 dev r1-eth1 proto kernel metric 256 pref medium
fc00:0:0:3::/64 via fc00:0:0:2::11 dev r1-eth1 metric 1024 pref medium
fe80::/64 dev r1-eth0 proto kernel metric 256 pref medium
fe80::/64 dev r1-eth1 proto kernel metric 256 pref medium

R2 routes:

fc00:0:0:1::/64 via fc00:0:0:2::10 dev r2-eth1 metric 1024 pref medium
fc00:0:0:2::/64 dev r2-eth1 proto kernel metric 256 pref medium
fc00:0:0:3::/64 dev r2-eth0 proto kernel metric 256 pref medium
fe80::/64 dev r2-eth0 proto kernel metric 256 pref medium
fe80::/64 dev r2-eth1 proto kernel metric 256 pref medium

H1 routes:
```

```
fc00:0:0:1::/64 dev h1-eth0 proto kernel metric 256 pref medium
fe80::/64 dev h1-eth0 proto kernel metric 256 pref medium
default via fc00:0:0:1::10 dev h1-eth0 metric 1024 pref medium

H2 routes:

fc00:0:0:2::/64 dev h2-eth0 proto kernel metric 256 pref medium
fe80::/64 dev h2-eth0 proto kernel metric 256 pref medium
default via fc00:0:0:2::11 dev h2-eth0 metric 1024 pref medium

H3 routes:

fc00:0:0:3::/64 dev h3-eth0 proto kernel metric 256 pref medium
fe80::/64 dev h3-eth0 proto kernel metric 256 pref medium
default via fc00:0:0:3::10 dev h3-eth0 metric 1024 pref medium
```

traces/L4-1-4.routes.after.txt

L4-1-5 From h1, perform a `traceroute` to h3, using the `-q 1 -N 1 -z 100` options, and then a second time using the additional `-I` option. While doing the traceroutes, run a wireshark capture on eth0 of r1 and save it to L4-1-5.pcapng. Using the trace file, explain how traceroute works in both cases (with and without the `-I` option). /2

The goal of traceroute is to get an overview of the route that a packet takes to get to a destination. It starts by sending a packet with a TTL of 1, causing the first-hop router to send an ICMP message back to the source. The TTL is incremented and this process is repeated until a 'Destination Unreachable (Port unreachable)' message is returned to the host. This response indicates that the end of the path was reached but that no application was running on the UDP port. We can also use ICMP messages instead of UDP. In this case, we simply wait for a ICMP ping reply instead of a 'Destination Unreachable (Port unreachable)' message.

Exercise 2: Routing loops

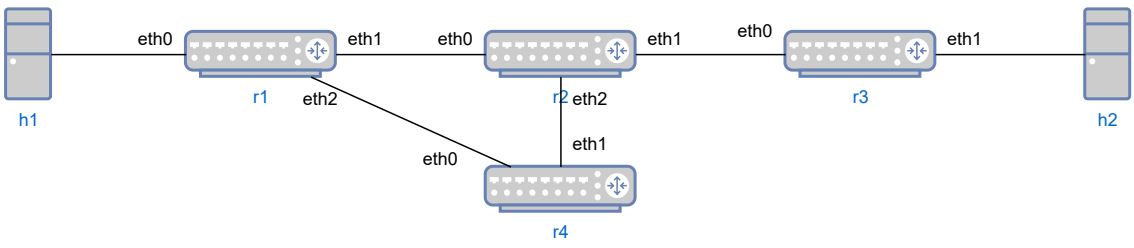


Figure 4.2: Network configuration for routing loops

Routing loops occur when a router that forwards a packet, receives the same packet later again, causing it to forward the same packet again (over the same path). This can happen again and again and is obviously a waste of resources. Moreover, the forwarded (looped) packet will never reach its intended destination. We will start from the topology shown in figure 4.2. As was the case in the previous exercise, write a python script that creates the following topology and **make sure no routing daemons are started by default!** Save it to `L4-2-1.py`.

1. Start your python script and add routes so that:

Host	eth0	eth1	eth2
h1	`fc00:0:0:1::1/64`	N/A	N/A
h2	`fc00:0:0:6::2/64`	N/A	N/A
r1	`fc00:0:0:1::11/64`	`fc00:0:0:2::11/64`	`fc00:0:0:5::11/64`
r2	`fc00:0:0:2::12/64`	`fc00:0:0:3::12/64`	`fc00:0:0:4::12/64`
r3	`fc00:0:0:3::13/64`	`fc00:0:0:6::13/64`	N/A
r4	`fc00:0:0:5::14/64`	`fc00:0:0:4::14/64`	N/A

Table 4.2: IP addresses for routing loops

- r1 will send traffic to the subnet of h2 to eth0 of r2.
- r2 will send traffic to the subnet of h2 to eth1 of r4.
- r4 will send traffic to the subnet of h2 to eth2 of r1.
- r2 and r4 have a route to h1 via r1.

2. Start a wireshark capture on eth0 of r2 and save it to `L4-2-1.pcapng`.
3. Start a ping command from h1 to h2. *Limit the amount of pings to just 1!*
4. After the ping, also send a `traceroute` from h1 to h2. Perform the traceroute commands with the `-q 1 -N 1 -z 100` options. Save the output to `L4-2-3.traceroute.txt`.

Use the data captured with Wireshark in L4-2-1.pcapng to answer the questions. Support your answers with the saved Wireshark data.

L4-2-1 In the trace file, identify two consecutive looped ping packets. Are they identical? If not, what is different? /1

We look at packet 1 and 2. This is effectively the same packet but with the Hop Limit header decreased by 3, since this packet passed through 3 routers in a loop.

L4-2-2 How many times does the looped ping packet appear in your trace file? Why doesn't it loop forever? /1

The packet appeared 21 times. It does not loop forever because of the IPv6 Hop Limit header. This causes a packet to get dropped after a certain number of hops, to prevent loops.

