Department of Computer Science
and Software Engineering

COSC 364
Internet Routing Protocols

Andreas Willig
andreas.willig@canterbury.ac.nz

Version: 1.0 (February 1, 2021)

Contents

Chapter 1

Administrative Matters

1.1 How to use the Booklets

The course COSC 364 will be sub-divided into a number of modules, these are:

- Module 0: Course organization [0.5 weeks]
- Module 1 - **IPv4 and Routers**: IPv4 refresher, router architectures [1.5 weeks]
- Module 2 - **Routing**: Internet routing (AS, OSPF, BGP) [4 weeks]
- Module 3 - **Planning**: Optimization, flow and network planning [5 weeks]

For the last two modules (Routing and Planning) there are separate **booklets**. The booklets are designed to contain **all** material relevant for the respective module. This includes as a minimum the relevant theory, lab materials, and a set of problems and review questions.

The booklets play a key role in the course:

- It is essential that you **read the booklet before the module starts**. In particular, you need to read (and understand!) the theory part of it and make a first attempt to work through the problems and review questions. We will **not** go through the entire theory in class. You also need to read the lab materials before you start working in the lab. Some preparatory lab problems should be done **before** the lab.
- We will not go through the theory during the lectures in detail. Instead we will use the lectures to do one or more of the following:
  - Review selected items from the theoretical part.
  - Discuss selected issues in some more depth.
  - Work through examples and selected exercises / review questions.
  - Discuss your questions and provide clarifications.

This is the third year the booklets are available, and therefore they are probably still a bit rough around the edges. All errors are mine, but I really would appreciate if you can point out any mistakes, typos, omissions, unclear points and the like. The booklets will be versioned, the version is shown on the cover page. From time to time I will publish new versions in which a bulk of errors have been corrected and some parts may have been revised/added. When writing me about an error, please let me know the version you are referring to and the relevant page numbers (if any).

1.2 General Questions and Answers

Question: Why are there three different types of problems?

Answer:

In the text you will find three different types of problems:

- Normal problems (or just problems) are part of the main text and need to be worked on by you. Some of these will be discussed in the lectures, others will not. All of these problems (or variations thereof) can occur in a test or exam.
- Bonus problems are also part of the main text, but you do not need to do them and they will not occur in a test or exam.
- Review problems occur typically in the labs and typically ask you to read some bits of documentation or a `man` page. You need to do these problems but they will not occur in a test or exam.

These types of problems are marked differently in the text (see also Section 1.3).

Question: What parts of the theory can come up in the exam/test?

Answer:

Everything from the theory part that is not marked as optional / excursion, including the problems (but not the bonus problems or review problems) and examples. The contents of the appendices will normally not be covered directly in the exam, but they provide pre-requisite material that is needed to understand the contents and which possibly not all people have on top of their head.

Question: What parts of the labs can come up in the exam/test?

Answer:

Again, more or less everything that is not marked as optional / excursion, including examples and most of the problems, but not the review problems.

Question: Will the exam/test be completely based on the booklets?

Answer:

You should not expect that **all** the problems in the exam/test will have shown up in the booklets directly or in a similar way. Many of the questions actually will, but some percentage of exam/test questions will also require some transfer of concepts or even creative thinking.

1.3 Typographic Conventions

Problems are an integral part of the text and are relevant for tests / exams. They appear like this:

Problem 1.3.1 (A not-so-hard problem)**.**
Disprove that all graphs are 3-colorable.

A particular type of problem are review problems. You need to do these as well (because they usually ask you to familiarize yourself with some important material), but they will not be asked in the exam. A classical example problem of this type is to read a certain `man` page. Review problems look like this:

Problem 1.3.2 (Read tons of `man` pages)**.**
Read the `man` pages for the following commands: `find`, `ls`, `grep`, `awk`, `sed`, ...

There are also bonus problems, which, while interesting, are not important for the main part and will not feature in tests / exams. They look like this:

Problem 1.3.3 (A harder problem)**.**
Prove that $P \neq NP$.

Sometimes I will add information that is not central to the ongoing discussion. Such an excursion (or digression) looks as follows:

Excursion (Dramatic food crisis)
A bag of rice has fallen over in China. Film at 11.

and can be completely ignored.

1.4 Resources and References

This booklet contains all that you need to know about routing for this course. However, routing in general and Internet routing protocols are a very complex topic with lots of details, and so the discussion here is necessarily sketchy and incomplete. If you want to dig deeper then I recommend the following books and resources:

- The RIP routing protocol (that you will meet in the assignment) is specified in [20].

- The OSPFv2 routing protocol (mostly used together with IPv4) is specified in RFC 2328 [23]. Furthermore, the author of this RFC has published two books on OSPF: [24] gives a description of the protocol operation, whereas the actual implementation in a Unix environment is described in [25]. A detailed comparison of OSPF and another link-state protocol called IS-IS is given in [9], further information can be found in [10].
- Some references for BGP include its specification [27], the books [15], [21, Chap. 8] and [13]. A discussion of BGP security issues can be found in [5].

1.5 Change Log

All the more important changes are listed here.

- Version 1.0: Initial version

1.6 Acknowledgements

The following people have reported errors or have made suggestions helping to improve the quality of this booklet:

- Adam Ross
- Chris Walker
- Manfred Marriott
- Cole Dishington
- Amelia Samandari
- Matthew Toohey
- Angelica Dela Cruz

Part I

Theory

Chapter 2

Internet Routing Architecture and Routing Protocols

The Internet is made up of a rather large number of end hosts and routers. As of July 2017 there are approximately 1 billion hosts reachable via IPv4[1] and many individual networks and routers have to cooperate to guarantee reachability between all these hosts. With the advent of the so-called **Internet of Things** [26] and further deployment of IPv6 the number of reachable hosts is expected to grow much, much larger.

Routing is a key function of the Internet, and it must work properly at really large (and growing!) scales. In this module we will look into the Internet routing architecture, autonomous systems (AS) and two different routing protocols (BGP and OSPF), representing two different levels of routing granularity.

2.1 Fundamentals

2.1.1 Shortest-Path Routing

To a first approximation, the Internet on the physical level is made of **subnetworks** connected by routers. A subnetwork normally refers to something like a local area network (Ethernet being an example), where two hosts or routers can reach each other on the link layer, whereas hosts on different subnetworks need routers to communicate.[2] A subnetwork is identified by an IP network address together with associated network mask. A router **advertises** its directly attached subnetworks as reachable through itself. We will use the term **subnetwork** to refer to an individual local area network like an Ethernet, to which a number of end hosts and perhaps one or more routers are attached. Routing in the Internet is largely concerned with reaching (aggregates of) subnetworks and not individual hosts.

Routing problems can be conveniently mapped to problems on graphs. The **fundamental task of routing** is to find between each pair of nodes in a "network" (usually represented as a directed or undirected graph $G = (V, E)$) a path that is optimal (or at least of reasonable quality) in some pre-defined sense. The nodes in the graph correspond to routers (representing/advertising their attached subnetworks), and the edges correspond to

[1] http://ftp.isc.org/www/survey/reports/current/

[2] In reality things are slightly more complicated: it is possible to set up different subnetworks on the same physical / cabling infrastructure. For example, with Ethernet it is possible to establish several so-called virtual LANs on the same physical network, and routers are required to communicate between hosts on different VLANs. To complicate matters further, it is even possible to have several subnetworks with different IP network addresses running on the same physical network without VLANs. But we won't deal with that.