L4-2-3 Include your traceroute output in the answer below. In that output, you should clearly see the routing loop. Using this output and the packets in the trace file, explain what happens. /2

```
oot@computernetwerken:/home/computernetwerken/lab4/shared# traceroute -q 1 -N
1 -z 100 h2
traceroute to h2 (fc00:0:0:6::2), 30 hops max, 80 byte packets
 1  r1 (fc00:0:0:1::11)  0.052 ms
 2  r2 (fc00:0:0:2::12)  0.043 ms
 3  r4 (fc00:0:0:5::14)  0.023 ms
```

```
 7  4  r1 (fc00:0:0:1::11)  0.018 ms
    5  r2 (fc00:0:0:2::12)  0.022 ms
 9  6  r4 (fc00:0:0:5::14)  0.023 ms
    7  r1 (fc00:0:0:1::11)  0.032 ms
11  8  r2 (fc00:0:0:2::12)  0.031 ms
    9  r4 (fc00:0:0:5::14)  0.033 ms
13 10  r1 (fc00:0:0:1::11)  0.024 ms
   11  r2 (fc00:0:0:2::12)  0.028 ms
15 12  r4 (fc00:0:0:5::14)  0.028 ms
   13  r1 (fc00:0:0:1::11)  0.030 ms
17 14  r2 (fc00:0:0:2::12)  0.037 ms
   15  r4 (fc00:0:0:5::14)  0.037 ms
19 16  r1 (fc00:0:0:1::11)  0.030 ms
   17  r2 (fc00:0:0:2::12)  0.033 ms
21 18  r4 (fc00:0:0:5::14)  0.039 ms
   19  r1 (fc00:0:0:1::11)  0.043 ms
23 20  r2 (fc00:0:0:2::12)  0.034 ms
   21  r4 (fc00:0:0:5::14)  0.036 ms
25 22  r1 (fc00:0:0:1::11)  0.038 ms
   23  r2 (fc00:0:0:2::12)  0.038 ms
27 24  r4 (fc00:0:0:5::14)  0.049 ms
   25  r1 (fc00:0:0:1::11)  0.045 ms
29 26  r2 (fc00:0:0:2::12)  0.041 ms
   27  r4 (fc00:0:0:5::14)  0.047 ms
31 28  r1 (fc00:0:0:1::11)  0.038 ms
   29  r2 (fc00:0:0:2::12)  0.058 ms
33 30  r4 (fc00:0:0:5::14)  0.057 ms
```

traces/L4-2-3.traceroute.txt

First, the packet is forwarded to r1. Then, it will forward the packet in a loop from r1 to r2, to r4, and finally back to r1. This loop is caused by the forwarding rules we manually set.

4.2 Border Gateway Protocol

In the previous exercises, you learned how to configure routing table entries manually. This was referred to as static routing. The topic of the next exercises is dynamic routing, where dynamic routing protocols (from now on, called routing protocols) set the routing tables automatically without human intervention. Routers and hosts that run a routing protocol, exchange routing protocol messages related to network paths and node conditions, and use these messages to compute paths between routers and hosts.

Most routing protocols implement a shortest-path algorithm, which, for a given set of routers, determines the shortest paths between the routers. Some routing protocols allow that each network interface be assigned a cost metric. In this case, routing protocols compute paths with least cost. Based on the method used to compute the shortest or least-cost paths, one distinguishes distance vector and link state routing protocols.

In a link-state algorithm, the network topology and all link costs are known, that is, available as input to the LS algorithm. In practice, this is accomplished by having each node broadcast link-state packets to all other nodes in the network, with each link-state packet containing the identities and costs of its attached links. In practice (for example, with the Internet's OSPF routing protocol), this is often accomplished by a link-state broadcast algorithm. The result of the nodes' broadcast is that all nodes have an identical and complete view of the network. Each node can then run the LS algorithm and compute the same set of least-cost paths as every other node.

Whereas the LS algorithm is an algorithm using global information, the distance-vector (DV) algorithm is iterative, asynchronous, and distributed. It is distributed in that each node receives some information from one or more of its directly attached neighbours, performs a calculation, and then distributes the results of its calculation back to its neighbours. It is iterative in that this process continues on until no more information is exchanged between neighbours. The algorithm is asynchronous in that it does not require all of the nodes to operate in lockstep with each other.

The notion of an autonomous system (AS) is central to the understanding of routing protocols on the Internet. An autonomous system is a group of IP networks under the authority of a single administration, and the entire Internet is carved up into a large number of autonomous systems. A list of all AS's can be found on the BGP Looking Glass website. Examples of autonomous systems are *Belnet*, *Telenet*, *Proximus*, etc. Each autonomous system is assigned a globally unique identifier, called the AS number. On the Internet, dynamic routing within an autonomous system and between autonomous systems is handled by different types of routing protocols. A routing protocol that is concerned with routing within an autonomous system is called an intradomain routing protocol or interior gateway protocol (IGP). A routing protocol that determines routes between autonomous systems is called an interdomain routing protocol or exterior gateway protocol (EGP).

We will start with interdomain routing by having a look at the BGP protocol.

Exercise 3: BGP

In this exercise, things are a bit different. You will find a file called `bgp.py` as an extra to this lab. This file contains the setup for the following exercise. This setup has also *disabled* any standard OSPF daemons in mininet, but has added BGP daemons instead that are responsible for exchanging BGP routing information. Additionally, on `as1rb`, a wireshark trace is automatically started on all interfaces simultaneously when the script starts, so you will be capturing all traffic from the beginning. After the script is stopped, the trace file can be found in `/tmp/bgp_as1rb.pcapng`.

L4-3-1 Study the `bgp.py` file and make a drawing of the topology it creates. Also include a table with the interface names and IP addresses that are being used. Finally, since we are going to use BGP, on your drawing, indicate which router belongs to which AS. /1

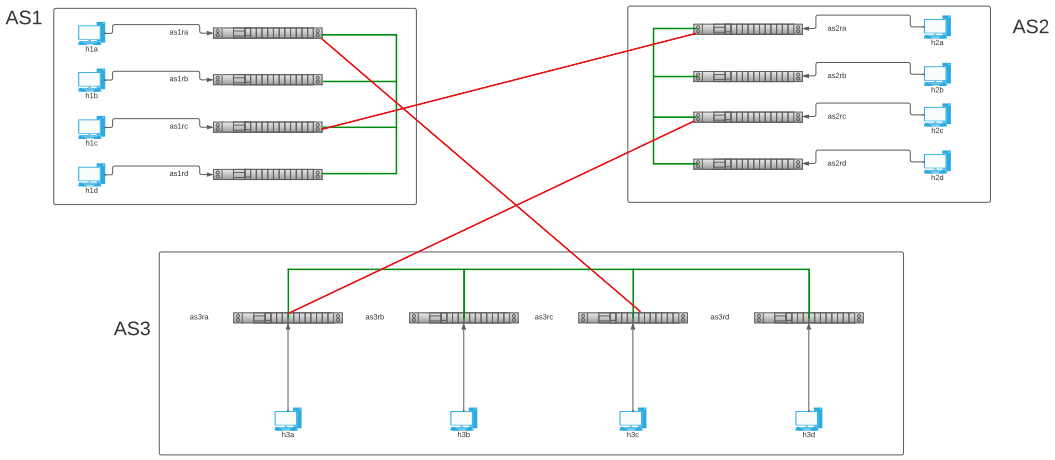


Figure 4.3: Network configuration

Node	Interface Name	IP Address
as1ra	as1ra-h1a	fc00:0:1:a::a/64
h1a	h1a-as1ra	fc00:0:1:a::1/64
as1rb	as1rb-h1b	fc00:0:1:b::b/64
h1b	h1b-as1rb	fc00:0:1:b::1/64
as1rc	as1rc-h1c	fc00:0:1:c::c/64
h1c	h1c-as1rc	fc00:0:1:c::1/64
as1rd	as1rd-h1d	fc00:0:1:d::d/64
h1d	h1d-as1rd	fc00:0:1:d::1/64
as2ra	as2ra-h2a	fc00:0:2:a::a/64
h2a	h2a-as2ra	fc00:0:2:a::2/64
as2rb	as2rb-h2b	fc00:0:2:b::b/64
h2b	h2b-as2rb	fc00:0:2:b::2/64
as2rc	as2rc-h2c	fc00:0:2:c::c/64
h2c	h2c-as2rc	fc00:0:2:c::2/64
as2rd	as2rd-h2d	fc00:0:2:d::d/64
h2d	h2d-as2rd	fc00:0:2:d::2/64
as3ra	as3ra-h3a	fc00:0:3:a::a/64
h3a	h3a-as3ra	fc00:0:3:a::3/64
as3rb	as3rb-h3b	fc00:0:3:b::b/64
h3b	h3b-as3rb	fc00:0:3:b::3/64
as3rc	as3rc-h3c	fc00:0:3:c::c/64
h3c	h3c-as3rc	fc00:0:3:c::3/64
as3rd	as3rd-h3d	fc00:0:3:d::d/64
h3d	h3d-as3rd	fc00:0:3:d::3/64

Table 4.3: IP addresses for hosts attached to routers

In the diagram, a red line represents a inter-AS link, whereas a green line represents one intra-AS connection for each two nodes connected to the line.

Node	Interface Name	IP Address
as1ra	as1ra-as1rb	fc00:0:1:ab::a/64
as1rb	as1rb-as1ra	fc00:0:1:ab::b/64
as1ra	as1ra-as1rc	fc00:0:1:ac::a/64
as1rc	as1rc-as1ra	fc00:0:1:ac::c/64
as1ra	as1ra-as1rd	fc00:0:1:ad::a/64
as1rd	as1rd-as1ra	fc00:0:1:ad::d/64
as1rb	as1rb-as1rc	fc00:0:1:bc::b/64
as1rc	as1rc-as1rb	fc00:0:1:bc::c/64
as1rb	as1rb-as1rd	fc00:0:1:bd::b/64
as1rd	as1rd-as1rb	fc00:0:1:bd::d/64
as1rc	as1rc-as1rd	fc00:0:1:cd::c/64
as1rd	as1rd-as1rc	fc00:0:1:cd::d/64
as2ra	as2ra-as2rb	fc00:0:2:ab::a/64
as2rb	as2rb-as2ra	fc00:0:2:ab::b/64
as2ra	as2ra-as2rc	fc00:0:2:ac::a/64
as2rc	as2rc-as2ra	fc00:0:2:ac::c/64
as2ra	as2ra-as2rd	fc00:0:2:ad::a/64
as2rd	as2rd-as2ra	fc00:0:2:ad::d/64
as2rb	as2rb-as2rc	fc00:0:2:bc::b/64
as2rc	as2rc-as2rb	fc00:0:2:bc::c/64
as2rb	as2rb-as2rd	fc00:0:2:bd::b/64
as2rd	as2rd-as2rb	fc00:0:2:bd::d/64
as2rc	as2rc-as2rd	fc00:0:2:cd::c/64
as2rd	as2rd-as2rc	fc00:0:2:cd::d/64
as3ra	as3ra-as3rb	fc00:0:3:ab::a/64
as3rb	as3rb-as3ra	fc00:0:3:ab::b/64
as3ra	as3ra-as3rc	fc00:0:3:ac::a/64
as3rc	as3rc-as3ra	fc00:0:3:ac::c/64
as3ra	as3ra-as3rd	fc00:0:3:ad::a/64
as3rd	as3rd-as3ra	fc00:0:3:ad::d/64
as3rb	as3rb-as3rc	fc00:0:3:bc::b/64
as3rc	as3rc-as3rb	fc00:0:3:bc::c/64
as3rb	as3rb-as3rd	fc00:0:3:bd::b/64
as3rd	as3rd-as3rb	fc00:0:3:bd::d/64
as3rc	as3rc-as3rd	fc00:0:3:cd::c/64
as3rd	as3rd-as3rc	fc00:0:3:cd::d/64

Table 4.4: IP addresses for intra-AS links

Node	Interface Name	IP Address
as1rc	as1rc-as2ra	fc00:12::c/64
as2ra	as2ra-as1rc	fc00:12::a/64
as2rc	as2rc-as3ra	fc00:23::c/64
as3ra	as3ra-as2rc	fc00:23::a/64
as3rc	as3rc-as1ra	fc00:13::c/64
as1ra	as1ra-as3rc	fc00:13::a/64

Table 4.5: IP addresses for intra-AS links

1. Start the `bgp.py` topology in mininet.
2. Wait a few seconds for the topology to settle. When `pingall` reports a 100% success rate, you can continue with the next step.
3. From h1b, issue a traceroute to h2b. Save the output to `L4-3-3.traceroute.txt`.
4. Bring the `eth3` interface of as1rc down using `ifconfig`.
5. Perform a traceroute again from h1b to h2b and add the output to `L4-3-3.traceroute.txt`.
6. Bring the `eth3` interface of `as1rc` back up. **Do not forget to configure the IP address of that interface again, as it will have been removed from that interface!**
7. Again, perform a traceroute from h1b to h2b and add the output to `L4-3-3.traceroute.txt`.
8. Stop the script. Rename the wireshark trace found in `/tmp` to `L4-3-2.as1rb.pcapng`.

Use the data captured with Wireshark in L4-3-2.as1rb.pcapng to answer the questions. Support your answers with the saved Wireshark data.

L4-3-2 Open the wireshark trace file and take a look at the BGP packets that were captured. Which types of BGP messages did you capture and what is their purpose? Refer to packet numbers in the trace for examples of those messages. /2

The following types of BGP messages were identified:

1. OPEN Message
2. KEEPALIVE Message
3. UPDATE Message
4. ROUTE-REFRESH

OPEN Message

One example of an OPEN packet in our trace is packet number 20. Each router within an autonomous system sends an OPEN message to every other router in that autonomous system, forming a full mesh. These messages use TCP as its underlying transport protocol and are meant to establish a connection such that routers can exchange reachability information.

KEEPALIVE Message

One example of a KEEPALIVE packet in our trace is packet number 52. These messages are sent periodically to verify that the peer router is still operational. A KEEPALIVE message gets a plain TCP response, as can be seen in packet number 53.

UPDATE Message

One example of a UPDATE packet in our trace is packet number 55. These also receive a plain TCP response to confirm the peer has received the message. These messages are only sent in response to events on the network. For example, if a link goes down, an UPDATE message will be sent to update reachability information.