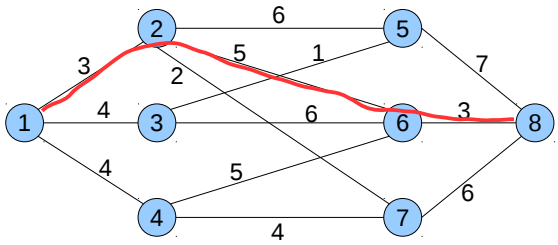


Figure 2.1: An example network showing the shortest path between nodes 1 and 8

the subnetworks interconnecting routers. A packet moves from node to node (i.e. from router to router) through the connecting edges (i.e. subnetworks). Sometimes a subnetwork is also referred to as a **link** or a **hop**. Some possible optimality criteria for paths are:

- minimize the number of hops (i.e. number of subnetworks to traverse)
- minimize the end-to-end delay
- distribute load evenly

and many others. Modern routing protocols either make a fixed choice of what to optimize (e.g. BGP in a sense minimizes the number of "hops"), or are agnostic to the detailed optimization target as long as it can be expressed as a cost value (like in OSPF). Cost values are scalar numbers following the convention that "smaller is better", i.e. smaller cost values are preferrable over larger ones. Examples of specific metrics that fall into this category are the transmission delay on a subnetwork (i.e. the delay is modeled as cost), the number of hops packets need to take, or monetary costs.

All the important routing protocols in the Internet (including RIP [20] and OSPF [9], [24], [25], [23]) use a refinement of the cost model: in particular, costs are assigned to individual hops (or in a graph context: to the edges) and the cost of a path is simply given by the sum of the costs of all the hops (edges) that make up this path. A routing algorithm tries to find the path with the minimum cost. This is known as **shortest-path routing.**

In Figure 2.1 we show an example network, where each edge is labeled with a cost value. The cost of an entire path is the sum of the costs of the edges constituting the path, and the figure shows the best-cost path from node 1 to node 8, which has a total cost of 11 and is better than all the other possible paths.

A special case of shortest-path routing is **minimum hop routing**, where one tries to minimize the number of hops (i.e. packet transmissions on intermediate subnetworks).

Problem 2.1.1 (Minimum hop routing)**.**
How would you configure the link costs in a network to achieve minimum-hop routing?

Problem 2.1.2 (Defining metrics)**.**
Suppose you know for each edge / link e in your network its bitrate b_e and its propagation delay τ_e. Use this information to assign cost values to the edges such that the end-to-end delay between a source s and a destination d is minimized. Assume in this that all packets have the same length l in bits.
What further assumption do you make in this?

Excursion (Widest-path routing)
Note that while shortest-path routing is the dominant paradigm in the Internet, it is not universally used. For example, in the plain old telephone system (POTS) another criterion called widest-path routing is used [21]. Each link is labeled with the number of further telephone calls that can be routed along this link before its capacity is exhausted (its "remaining capacity"). The minimum capacity of a path then is the minimum of all the remaining capacities of its links (which is **not** an additive metric), and then the path with the maximal minimum path capacity is chosen. The intention of this metric is to push out the point in time where some link becomes exhausted as much into the future as possible.

In Appendix A a refresher on fundamental notions concerning graphs and shortest-path routing is given, including two of the main shortest-path routing algorithms, the Bellman-Ford algorithm and the Dijkstra algorithm.

2.1.2 Routing Algorithms and Routing Protocols

A **routing algorithm** solves the routing problem in a centralized fashion. It is assumed at all necessary network information (all the routers and subnetworks / hops and their costs) is available in one place so that we can run an algorithm like the Bellman-Ford or the Dijkstra algorithm (see Appendix A).

A **routing protocol** embeds a routing algorithm into a real networking context and enables the **distributed** computation of routes. In particular, routing protocols organize the information exchange between routers so that each router has all the information necessary to run a routing algorithm and determine the routes. Routing protocols have to deal with various problems and imperfections that can occur in networks:

- Information acquisition and dissemination takes time and bandwidth and might fail or lead to inconsistent information in routers.
- Packets with routing information might get lost.
- Routers or subnetworks / links might crash, or routers might send wrong packets as a result of software bugs.
- Malicious routers might advertise wrong information.

In COSC264 you have learned about two families of routing protocols: distance-vector protocols and link-state protocols. Broadly speaking, distance-vector protocols (like RIP) are organised around the Bellman-Ford algorithm, whereas link-state protocols like OSPF normally use variants of the Dijkstra algorithm. But in reality the actual routing algorithms only make up a tiny proportion of the overall implementation complexity of a fully-fledged routing protocol, as the latter has to provide code to deal with all the issues mentioned above.

In the following we list some properties that designers of routing protocols might wish to optimize for. It is usually not possible to reach all these goals simultaneously:

Correctness properties :

- Computed routes should be loop-free.
- When the network is (strongly) connected, then the protocol should be able to find a route between each pair of nodes.

Robustness :

- A routing protocol should be able to cope with link or station failures, newly established links or stations, changes in link metrics, or congestion situations. In such cases it should be able to establish new routes when old ones become infeasible or are no longer optimal.

Performance parameters :

- From an end-user perspective the identified routes should offer small delays, large data rates and a small packet loss rate.
- From a provider / operator perspective the identified routes should utilize existing equipment well, balance the load, minimize operational costs and maximize the total amount of traffic carried (and thus the revenue).
- When a link status (link goes down or comes up) or a router status changes, routes might need to be re-computed. The **convergence time** of a routing protocol in a given network is the time required by the protocol to achieve a consistent and correct state after such a change in topology, and clearly this time should be small.

Problem 2.1.3 (Users vs. Operators)**.**
Can you always reconcile users goals like small delay and high rate with operator goals?

2.1.3 Addressing and CIDR

We will briefly review IPv4 addressing and CIDR and also have a look at the concept of address aggregation.

A 32-bit IPv4 address (henceforth simply "IP address") consists of a **network part** and the **host part**. The network part are the leftmost k bits of an IP address, the host part are the remaining $32 - k$ bits on the right. A very important question is how many bits should actually be allocated to the network part, i.e. what the choice of k shall be.

Problem 2.1.4 (Choice of k)**.**
Assume that routers only store routing information about IP subnetworks and not individual hosts. When looking only from this perspective, would you want to choose k small or large? What impact does this choice have on the router's speed of processing and memory requirements?

In the early days of the Internet a scheme called "classful addressing" has been used, where the choices for k were limited to $k = 8$ (for class-A addresses), $k = 16$ (for class-B addresses) and $k = 24$ (for class-C addresses). It was possible to tell directly from an IP address to which class it belongs, as the entire address space has been sub-divided into ranges:

- The leftmost bit of a class A address is `0`
- The leftmost two bits of a class B address are `10`
- The leftmost three bits of a class C address are `110`

There is a fourth address range for multicast addresses (class D), where the leftmost four bits are `1110` and the remaining bits being completely allocated to a multicast group address, and a fifth address range (class E, leftmost four bits are `1111`) for experimental purposes.

Problem 2.1.5 (Find the address ranges)**.**
For each of the three main classes (A,B,C) find the precise address ranges and identify how many hosts can theoretically be in a network of the respective class. For class C please consider both the cases where classes D and E are excluded and where they are included into class C.

This has proved to be very inflexible and led to poor utilization of the address space. In particular, after the (few) class-A addresses have been allocated to companies and organisations helping to pioneer the Internet, people turned to class-B addresses. However, only few organizations having a class-B address really have 65,534 different end hosts, leaving large parts of the address space un-utilized. And resorting to class-C addresses blew up routing tables in routers, straining their memory and leading to larger lookup times.