ROUTE-REFRESH

One example of a ROUTE-REFRESH packet in our trace is packet number 68. These are used to request peers to send what routes they have for a certain IP address.

L4-3-3 How did the route change when you brougth the interface down? How did it change when you brought it back up again? /1

Initially, the route was as follows:

1. as1rb: this is the only router attached to h1b
2. as1rc: AS1's border gateway router with AS2
3. as2ra: AS2's border gateway router with AS1
4. as2rb: this is the only router attached to h2b
5. h2b: the destination

After bringing the interface down, the route changed to:

1. as1rb: this is the only router attached to h1b
2. as1ra: AS1's border gateway router with AS3
3. as3rc: AS3's border gateway router with AS1
4. as3ra: AS3's border gateway router with AS2
5. as2rc: AS2's border gateway router with AS3
6. as2rb: this is the only router attached to h2b
7. h2b: the destination

From the traceroute, it can be seen that the traffic is routed through AS3, since a necessary interface in the border gateway router between AS1 and AS2 was brought down. After bringing the interface back up, the original route is restored.

L4-3-4 Why did BGP decide upon the original route (before the interface was brought down)? /1

BGP uses a route selection algorithm to determine the route to take. If a preference value local to the the router is set, the route with the highest of such values is chosen. If no routes have this value set, the path with the shortest AS path length is chosen. This rules out passing through AS3. All paths that are left all have the same next-hop value. We thus forward the packet to the next-hop router, which is as1rc in our case. Since this is intra-AS routing, an IGP takes care of this.

L4-3-5 Explain *in detail* using the BGP packets in your wireshark trace how `as1rb` knows how to change the routes so h1b and h2b can continue to communicate (when the link is taken down and again when it is brought back up). Explain the different fields in the BGP messages that are used to distribute the necessary information and explain their meaning. /5

In packet 408, we can see router AS1RC withdrawing its routes for all subnets in AS2, since the link to AS2 went down. Since router AS1RA also receives this update (withdrawal) message, it is able to compute that the routes it has for AS2 are now the preferred route to AS2. AS1RA thus sends an UPDATE message (packet number 410) to AS1RB, which previously has its route to subnets in AS2 withdrawn. AS1RB now has the routes passing through AS3 set.

For the update message, the following specific headers are used:

1. Withdrawn Routes: a list of prefixes to be withdrawn
2. Address family identifier: to distinguish between IPv4 and IPv6 v6 addresses
3. Next Hop: the IP address of the next-hop router that will forward the packets towards the destination
4. Path Attributes: a list of attributes that provide additional information about the path to the destination, such as the AS Path, the Local Preference, the Multi-Exit Discriminator (MED), and others.

In this case, when AS1RC withdraws its routes for subnets in AS2, it sends an UPDATE message with the Withdrawn Routes header containing the prefixes for the subnets that are no longer reachable. The Address Family Identifier header indicates the type of address (IPv4 in this case). The Next Hop header is set to the IP address of AS1RC, since it is the next-hop router for those prefixes. When AS1RA sends an UPDATE message to AS1RB, it includes the Path Attributes header with the AS Path attribute, which indicates the sequence of ASes that the route has traversed. In this case, the AS Path includes AS1RA and AS3, since the route is now passing through AS3 to reach AS2.

4.3 Open Shortest Path First

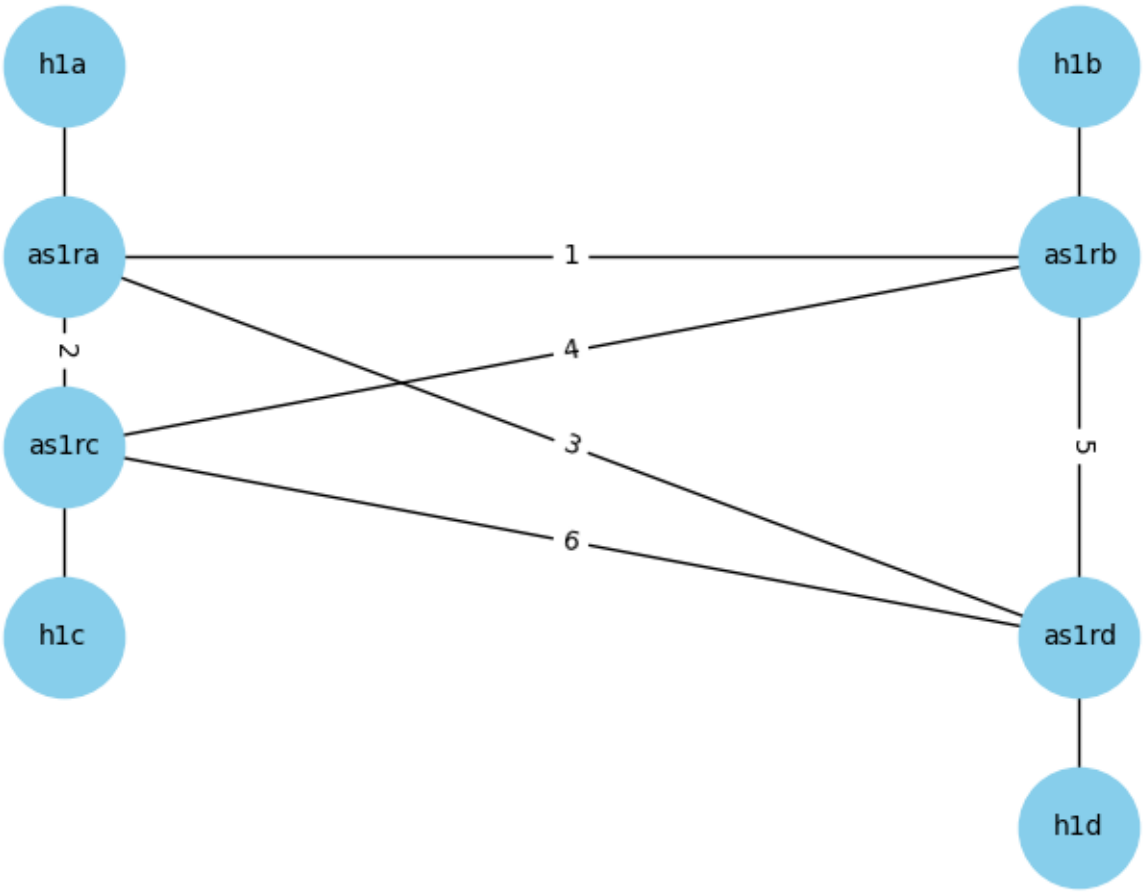
In this exercise, you explore the Open Shortest Path First (OSPF) routing protocol. OSPF is a link state routing protocol, where each router sends information on the cost metric of its network interfaces to all other routers in the network. The information about the interfaces is sent in messages that are called link state advertisements (LSAs). LSAs are disseminated using flooding, that is, a router sends its LSAs to all its neighbours, which, in turn, forward the LSAs to their neighbours, and so on. However, each LSA is forwarded only once. Each router maintains a link state database of all received LSAs, which provides the router with complete information about the topology of the network. Routers use their link state databases to run a shortest path algorithm that computes the shortest paths in the network.