Therefore, an addressing scheme called **classless inter-domain routing** (CIDR) has been introduced in 1993 (specified in RFCs 1518, 1519) and is mandatory nowadays. CIDR is supported by all modern routing protocols (OSPF, BGP, IS-IS, RIPv2, EIGRP) and operating systems, and has completely replaced classful addressing. The idea is simple: instead of specifying the width of the network part implicitly (as in the classful scheme, where we can tell the width of the network part from the address range / the leading few bits) in CIDR it is specified **explicitly**. In particular, in CIDR a network is specified by **two** 32-bit values:

- A 32 bit network address
- A 32 bit network mask (**netmask**)

For a given 32-bit network address the netmask specifies which bits belong to the network-id and which bits belong to the host-id. Netmasks are of a particular form: the leftmost k bits are one and the remaining $32 - k$ bits are zero, and this says that just the leftmost k bits of the network address are the network part (of this address), and the remaining bits are the host part. As a shorthand it is customary to write the specification of a network in the form

```
172.84.0.0 / 16
```

where `172.84.0.0` is the network address and `/16` refers to a netmask with the leftmost 16 bits set to one. Some example netmasks:

Netmask	Shorthand
`11111111.11110000.00000000.00000000`	/12
`11111111.11111111.00000000.00000000`	/16
`11111111.11111111.11100000.00000000`	/19
`11111111.11111111.11111110.00000000`	/23

This approach allows for a more fine-grained choice of network sizes. For example, during the times of classful addressing an organization with just 10 end hosts would have had to apply for a class C address. With CIDR it can instead ask for a `/28` network address, which still fits the required 10 hosts but does not create as many un-used addresses as a class C network address would.

An important notion is the **network address**: when we are given some 32-bit IP address A together with its corresponding `/k` netmask, then the leftmost k bits of A are stringed together with $32-k$ zero bits to give a new address A_N, which refers to the entire network that address A belongs to. For example, a host `192.84.15.7` located in a `/16` network resides in the network with network address `192.84.0.0/16`. As a convention, an IP address in which all the bits of the host-part are set to zero refers to the *network as a whole*, so the term

network address makes sense. As another convention, an IP address in which all host bits are set to one refers to the broadcast address of the network, and by sending a packet to this address you express your intention to send a packet to **all** hosts in this network and only in this network (whether this will really happen is a matter of policy and implementation).

Problem 2.1.6 (Calculate network part)**.**
An end host has address `192.168.40.3` and netmask `/24`, please calculate the network address of the host. Then perform the same calculation with a netmask of `/21`.
What can you observe?

Now consider the operation of a router. A router keeps a routing table (or also often called a forwarding table) with entries telling for a given network address (including its netmask) what will be the outgoing link and the next-hop router to reach this network. For each IP packet to be forwarded, the IP destination address will be extracted out of the packet and the routing table is consulted to find a matching entry (see below). When such an entry has been found, the packet will be forwarded to the indicated next hop. As an example, assume the router has the following entries in its routing table:

Destination network address	Outgoing interface	Next-hop router
`63.120.14.0/8`	`eth0`	`78.12.4.1`
`64.64.0.0/14`	`eth1`	`78.12.4.2`
`144.12.85.128/28`	`eth2`	`78.12.4.3`

and no others. Suppose that the router has to forward an IP packet to destination address `a.b.c.d` (which is not equal to one of the router's IP addresses and which is not directly reachable). Let us consider a few example cases:

- If the packet destination address is, for example, `10.11.12.13` then none of the entries matches and the packet should be dropped.
- If the destination address is `144.12.85.129` then the last entry matches and the packet should be sent to outgoing interface `eth2` to the next-hop IP router shown in the table (which is directly reachable via this interface).
- If the destination address is `144.12.85.177` then the last entry does not match (and no other does) so the packet should be dropped.

To check whether the IP destination address `a.b.c.d` belongs to (or **matches**) some given routing table entry `u.v.w.x/k` of network `u.v.w.x` with netmask `/k`, the router performs the following computation:

- Calculate the bitwise `AND` between the destination address `a.b.c.d` and the netmask `/k` (its binary representation). The result of this computation is denoted as `f.g.h.i`.
- If the result `f.g.h.i` is exactly equal to the network address `u.v.w.x` (for which we assume that the rightmost $32 - k$ bits are zero, otherwise we will apply the bitmask `/k` to it as well) then the entry matches, otherwise it does not.

In general it is possible for more than one entry to match a given destination address (see Section 2.1.4). If only one entry matches, then the packet is forwarded to the outgoing interface (and next hop router) indicated in the entry. If several entries match, then the **most specific** entry is chosen, i.e. the entry with the smallest network

size (or the largest value of k in the netmask). With this convention the **default address** or default router can simply be accommodated by a table entry for destination address `0.0.0.0/0` (with a `/0` netmask **all** bits of the destination address would become zero when applying the `AND` operation to it). For a more detailed discussion of the operations a router carries out in the process of forwarding see Section 2.1.5.

Problem 2.1.7 (Some calculations)**.**
Carry out the calculation for the exemplary routing table and the three example destination addresses from above.

For a given destination address a router generally has to check the entire routing table for a matching entry. A great deal of work has been spent on making this lookup process faster than a linear search [21, Chap. 15].

Excursion (Reserved address ranges)

There are certain special-purpose or reserved address ranges in IPv4, some are shown in this table (quoted from [21], there are more than shown here):

Address Block	Current Usage
`10.0.0.0/8`	Private-use IP networks
`127.0.0.0/8`	Host loopback network
`169.254.0.0/16`	Link-local for point-to-point links (e.g. dialup)
`172.16.0.0/12`	Private-use IP networks
`192.168.0.0/16`	Private-use IP networks

Private-use IP addresses are often used for broadband clients or by NAT boxes. When at home, under Linux use the command `ifconfig` (or `/sbin/ifconfig`, under Windows use `ipconfig`) to print your own IP address. Chances are that it belongs to one of the private-use ranges mentioned in the table.

The key convention around private-use addresses is: no IP packet that carries such an address in its source- or destination-address part will be forwarded in the public Internet, they are only forwarded within parts of the network belonging to your Internet provider. When you wonder how you can nonetheless visit any website, please read about NAT (network address translation).

Another special address is the "traditional" loopback address of a host, `127.0.0.1`, but in fact any address from the `127.0.0.0/8` network serves the same purpose. By using the loopback address you can refer to your own host without bothering to find out what its actual IP addresses really are.

Excursion (Network Address Translation (NAT))

One of the mechanisms to deal with the shortage of IPv4 addresses is **network address translation** or NAT. To explain this in a simple setting, consider a home network made up of a broadband router and a number of devices (e.g. laptops, tablets, smartphones) being used by the people living there.

If you would print the IP addresses of all your devices you would quite likely notice that they are all from one of the private-use IP address ranges. If these are not routed in the public Internet, then how can you still communicate with some external web server?

The following happens: the ISP you are customer of will only allocate a single publicly routable IP address to your home, and this address will be given to your broadband router. This broadband router will furthermore act as a DHCP server for your home network and is hence responsible for allocating IP addresses to your various devices, which it will take from one of the private-use IP address ranges.

When your device sends an IP packet to, say, a public web server, it will put its private-use IP address `SL` into the `SourceAddress` field of the IP header and will furthermore use some random port number `PL` in the `SourcePort` field of the encapsulated TCP header. When this packet reaches the broadband router (which is also the default router for all your devices), it will replace the IP

address in the SourceAddress with the publicly routable address, and if necessary it will replace the SourcePort number PL by a number of its own choosing, say P. Furthermore, the broadband router will keep track of the mapping between the allocated source port number P and the IP SourceAddress S and source port number PL of the originating device.