OSPF is the most important link state routing protocol on the Internet. The functionality of OSPF is rich, and the lab exercises highlight only a small portion of the OSPF protocol. In this exercise, we use OSPF version 3 (OSPFv3).

Exercise 4: OSPF

You once again can find a pre-existing topology for the OSPF setup in `ospf.py`. This topology is basically a subset of the topology in the previous exercise. AS number 1 is extracted from that topology, and all the routers in that topology are now configured to run OSPF. Similar to the previous exercise, when you start up the script, a wireshark capture logging all traffic on all interfaces of as1ra is automatically started. It is saved to `/tmp/ospf_as1ra.pcapng`.

L4-4-1 Study the `ospf.py` file and make a drawing of the topology it creates. Also include a table with the interface names and IP addresses that are being used. Finally, since we are going to use OSPF, on your drawing, indicate which weights (called `igp_metric` in the script) are configured on each link. /1



Router	Interface Name	IP Address
as1ra	as1ra-as1rb	fc00:0:1:ab::a/64
as1ra	as1ra-as1rc	fc00:0:1:ac::a/64
as1ra	as1ra-as1rd	fc00:0:1:ad::a/64
as1ra	as1ra-h1a	fc00:0:1:a::a/64
as1rb	as1rb-as1ra	fc00:0:1:ab::b/64
as1rb	as1rb-as1rc	fc00:0:1:bc::b/64
as1rb	as1rb-as1rd	fc00:0:1:bd::b/64
as1rb	as1rb-h1b	fc00:0:1:b::b/64
as1rc	as1rc-as1ra	fc00:0:1:ac::c/64
as1rc	as1rc-as1rb	fc00:0:1:bc::c/64
as1rc	as1rc-as1rd	fc00:0:1:cd::c/64
as1rc	as1rc-h1c	fc00:0:1:c::c/64
as1rd	as1rd-as1ra	fc00:0:1:ad::d/64
as1rd	as1rd-as1rb	fc00:0:1:bd::d/64
as1rd	as1rd-as1rc	fc00:0:1:cd::d/64
as1rd	as1rd-h1d	fc00:0:1:d::d/64

Table 4.6: Interface names and IP addresses sorted by router type

---

1. Start the `ospf.py` script.

2. Because of the topology and configuration choices in the script, it takes a while before all routes are ready. After about 30 seconds, this should be the case. Check that a `pingall` command reports a 100% success rate before continuing.
3. On h1a, start three pings (one to each of h1b, h1c and h1d). Don't set any parameters, but just let the pings continue to run. Save the output of the ping commands to `L4-4-1.ping-h1X.txt`, where X is b, c or d, respectively.
4. From h1a, issue a `traceroute` to all other hosts (just the hosts, not the routers). Do the same from host h1d. Save the output of both traceroutes to `L4-4-1.traceroute.1.txt`.
5. Bring down the link between as1ra and as1rc. **Do this on as1rc as it will otherwise interfere with the wireshark capture that is running on as1ra!**
6. Do the same traceroutes as above, and save the output to `L4-4-1.traceroute.2.txt`.
7. Now, also bring down the link between as1ra and as1rb and between as1rb and as1rd.
8. Again perform the same traceroutes, and save the output to `L4-4-1.traceroute.3.txt`.
9. Bring all the interfaces back up one by one **and don't forget to set the IP address on them again!** After bringing the interfaces up, monitor the ping commands until you notice the routes have changed and settled.
10. Finally, verify that, now all links are back up, your traceroutes yield the same results as in the beginning of the exercise.
11. Stop the pings and the mininet script. Rename the wireshark capture to `L4-4-1.pcapng`.

Use the data captured with Wireshark in L4-4-1.pcapng to answer the questions. Support your answers with the saved Wireshark data.

L4-4-2 Explain the routes you obtained when you did the first set of `traceroutes`. Why are the routes what they are? /1

For the traceroute output from h1a:

1. h1a → h1b: h1a (as1ra) → as1rb → h1b. This route has a total cost of 1 (as the link between as1ra and as1rb has an IGP metric of 1).
2. h1a → h1c: h1a (as1ra) → as1rc → h1c. This route has a total cost of 2 (as the link between as1ra and as1rc has an IGP metric of 2).
3. h1a → h1d: h1a (as1ra) → as1rd → h1d. This route has a total cost of 3 (as the link between as1ra and as1rd has an IGP metric of 3).

For the traceroute output from h1d:

1. h1d → h1a: h1d (as1rd) → as1ra → h1a. This route has a total cost of 3 (as the link between as1rd and as1ra has an IGP metric of 3).
2. h1d → h1b: h1d (as1rd) → as1ra → as1rb → h1b. This route has a total cost of 4 (3 from as1rd to as1ra and 1 from as1ra to as1rb).

3. h1d → h1c: h1d (as1rd) → as1ra → as1rc → h1c. This route has a total cost of 5 (3 from as1rd to as1ra and 2 from as1ra to as1rc).

These routes are chosen because they have the lowest total cost based on the IGP metric, which is the primary factor that OSPF uses to determine the shortest path.

L4-4-3 After you brought down the first interface, which routes changed and what are the new routes? Focus only on the routes starting at h1a and h1d, and refer to your traceroute output. /1

After bringing down the first interface (between as1ra and as1rc), the route from h1a to h1c and the route from h1d to h1c changed. Here are the new routes:

From h1a to h1c:

1. h1a → as1ra (fc00:0:1:a::a)
2. as1ra → as1rb (fc00:0:1:ab::b)
3. as1rb → as1rc (fc00:0:1:bc::c)
4. as1rc → h1c (fc00:0:1:c::1)

From h1d to h1c:

1. h1d → as1rd (fc00:0:1:d::d)
2. as1rd → as1rc (fc00:0:1:cd::c)
3. as1rc → h1c (fc00:0:1:c::1)

The reason for the route change is that the link between as1ra and as1rc was brought down. As a result, the shortest path from h1a to h1c now goes through as1rb, and the shortest path from h1d to h1c doesn't change, but the link between as1rd and as1rc is used instead. The other routes remain the same as before because they were not affected by the change in the network topology.

L4-4-4 When you brought down the first interface, you should have seen OSPF packets in your trace file. Which packets are those and what is their purpose? /1

Packet 830 & 831: OSPF LS Update packets - These packets are used to share link-state information with OSPF neighbors. In this case, the LS Update packets are sent due to the change in the network topology when the first interface was brought down.

Packet 846 & 847: OSPF LS Acknowledge packets - These packets acknowledge the receipt of LS Update packets, confirming that the link-state information has been successfully received.

These OSPF packets indicate that the OSPF routers in the network are detecting the change in the topology and reacting to it by updating their link-state databases and routing tables.

L4-4-5 Explain *in detail* using the OSPF packets in your wireshark trace how the routers know how to change the routes so all hosts can continue to communicate. Explain the different fields in the OSPF packets that are used to distribute the necessary information and explain their meaning.

/3

1. OSPF Hello Packets (Packets 822, 823, 840, and 841, but there are many more):
   Hello packets are used to discover OSPF neighbors and maintain adjacencies between them. These packets contain the following fields in our case:
   1. Interface ID: A unique identifier for the OSPF-enabled interface on the router.
   2. Router Priority: A value used to determine the designated router (DR) and backup designated router (BDR) on a broadcast network segment.
   3. Options: Flags that indicate optional OSPF capabilities supported by the router (R: Router, E: External Routing, V6: OSPF for IPv6).
   4. Hello Interval: The time interval (in seconds) between the periodic Hello packets sent by the router on the OSPF interface.
   5. Router Dead Interval: The time interval (in seconds) for which a neighbor must receive a Hello packet before considering the neighbor as down.
   6. Designated Router: The router ID of the designated router on the network segment.
   7. Backup Designated Router: The router ID of the backup designated router on the network segment.
   8. Active Neighbor: The router IDs of the neighbors with which the router has established an adjacency.
2. OSPF LS Update Packets (Packets 830 and 831):
   LS Update packets are used to share link-state information between OSPF routers. These packets contain one or more Link-State Advertisements (LSAs) that describe the router's local links and their associated costs. The main fields in an LSA include:
   1. Number of LSAs: The number of Link-State Advertisements (LSAs) contained in the LS Update packet.
   2. LS Age: The time (in seconds) since the LSA was originated.
   3. Do Not Age: A flag indicating whether the LSA should age or not.
   4. LS Type: The type of LSA (e.g., 0x2001 for Router-LSA).
   5. Link State ID: A unique identifier for the LSA.
   6. Advertising Router: The router ID of the router that originated the LSA.
   7. Sequence Number: A number that increments with each new instance of the LSA to detect old or duplicate LSAs.
   8. Checksum: A checksum used for error detection.
   9. Length: The length of the LSA, including the LSA header.
   10. Flags: Flags specific to the LSA type.
   11. Options: Flags that indicate optional OSPF capabilities supported by the router (same as in the Hello packet).

3. OSPF LS Acknowledge Packets (Packets 846 and 847):
   LS Acknowledge packets are sent in response to received LS Update packets, confirming that the recipient router has received and processed the LSAs. These packets contain the same LSA headers as in the corresponding LS Update packets.

L4-4-6 When you brougth the interfaces back up one by one, did the routes change again? Why? How do the routers decide to change their routes? /2

The routers use OSPF to exchange routing information with their neighbors by sending OSPF packets. When a link is established, routers exchange Hello packets to discover neighbors and form adjacencies. After the adjacency is formed, routers exchange their LSDBs, DB Description packets, LS Request packets, LS Update packets, and Link-LS Acknowledge packets.

When the link between as1rb and as1rd is established again, we can see from the Wireshark traces that the routers exchange DB Description, LS Request, LS Update, and LS Acknowledge packets (lines 1877-1891, 1905, and 1906). This exchange allows the routers to update their LSDBs, which contain information about the network's topology.

Similarly, when the link between as1ra and as1rb is established again, we can see the routers exchange LS Update and LS Acknowledge packets (lines 2667 and 2681).

When the link between as1ra and as1rc is established again, we can also see the routers exchange LS Update and LS Acknowledge packets (lines 2761-2762, 2770, and 2771).

L4-4-7 Did it take longer or shorter for the routes to change when the interfaces were brought back up than when they were brought down? Why? /1

When the interfaces were brought down, the routes changed faster because OSPF routers quickly detect the loss of a neighbor and the associated link due to the Hello protocol. Routers send Hello packets at regular intervals specified by the Hello Interval parameter (5 seconds) to their neighbors to maintain adjacencies. If a router fails to receive a Hello packet from a neighbor within the specified dead interval (20 seconds), it considers the neighbor and the link as "down."

As soon as the link is considered down, the router generates a new LSA to inform other routers about the topology change. LSAs are flooded throughout the OSPF domain, and upon receiving the updated LSAs, other routers update their LSDBs and recalculate the shortest paths using the SPF algorithm. This process happens relatively quickly, leading to faster route changes when the interface was brought down.

When the interfaces are brought back up, the process of re-establishing OSPF adjacencies and exchanging routing information takes more time:

1. Routers send Hello packets to discover neighbors and establish adjacencies. This process depends on the hello_int parameter, which defines the interval between Hello packets. The time it takes to form an adjacency depends on how quickly the routers exchange these packets and the dead interval configured.

2. After adjacencies are established, routers exchange their LSDBs using DB Description, LS Request, LS Update, and LS Acknowledge packets. This process may took some time.
3. Once the LSDBs are updated, routers run the SPF algorithm to calculate new shortest paths and update their routing tables. The time required for this calculation depends on the size and complexity of the network topology.

It takes longer for the routes to change when the interfaces are brought back up because the process of re-establishing OSPF adjacencies, exchanging routing information, and updating LSDBs is more time-consuming than detecting a lost neighbor and flooding the updated LSAs when a link goes down.

4.4 BGP and OSPF Combined

Exercise 5: Bringing everything together

Now that you have configured BGP for exchanging inter-AS routing information, and OSPF for intra-AS routing, it is time to bring everything together. Starting from the `bgp.py` and `ospf.py` scripts, you need to create a new script called `bgp_ospf.py`, that satisfies the following requirements:

1. The topology in the script is the same topology as in the `bgp.py` script, except that, in every AS, the direct link between `asXra` and `asXrc`, and the direct link between `asXrb` and `asXrd` are **removed**.
2. The BGP configuration remains the same as in the BGP exercise.
3. Within each AS, OSPF is used to exchange routing information.
4. There is **no OSPF traffic** between the different AS.

 Consult the ipmininet documentation on how to disable OSPF on an interface.

Once you have created the specified topology, perform the following steps:

1. Start the `bgp_ospf.py script`.
2. Wait for the topology to settle. If everything is configured correctly, a `pingall` command should report a 100% success rate.