When the web server sends a response packet, it will carry your public IP address in the IP DestinationAddress field and the port number P in the TCP DestinationPort field. The broadband router extracts P, retrieves the original IP source address SL and source port number PL, and changes the IP DestinationAddress and the TCP DestinationPort fields of the packet to SL and PL, respectively.

This is the simplest form of NAT, there are other forms, e.g. so-called Carrier-Grade NAT.

2.1.4 Address Aggregation

A big problem in the Internet is the size of routing tables: the more network addresses (and accompanying netmasks) are stored in a routing table, the more memory it takes and the more time it takes to do a table lookup for a data packet.

It is therefore crucial to reduce the size of routing tables, and address aggregation is a key mechanism for this. An example: suppose you are an organization with four internal IP networks:

- 64.212.64.0/24
- 64.212.65.0/24
- 64.212.66.0/24
- 64.212.67.0/24

This organization is attached to the public Internet via one router, which we figuratively call the organizations *border router*. Without address aggregation, the border router would advertise these four network addresses to the outside world as reachable through itself. With address aggregation the border router would "aggregate" these four networks into one slightly larger network with address

- 64.212.64.0/22

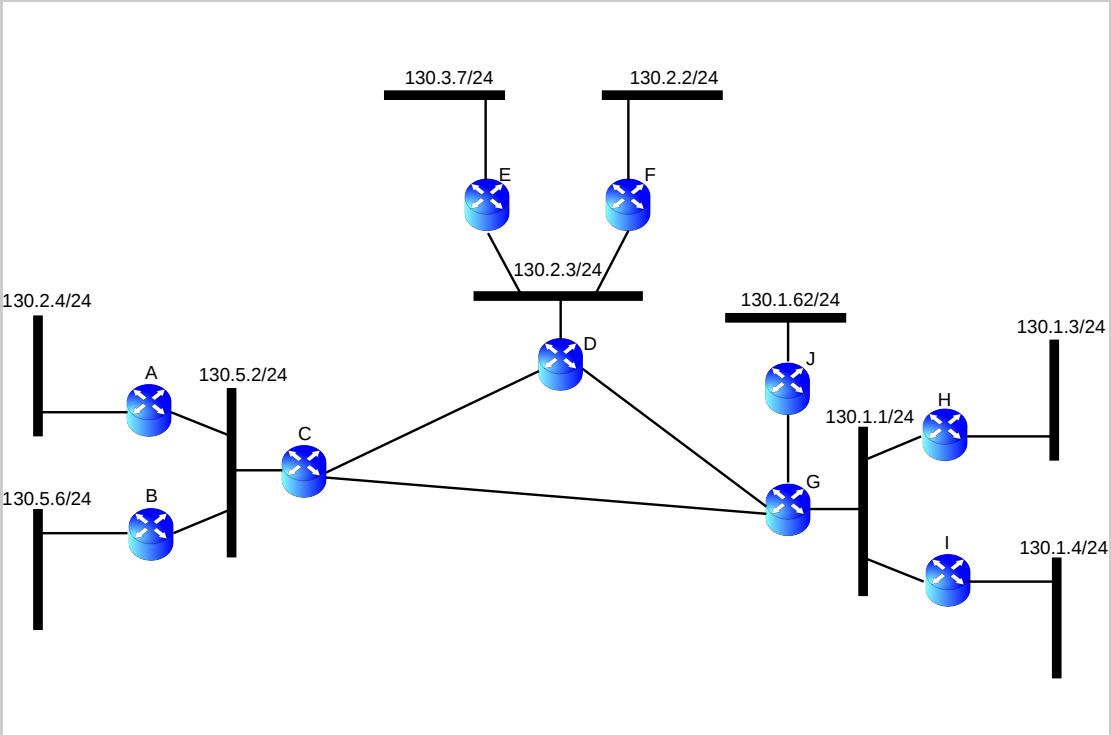
and only advertise this one to the outside world as reachable through itself, while still keeping all four network addresses in its own forwarding table. What will happen?

- The other routers in the outside world only need to store one network entry instead of four.
- *Within* your organization the four networks still remain visible, and all the internal routers within your organization, including the border router, know about them. So, when you send an IP packet from one host within your organization to any other other host within your organization it will just be routed as before, using the internal routers.
- All internal routers will have default routes which eventually point (directly or indirectly) to the border router. When a host inside the organization sends an IP packet to some host outside the organization it can simply do so. The packet will cross the border router and then travel into the public Internet, reaching its destination.
- The most interesting case is that of an external host sending a packet to a host inside your network, say host 64.212.66.77. This external host will send the packet to its next-hop router. This next-hop

router will read off the destination address of the packet (here: `64.212.66.77`) and tries to find a matching address in its routing table. Since the routing table contains an entry for `64.212.64.0/22`, this entry will match (compare Section 2.1.3) and the router will forward the packet. This way, the packet will eventually reach the border router. The border router now knows the *internal* network structure of your organization and can forward the packet in the same way as it would for a purely internal packet.

So, the border router has to know the internal structure of your network, but does not advertise this internal structure (in particular: all the individual networks) to the outside world, it only advertises the summarized / aggregated information and this is all that external routers need to know.

Problem 2.1.8 (Address aggregation)**.**
Consider the network shown in this figure:



(which has been adapted from [24, Fig. 1.5]). In this figure, the thick black lines are Ethernet networks, and the thin lines are either network interfaces of a router towards an Ethernet, or they represent point-to-point links (in particular between routers `C`, `D` and `G`). Each Ethernet network is labeled with its IP network address and network mask. You can assume that routers `C`, `D` and `G` are border routers of their respective organizations.

- Give the routing tables of the border routers `C`, `D` and `G` without address aggregation. Just list the network addresses / netmasks and either the next-hop router for the destination, or mark the network as directly attached. Don't bother about outgoing interfaces.
- Apply address aggregation and list the routing tables in the three border routers.
- What will the routing tables of the non-border routers look like?

Solution to Problem 2.1.8

In the case without aggregation each of the border routers will simply have an entry for all the networks, with a total of ten entries.

Let us see how aggregation can be done in this case. Consider first router G:

- Suppose that router `G` is a "border router" of some organization which has internal networks `130.1.62/24`, `130.1.1/24`, `130.1.3/24`, and `130.1.4/24`
- Router `G` does not announce all these four networks but a *summarized* network, for example `130.1/16`, to the outside world (routers `C`, `D` and possibly others)
- Outside routers need one routing table entry instead of four
- Any packet from the outside destined to any of the four internal networks will go through router `G` which (alone) knows how to forward the packet internally

Next consider routers `C` and `D`:

- Router `D` aggregates its two internal networks `130.2.2/24` and `130.2.3/24` as `130.2/16` and advertises the latter. It furthermore advertises its internal network `130.3.7/24`
- Router `C` has two internal networks from range `130.5` and advertises these as `130.5/16`, but also advertises its internal network `130.2.4/24` to the outside world
- As a result, router `G`:
  - Has two forwarding table entries (instead of three) for `130.2.x` networks: the aggregated `130.2/16` it got from router `D`, and the individual network `130.2.4/24` it got from `C`.
  - Will have two candidate forwarding table entries when it receives a packet destined to `130.2.4.17`
- To resolve this ambiguity, in case of multiple matching forwarding table entries the *most specific* entry is chosen
  - Most specific = more ones in the network mask
  - Here: entry `130.2.4/24` from router `C` is preferred over entry `130.2/16` from router `D`

After doing this, router `G` will have:

- four entries for its own internal networks
- two entries advertised by router `D`
- two entries advertised by router `C`

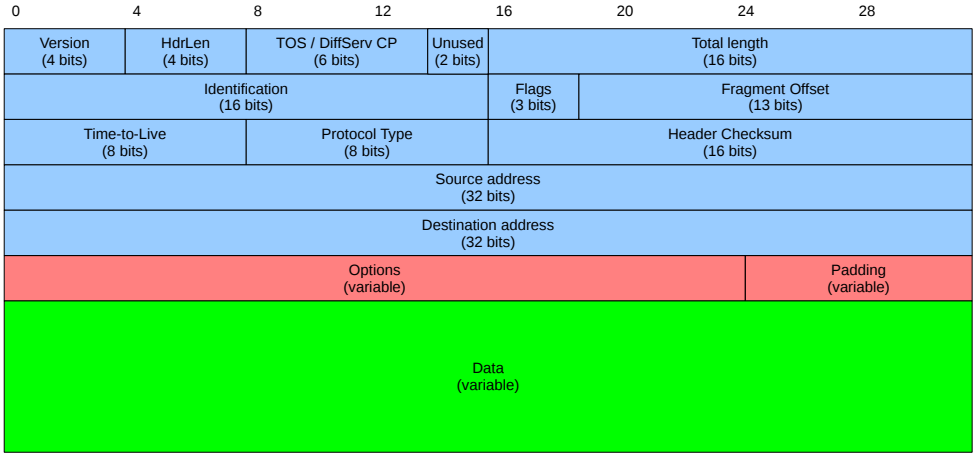
so we have saved two entries.

For the routing table of the non-border routers there are two different possibilities:

- Besides the entries for their directly attached networks a non-border router will only have a default entry `0.0.0.0/0` pointing to their organisation's border router. This approach keeps the routing tables of non-border routers small, but only if the border routers suppress information about the "outside world".
- The border router of an organization does not suppress any information about external IP prefixes and makes it available to all non-border routers of the same organisation. In this case the non-border routers also know all the available IP prefixes.

Now that we know the concept of address aggregation, we can introduce the notion of an **IP prefix**. In fact, it is not IP subnetworks but IP prefixes which are the basic pieces of information that routing protocols disseminate. An IP prefix is simply a network address (together with associated netmask) which can refer to one or, through aggregation, to more than one IP subnetwork, and which is advertised by at least one router. When the IP prefix refers to more than one subnetwork and the prefix has a netmask with k bits, then the first k bits of the individual network addresses of the subnetworks equal the network address of the IP prefix. Note that the usage of the notions IP prefix and IP subnetwork is not uniform in the literature.

2.1.5 Classical Forwarding Behaviour



When an IP router[3] receives a data packet, it performs a number of steps to process it:

- The packet arrives on an incoming network interface (which in routers are also often called **ports** – not to be confused with the concept of port numbers used by TCP and UDP). The incoming interface first strips off all the headers belonging to the medium access control (MAC) / link layer technology, e.g. an Ethernet adapter will strip off the Ethernet header and trailer, check correctness of the Ethernet packet (in particular the Ethernet checksum) and drop the packet if it is incorrect. Otherwise the Ethernet interface will inspect Ethernet's `type/length` header field to infer the higher-level protocol to which the Ethernet payload belongs. If the payload is an IPv4 packet, it will be handed over to the IPv4 software and stored in the IP input queue.
- In the next step an IPv4 packet is checked for basic correctness, e.g. whether the `Version` field in the IPv4 header (shown above) has the value `4` or whether the header checksum is correct.

[3]The first few of these steps also apply in an end host. However, as forwarding is not enabled in an end host, processing stops after checking the destination address.

- Next, packet options are processed, but this happens rarely. One particular option is the source routing option, in which an IP source node can specify the precise sequence of routers that an IP packet should follow towards its destination. We will ignore packet options in the remaining booklet.
- Next, the destination address `dst` of the packet is checked:
  - If `dst` equals one of the IP addresses of the router (recall that each networking interface / port of a router has its own IP address) the packet is destined to the router itself. In this case the router checks the `Protocol` field in the IP header and delivers the IP payload to the corresponding higher-layer protocol software. No further processing takes place. For example, a TCP segment embedded in an IP packet would be identified by the value `0x06` in the `Protocol` field, and the packet would be handed to the TCP software.
  - If `dst` equals the broadcast address of one of the IP subnetworks the router is directly attached to, it again inspects the `Protocol` field in the IP header, delivers the IP payload to the corresponding higher-layer software and stops processing the packet.
  - If `dst` belongs to the IP multicast address range, its further processing depends on how the router supports IP multicast. This is outside the scope of this booklet.
  - In all other cases the router will have to forward the packet further to its destination. Before starting the forwarding process, the router checks its own configuration to see whether forwarding is enabled. If not, the packet is dropped.
- All the remaining steps are concerned with the actual forwarding of the packet, this is called the **IP output stage**.
- First it is checked whether the packet is destined to a station on an IP subnetwork to which the router is directly attached ("destined to a directly reachable host"). If that is true, the router will attempt to deliver the packet directly to the final destination (and then forwarding stops). For example, if the IP subnetwork is an Ethernet, the router will stuff the IP packet as payload into an Ethernet packet and will set the Ethernet destination address of this packet to the Ethernet MAC address of the final destination. The router might have to use the ARP procedure to find out the destination's MAC address. If the final destination cannot be reached, the router may generate a corresponding ICMP message (ICMP `type=3`, `code=1`).
- If the packet is not destined to a directly reachable host, the router will do the following:
  - It decrements the `TTL` field in the IP header and re-computes the header checksum. If the `TTL` field has reached value zero, the packet is dropped and possibly a courtesy ICMP message (`type=11`, `code=0`) is generated.
  - Otherwise the router consult the **forwarding table** (see below) to find an entry which matches the IP destination address `dst` of the packet. If no such entry is found, the packet is dropped and processing stops (possibly a courtesy ICMP message is generated, `type=3`, `code=6`). If one or more entries are found (which can happen as the result of address aggregation, see Section 2.1.4), the most specific entry (i.e. the entry with the most ones in the network mask) is picked and the packet is sent to the outgoing interface and next-hop router indicated by the forwarding table entry.
- IP packets that the router generates itself proceed directly to the IP output stage (the same is true for packets generated by an end host).

Problem 2.1.9 (How to check whether packet is a broadcast packet?)**.**
Suppose that a router is directly attached to an IP subnetwork with address `a.b.c.d/k` where `a.b.c.d` is the IP network address and `/k` is the network mask. How do you check whether an IP packet with destination address `dst` is sent to the broadcast address of `a.b.c.d/k`?

Problem 2.1.10 (How to check whether packet is destined to direct neighbor?)**.**
Suppose that a router is directly attached to an IP subnetwork with address `a.b.c.d/k` where `a.b.c.d` is the IP network address and `/k` is the network mask. How does the router check whether an IP packet destined to `dst` belongs to a host on `a.b.c.d/k`?

A key data structure in this process is the **forwarding table** (sometimes also called **routing table**, but this term is overloaded). This data structure is maintained by a **routing daemon**, i.e. a program running in the background which exchanges routing information with other routers according to a routing protocol (like OSPF, RIP or BGP), performs route calculations and then derives the forwarding table contents from the outputs of these calculations. A forwarding table entry contains the following fields:

- An IP prefix (i.e. an IP network address and associated netmask).
- A specification of the outgoing network interface, i.e. the network interface that a matching packet should be sent to.
- If the outgoing network interface uses a technology in which several stations can be attached to the same subnetwork (i.e. pretty much any technology except point-to-point links), then the IP address of the next hop router (more precisely: the IP address which the next hop router has on the common subnetwork) needs to be specified as well.

A router will at minimum have one routing table entry for each IP subnetwork it is directly attached to. Furthermore, there may be one special entry for the IP prefix `0.0.0.0/0`, which by definition matches **every** IP destination address. This is the **default entry**.

2.2 Autonomous Systems (AS)

Let us start with a simple question: Can you apply a distance-vector routing protocol (like for example a suitably modified version of RIP) or a link-state protocol to the entire Internet? The answer to this question is a firm **no**. A huge technical issue is the sheer scale of the Internet, which both types of routing protocols could not handle well.

Problem 2.2.1 (Scalability issues)**.**
Review the basic operation of distance-vector and link-state routing protocols from COSC 264 and argue why these two types of protocols, when applied to the entire Internet, would have severe scalability problems.

Besides these scalability issues there are a number of more administrative issues. A central point here is that the Internet infrastructure (subnetworks, routers) is not under the control of a single administrative authority. Rather, there is a large variety of stakeholders owning some parts of it, including:

- Transport network owners (e.g. Telecom companies, Sprint and others),
- Internet Service Providers (ISP's), providing connectivity and transport services to customers (which can be end customers or smaller ISP's),

- Companies or institutions having own networks and offering services (e.g. Facebook, Google, Microsoft, government agencies, universities, ...),
- The general public financing infrastructure like exchange points,
- ... and many others.

If just a single routing protocol were to be used throughout the Internet, then all these parties would need to agree on that and would need to agree on the configuration and operation of this protocol. The likelihood of this happening is not very high. Furthermore, routing in such a network would not only be influenced by purely technical or topological considerations (which router is connected to which other routers), but also by financial ones: an end customer like a company or university normally has to pay an ISP before the ISP will forward any packets.

2.2.1 Definition of an AS

To cope with both the scalability issue and the administrative issues, the Internet is partitioned into smaller fiefdoms called **autonomous systems** (AS). A definition of an AS is given in RFC 1930 ([14], emphasis not mine):

Definition 2.1. *An AS is a connected group of one or more IP prefixes run by one or more network operators which has a SINGLE and CLEARLY DEFINED routing policy.*

Some comments:

- As introduced in Section 2.1.4 an IP prefix is an network address (with associated netmask) representing one or more (aggregated) IP subnetworks.
- An AS is often (but not always) owned by one administrative authority, e.g. an Internet Service Provider (ISP) or a (large) company.
- Within an AS, the owner can run one or more so-called interior routing protocols like RIPv2, OSPF (see below) and configure these as he or she likes.
- A **routing policy** does **not** refer to the routing protocol running within an AS, but to the exchange of routing information between different ASes, which in turn is governed by agreements and commercial / legal interests.

An AS is identified by either a 16-bit or 32-bit unsigned integer number (until 2007 only 16-bit numbers were used). To get an AS identifier, you have to apply for it with your regional Internet registry, for example:

- RIPE (Europe)
- AfriNIC (Africa)
- ARIN (North America)
- APNIC (Asia, Australia, NZ)
- LACNIC (Latin and South America)

These regional registries in turn get allocated AS number blocks from the Internet Authority for Assigned Numbers (IANA).[4]

Some example AS numbers are[5]:

AS number	Owner
AS 1	Level 3 Communications, Inc.
AS 3	Massachussetts Institute of Technology
AS 25	University of California, Berkeley
AS 3598, 6194	Microsoft Corporation
AS 6185	Apple Inc.
AS 6195	Goldman Sachs
AS 9432	University of Canterbury

2.2.2 Interior vs. Exterior Routing Protocols

With the concept of an AS in place, we can now introduce a central idea in Internet routing. There are two different levels of routing, corresponding to two different types of routing protocols:

- Routing *within* an AS: for this so-called **interior routing protocols** are used, and examples of such protocols are distance-vector protocols like RIP and EIGRP [12], or link-state protocols like OSPF and IS-IS [16], [9].
- Routing *across* an AS, or routing between AS: for this a so-called **exterior routing protocol** is used. Nowadays there is only one widely used protocol in this class, BGPv4 [27], [15], [21, Chap. 8], [13].

An interior routing protocol runs **inside an AS**. This means precisely that only routers belonging to the same AS exchange routing information with each other using an interior routing protocol. External routers (i.e. routers belonging to other AS) are not involved (with one exception, see below). Note furthermore that it is entirely possible to run several interior routing protocols within an AS at the same time. An interior routing protocol operates on the **physical topology**, i.e. the physical routers and their interconnections through physical subnetworks. The protocol exchanges information about both internal IP subnetworks or prefixes (called **internal routes**) and external ones (**external routes**).

An exterior routing protocol is a routing protocol used to exchange routing information **between different AS**. The main example is BGPv4 (RFC 4271, [27]), which belongs to the family of path-vector protocols and which will be discussed in Chapter 4. More precisely, an AS runs one or more (BGP) **border routers**[6], which exchange information with border routers of other ASes using the BGP protocol (only border routers speak this protocol). When the owners of two AS agree to connect their AS, they establish a communication session between some of their border routers. An important point to understand is that whether or not two AS are linked to each other is not only governed by physical connectivity (which is a necessity), but also by establishing an administrative relationship (e.g. one AS pays the other to forward its traffic). When two ASes are connected, their border routers exchange information about IP prefixes, the AS numbers owning these prefixes and the AS paths to reach them. An AS path is a sequence of connected AS, identified by their AS numbers. BGP border routers learn the AS topology, and **routing is carried out on the AS topology**!! In particular, BGP finds routes

[4] `http://www.iana.org/assignments/as-numbers`

[5] `http://bgp.potaroo.net/cidr/autnums.html`

[6] Note that the term "border router" is overloaded. In the BGP protocol it has a specific meaning, but we have also used it in our discussion of address aggregation (Section 2.1.4) in a generic way, without any reference to any specific routing protocol. OSPF even has different types of border routers.

with the minimum number of AS hops towards a destination IP prefix, no information is exchanged about the number of routers within an AS.

This last point is important to understand: interior routing protocols operate on the physical topology within an AS, whereas exterior routing protocols operate on the topology of AS connections (the network could be called the **AS network**), the physical topology of an external AS is completely invisible to them. The AS network is essentially formed by administrative / political / commercial agreement.

How are these two layers of routing linked together? An AS border router not only speaks BGP to border routers of other AS, it is also an integral part of its own AS and participates in its own interior routing protocols. Such a border router performs a number of important functions:

- It advertises internal IP prefixes to the outside world using BGP, usually at the highest possible level of aggregation. In particular, it advertises that its own prefixes are located within its own AS. The border routers of peer AS disseminate this information further using BGP and eventually all BGP border routers know about it.
- It is quite common that any inbound traffic (i.e. traffic that originates in an external AS and has an internal destination) passes through an AS border router, which then uses information learned from its interior routing protocol(s) to further forward the packet.
- It makes the outside world reachable for internal routers. In particular, all packets originating in the own AS and destined to an external address pass through the border router and are handed over to the next-hop AS.

Problem 2.2.2 (Providing Access to External IP Prefixes)**.**
There are two methods by which a BGP border router can provide access to external IP prefixes to internal hosts and routers:

- The BGP border router can be configured as (direct or indirect) default router for the internal network without advertising external prefixes into the internal network.
- The BGP border router can advertise to its internal network all external prefixes as reachable through itself.

Identify the advantages or disadvantages of these methods, particularly when an AS has several border routers and is connected to several other AS.