3. Create a scenario involving *at least* traceroute commands to be able to identify the route a packet takes from a certain host to another host. The scenario should cover the following:
   - You should have a ping/ssh/iperf/... connection between two hosts in a different AS. (When using iperf, you should make sure any trace files you make do not get too big by limiting the iperf bandwidth and/or the link bandwidth in your topology.)
   - Start up wireshark traces on a relevant interface that will contain BGP traffic, as wel as on an interface that will contain OSPF traffic. *Do not forget to include these traces as attachment to this lab report!*.
   - At every step of the scenario, check the actual route between the two hosts with `traceroute`.
   - You should break the link on the initial path between the two AS systems and check how the routes are changed.
   - You should also break *at least one link* within the same AS that is on the path between both hosts and check how the routes are changed.
   - You should also re-enable the links you brought down and check how the routes are changed.
   - Whenever bringing down links, verify that at all times a valid path exists between the two hosts you are using in your scenario.

   L4-5-1 Make a drawing of your final topology. Include any details such as AS numbers, link weights, etc. Include all relevant output as extra files and mention them in your answer (traceroute output, ping output, iperf output, ...). Also, the wireshark captures should be

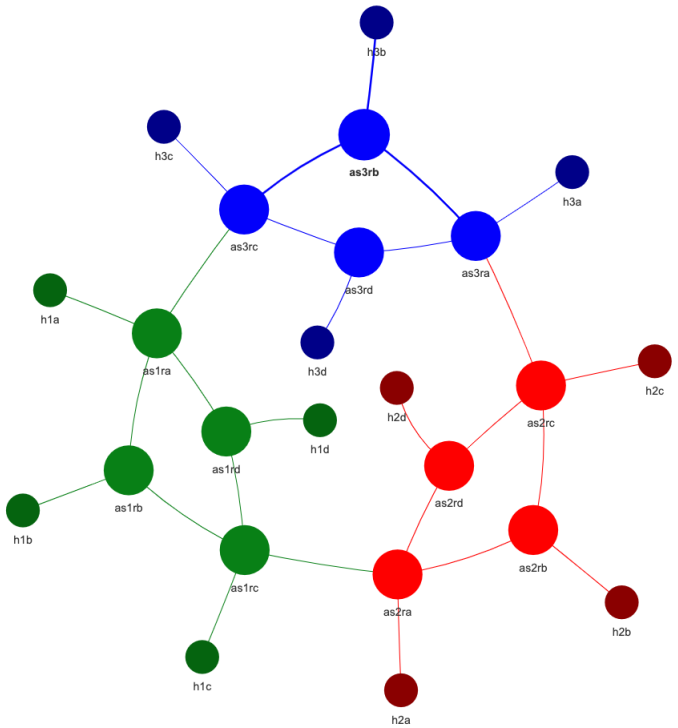
included. Clearly mention which capture was made on which interface. Explain *in detail* the different steps in your scenario, and for every step, explain how and when the topology changes. Things to take into consideration:

- Does the topology change immediately, or does it take a while? Why?
- Is there any packet loss? Why?
- How do the routers know how to change the topology?
- When the topology is changed (e.g. a link is brought up or down): how far does this information travel (i.e. which routers are informed)? Why?

The above questions apply both when taking down the interfaces, as well as when you bring them back up! /15

Part 1. Original Network Topology

This is the original network topology:



For our scenario, we are checking how the traceroute changes from h1d to h2d. We have set the IGP metric for all intra-AS links to 1 for convenience.

Part 2. Ping from h1d to h2d

First, we set up a ping from h1d to h2d so that we can monitor for packet loss. Can be found in traces/ping-h1d-h2d.txt.

Part 3. Monitoring OSPF and BGP packets

Second, we monitor for OSPF packets on as1rd and BGP packets on as1rc using Wireshark. traces/ospf.pcapng and traces/bgp.pcap.

Part 4. Initial Traceroute

Third, we perform our first traceroute and we obtain the following route:

```
    traceroute to fc00:0:2:d::2 (fc00:0:2:d::2), 30 hops max, 80 byte packets
 1  as1rd (fc00:0:1:d::d)  0.056 ms  0.011 ms  0.032 ms
 2  as1rc (fc00:0:1:bc::c)  0.065 ms  0.026 ms  0.014 ms
 3  as2ra (fc00:12::a)  0.042 ms  0.020 ms  0.020 ms
 4  as2rd (fc00:0:2:ad::d)  0.034 ms  0.029 ms  0.024 ms
 5  h2d (fc00:0:2:d::2)  0.035 ms  0.029 ms  0.062 ms
```

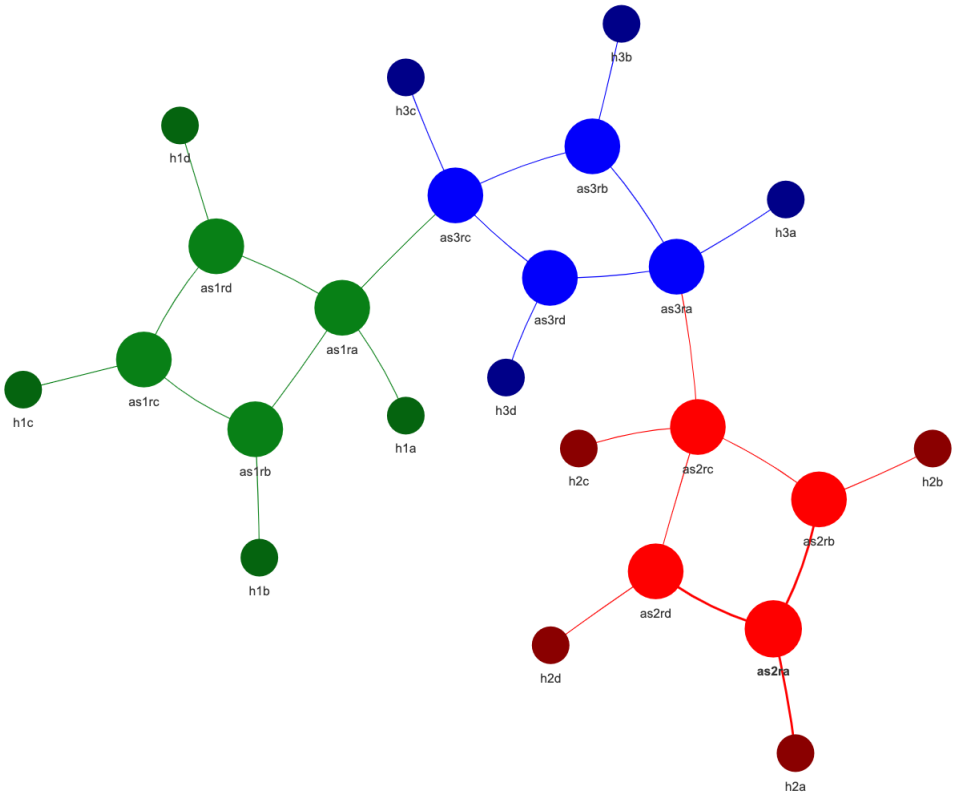
This makes sense, it is indeed the shortest path if we take a look at our first graph.