2.2.3 AS Types

There is a rough classification of AS, which we explain with reference to Figure 2.2. An AS is represented by a cloud, and can have one or more AS border routers (shown in blue). A line between two border routers / AS indicates that an AS connection (or a **peering**) has been established.

A **multi-homed AS** is connected to more than one other AS. In Figure 2.2 AS2, AS3, and AS4 are multi-homed. In contrast, a **stub AS** is only connected to one other AS. In the figure, this applies to AS1 and AS5.

A **transit AS** is a multi-homed AS which carries "other people's traffic" (likely for money), i.e. it forwards IP packets where both the IP destination address and the IP source address do not belong to the AS – such traffic is known as **transit traffic**. A non-transit AS (e.g. belonging to a company or an end customer) normally drops

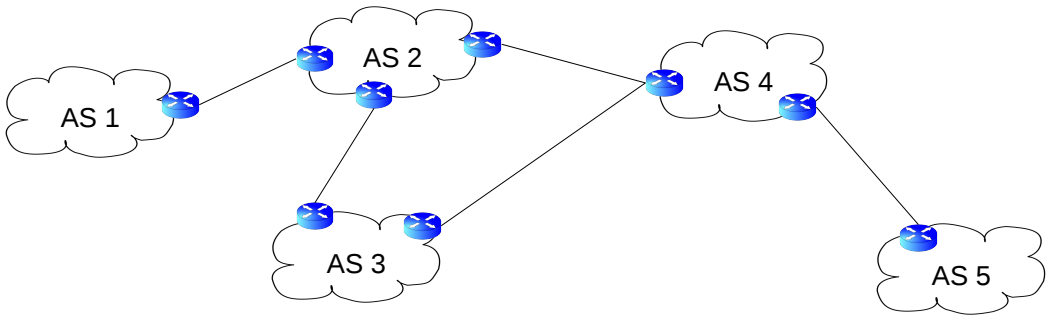


Figure 2.2: Different AS

packets which have not been sent by a host within the own AS or are destined to such a host. An ISP will have one or more transit AS, whereas end customers usually have non-transit AS.

Note that a non-transit AS can be stub or multi-homed. The latter is usually better for improving resilience against network faults.

2.2.4 ISPs, Peering and Points of Presence

An **Internet Service Provider** offers Internet access to end customers like companies, universities and the like. An ISP usually owns one or more transit AS and exchanges traffic with other ISPs and their customers.

A customer is an organization possessing one or more IP prefixes (and the underlying networks) and wishing to attach these to the public Internet with the help of an ISP. A customer can either have its own AS (like for example the University of Canterbury), or it can simply have a number of IP prefixes and these then become part of one of the AS's owned by the ISP.

Excursion (ISP Tiers)
ISP can be coarsely classified into **tiers**:

- **Tier-1 ISP** (the big fish): these run a wide-area (national, continental, global) transport network (called a backbone), typically with high-capacity optical links. It is also typically a company of this weight that is able to invest into sea-cables. They usually have peering agreements (see below) with other Tier-1 ISPs and have Tier-2 or -3 ISPs as customers. Examples of Tier-1 ISPs are Sprint, Level 3, AT&T, Deutsche Telekom.
- **Tier-2 ISP**: these typically operate on a regional or national level, are customers of Tier-1 ISPs (i.e. pay them for transit), and can be customers or peers of other Tier-2 ISPs.
- **Tier-3 ISP**: these operate on a regional level and pay Tier-1 or Tier-2 ISPs for connectivity. It also often happens that this type of ISP sells Internet access to end customers (but ISPs of the other tiers can do this as well).

Note, however, that there is no hard and fast rule by which an ISP can be uniquely assigned to a tier.

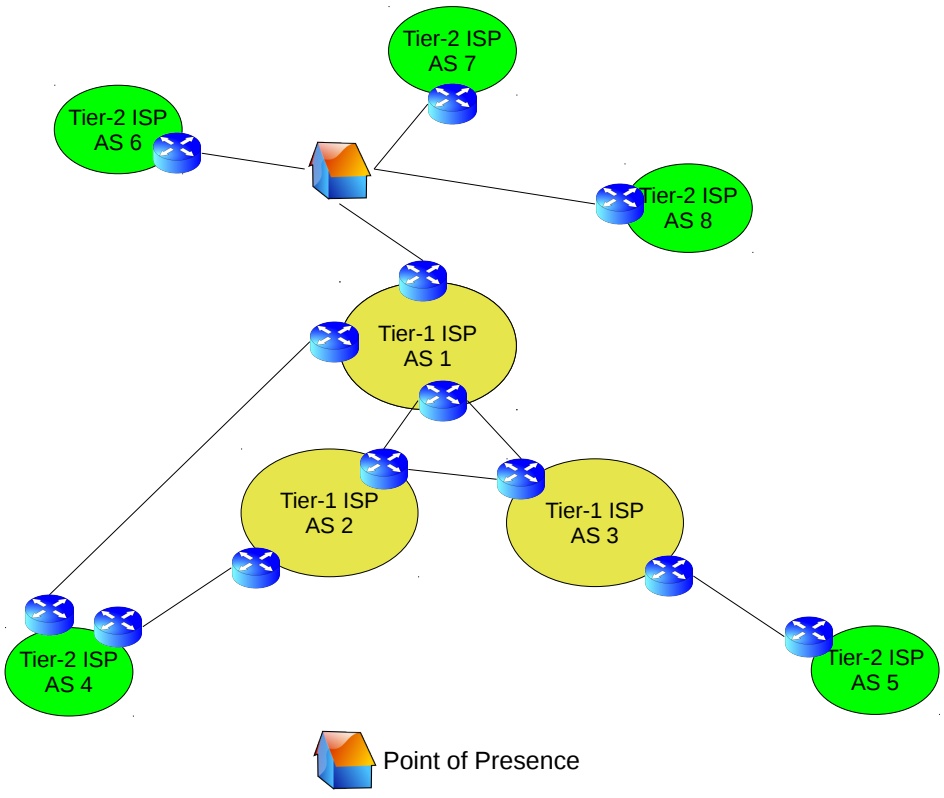
Interconnected ISPs usually have one of two different types of contractual relationship: In a **customer or transit relationship** one ISP pays the other for transit (usually a smaller / higher tier ISP pays the larger one). When the payment is volume-based (and it often is) then an infrastructure for billing and traffic metering needs to be established and maintained, allowing to measure the traffic volumes accurately and reliably (so that the measurement results withstand scrutiny in a possible litigation). In a **peering relationship** two ISPs (typically on the same tier) recognize that they need each others transit service and exchange about the same volume of

traffic both ways, so they agree on simply establishing connectivity *without* setting up measurement and billing infrastructure, simplifying administration.

Furthermore, there are two different ways to actually realize an interconnection between two ISPs. In **private peering** two ISPs establish direct connectivity between each other (with either a peering or a transit contract), whereas for **public peering** a particular institution (called a **point of presence** (PoP) or **exchange point**) is established, interconnecting several ISPs at once. A PoP is usually neutral and administered by a third-party institution. Each involved ISP places one or more border routers at the PoP facility, and through these all involved ISPs exchange AS-level routing information and actual data traffic with each other. The PoP infrastructure provides high-speed links among these routers, and often a PoP looks like a very large, high-performance switched-Ethernet network to all the routers it interconnects. It is also not uncommon that PoPs house routers of particular end customers (e.g. owning a large data center) or content delivery networks. Most countries operate at least one PoP.

Problem 2.2.3 (Content Delivery Networks)**.**
Find out what a content delivery network (CDN) is.