Part 5. Bringing Down the Inter-AS Link

Fourth, we go to as2ra and bring down the inter-AS link with as1rc. We notice in our BGP trace that there is an UPDATE message, letting us know that the network is updating the reachability information. When we check the traceroute, we get the following route:

```
traceroute to fc00:0:2:d::2 (fc00:0:2:d::2), 30 hops max, 80 byte packets
1  as1rd (fc00:0:1:d::d)  0.057 ms  0.011 ms  0.065 ms
2  as1ra (fc00:0:1:ad::a)  0.075 ms  0.029 ms  0.044 ms
3  as3rc (fc00:13::c)  0.072 ms  0.034 ms  0.031 ms
4  as3rb (fc00:0:3:bc::b)  0.071 ms  0.050 ms  0.024 ms
5  as3ra (fc00:0:3:ab::a)  0.073 ms  0.041 ms  0.034 ms
6  as2rc (fc00:23::c)  0.067 ms  0.228 ms  0.052 ms
7  as2rd (fc00:0:2:cd::d)  0.075 ms  0.048 ms  0.044 ms
8  h2d (fc00:0:2:d::2)  0.166 ms  0.060 ms  0.046 ms
```

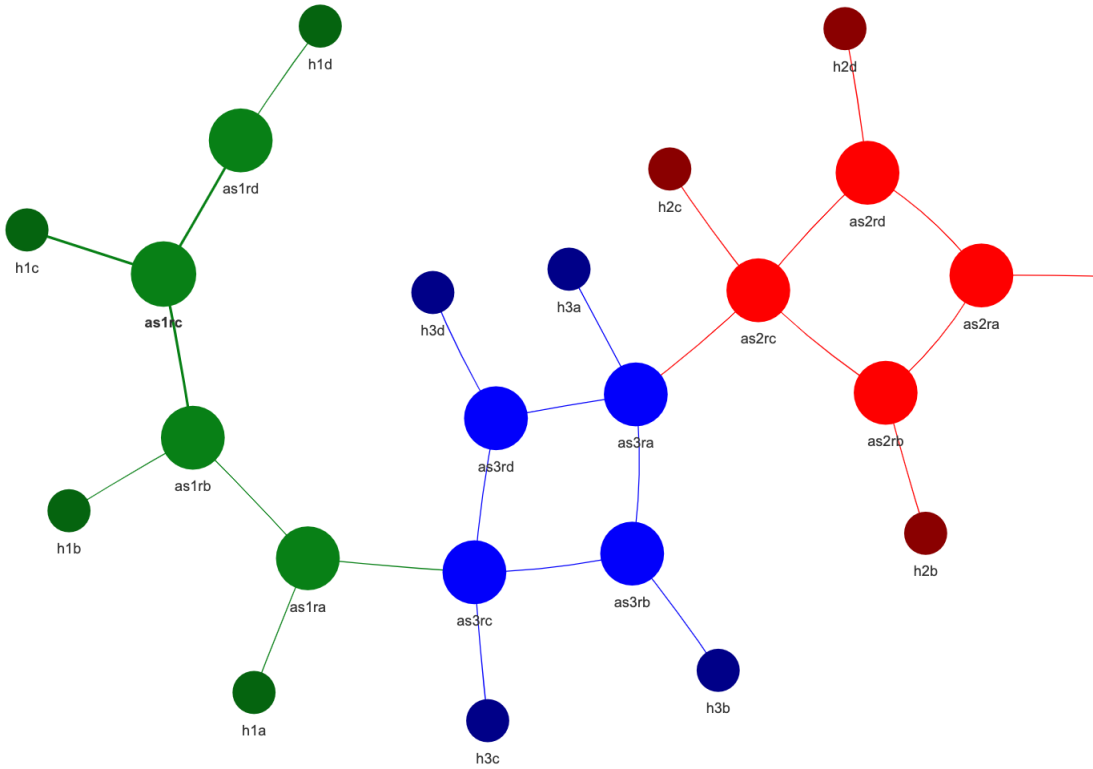
Which makes sense if we look at the following visualization of the network after bringing the link down:



Part 6. Bringing Down the Intra-AS Link

Fifth, we go to as1ra and bring down the link with as1rd. This time, this is an intra-AS link, so we quickly see in our OSPF trace that there are LS update packets being sent, which is shortly followed by LS acknowledge packets, letting us know that the new link-state information has been successfully received.

The resulting topology and subsequent traceroute look like this:




```
traceroute to fc00:0:2:d::2 (fc00:0:2:d::2), 30 hops max, 80 byte packets
1  as1rd (fc00:0:1:d::d)  0.057 ms  0.011 ms  0.065 ms
2  as1rc (fc00:0:1:c::d)  0.074 ms  0.011 ms  0.065 ms
3  as1rb (fc00:0:1:b::d)  0.067 ms  0.011 ms  0.065 ms
4  as1ra (fc00:0:1:ad::a)  0.070 ms  0.029 ms  0.044 ms
5  as3rc (fc00:13::c)  0.072 ms  0.034 ms  0.031 ms
6  as3rb (fc00:0:3:bc::b)  0.071 ms  0.050 ms  0.024 ms
7  as3ra (fc00:0:3:ab::a)  0.073 ms  0.041 ms  0.034 ms
8  as2rc (fc00:23::c)  0.067 ms  0.228 ms  0.052 ms
9  as2rd (fc00:0:2:cd::d)  0.075 ms  0.048 ms  0.044 ms
10  h2d (fc00:0:2:d::2)  0.166 ms  0.060 ms  0.046 ms
```

Part 7. Restoring Both Links

Finally, we bring both links back up. This time, it took a couple of minutes for the link states to be updated again since it takes longer to re-establish network adjacencies than it is to notice that you have lost a neighbor. We get the following traceroute, which is the same as the original:

```
    traceroute to fc00:0:2:d::2 (fc00:0:2:d::2), 30 hops max, 80 byte packets
 1  as1rd (fc00:0:1:d::d)  0.056 ms  0.011 ms  0.032 ms
 2  as1rc (fc00:0:1:bc::c)  0.065 ms  0.026 ms  0.014 ms
 3  as2ra (fc00:12::a)  0.042 ms  0.020 ms  0.020 ms
 4  as2rd (fc00:0:2:ad::d)  0.034 ms  0.029 ms  0.024 ms
 5  h2d (fc00:0:2:d::2)  0.035 ms  0.029 ms  0.062 ms
```

Acronyms

IP Internet Protocol

IPv6 IP version 6

OSPF Open Shortest Path First

BGP Border Gateway Protocol