In the figure below, all Tier-1 ISPs use private peering among each other, whereas the Tier-2 ISPs are connected to the Tier-1 ISPs either through private peering (e.g. AS3 and AS5) or PoPs (AS1, AS6, AS7, AS8).



Chapter 3

OSPF

In this chapter we will have a closer look at the OSPF routing protocol, where OSPF stands for "Open Shortest-Path First". OSPF is a widely used interior routing protocol[1] and nowadays comes in two main versions: OSPFv2 (specified in RFC 2328 [23]) is used together with IPv4, whereas the more recent OSPFv3 protocol (RFC 2740 [7]) is required for operation with IPv6. OSPFv3 is not backward compatible with OSPFv2 (which means that OSPFv2 and OSPFv3 routers cannot talk to each other), but many mechanisms are similar. Here we will focus entirely on OSPFv2. The author of the OSPF RFC [23] has published two books on OSPF: [24] gives a description of the protocol operation, whereas the actual implementation in a Unix environment is described in [25]. A detailed comparison of OSPF and another popular link-state protocol called IS-IS is given in [9], further information can be found in [10]. Some of the examples in this chapter have been taken from these books.

Excursion (A case of "Not Invented Here")

In [9] and [24] an interesting glimpse into the history behind the OSPF development is given. In the early 1990s people in the IETF have realized that the then-current distance-vector protocols like RIP had serious weaknesses and wanted to exploit the more recent class of link-state protocols. However, there actually already existed a link-state protocol, the IS-IS protocol that has been developed by an ISO committee as part of an effort to design and implement a protocol stack following the full OSI seven-layer reference model. From the perspective of the IETF a key issue with the IS-IS specification was that it was not "theirs", i.e. not under the control of the IETF. This, among other reasons, triggered the development of the IETFs own link-state protocol, OSPF.

OSPF is a complex protocol, and if nothing else, this chapter will give you opportunities to appreciate this fact. Still, many details and concepts are left away.

3.1 Issues With Distance-Vector Protocols

As you might remember from COSC 264, in distance-vector protocols like RIP a router exchanges information with directly neighboured routers, both periodically and sometimes sporadically (e.g. triggered by special events like discovering a broken link). A RIP router sends a table (or vector) to a neighboured router, in which it lists:

- **All** destination IP subnetworks or IP prefixes it knows of (together with their netmask), and

[1] Please revise Section 2.2 if you are not entirely sure what this means.

- the respective cost to reach them.

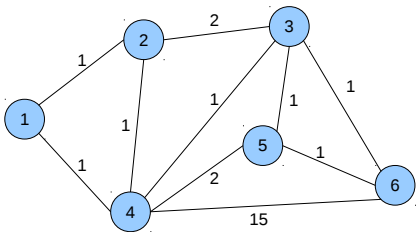
In RIP one transmission of this table is called an **update**. In a large network with very many prefixes, updates can become rather long, and since each router transmits them periodically, they can use up a substantial fraction of the available bandwidth of the links attached to a router. Secondly, since updates can be long, it will take a receiving neighboured router some time to actually process them and re-calculate its own table, which it then sends as an update to its neighbours.

Since RIP has adopted an approach where "bad news" like broken links or crashed routers are distributed through triggered updates but "good news" like availability of links only spreads through the periodic updates, it can take quite a while for "good news" (like the initial announcement of the existence of a link or the fact that a link that was previously down has now been brought up again) to be established throughout a large network with many hops. In other words: convergence can become quite slow! This behaviour is known as "bad news travels fast but good knews travels slow".

This generally slow convergence time of RIP in large networks implies that there can be longer periods of time where different routers have diverging views on the network, and this can lead to a range of problems like routing loops or the count-to-infinity problem. The RIPv2 specification [20] contains a good explanation of some of these problems and the mechanisms used in RIP to mitigate them (at least partially). You will need to read the RIPv2 RFC as part of the routing assignment.

3.1.1 Routing Loops

We will use the network shown in the following figure to show by example the creation of a routing loop in a distance-vector protocol.



Consider the following sequence of events:

- Suppose at time t_0 routing has converged, all nodes have correct routing tables. The routing tables at nodes 2 and 3 (showing only routes to node 6) are:

  Node 2

Dst	Cost	Outg
6	3	2-3

  Node 3

Dst	Cost	Outg
6	1	3-6

  In this table the last column specifies the outgoing interface that the router would use to get to the desination, and in this example outgoing interfaces are given by links $x - y$ between nodes x and y.

- At time t_1 link 3-6 fails

- At time t_2 node 3 updates its routing table entry $\bar{D}_{3,6} = \infty$, and the routing tables (before node 3 sends any update) become

Node 2		
Dst	Cost	Outg
6	3	2-3

Node 3		
Dst	Cost	Outg
6	∞	3-6

- At time t_3 node 2 sends a DV message to node 3, including $\bar{D}_{2,6} = 3$. Node 3 receives the following DV message from node 2

Id=2	Dst=1, Cost=1	Dst=2, Cost=0	Dst=3, Cost=3	Dst=4, Cost=1	Dst=5, Cost=3	Dst=6, Cost=3

- At time t_4 both nodes 2 and 3 perform a routing computation for all known destinations and update their routing tables. The resulting routing table is

Node 2		
Dst	Cost	Outg
6	3	2-3

Node 3		
Dst	Cost	Outg
6	5	3-2

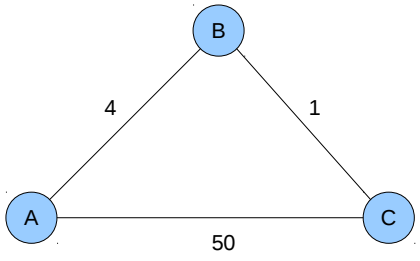
- **And here we have a loop!**

Routing loops can indeed occur in DV protocols e.g. after link failure or major increase in a link metric. In this example, the loop would not have occured if node 3 would have updated its table **and transmitted an updated vector** immediately after t_1 and before time t_3. However, in a distributed environment such race conditions cannot be entirely removed. A more rigorous solution (called Diffusing Update Algorithm) has been incorporated into the EIGRP protocol (Enhanced Interior Gateway Routing Protocol, see [21, Chap. 3], [12]).

In networks where routing loops can occur, packets must somehow be prevented from circulating along the loop forever. The IP protocol contains a mechanism to cope with such failures of routing protocols, the **time-to-live** field and associated behaviour of routers:

- A packet source initializes a specific header field, the `time-to-live` (TTL) field, to some positive integer value (e.g. 32 or 64). The TTL field indicates the maximum number of hops that a packet may take before it gets dropped.
- A router behaves as follows:
  - It reads the TTL field off an incoming packet.
  - If the TTL field is one and the router cannot directly reach the final destination, the packet is dropped.
  - Otherwise, the TTL field is decremented, written back into the packet (with additional checksum re-calculation) and the packet is forwarded further.

The count-to-infinity problem is somewhat related, we again illustrate it with an example network, shown in the following figure (taken from [19, Sec. 4.5]).



Consider the following sequence of events:

- Time t_0: network has converged, and the routing tables are:

Node A		
Dst	Cost	Outg
A	0	–
B	4	A-B
C	5	A-B

Node B		
Dst	Cost	Outg
A	4	B-A
B	0	–
C	1	B-C

Node C		
Dst	Cost	Outg
A	5	C-B
B	1	C-B
C	0	–

- Time t_1: cost on link A-B increases from 4 to 100, node B detects cost change, and computes new path to A (taking into account his knowledge that C can offer a path of length 5 to A), giving:

Node A		
Dst	Cost	Outg
A	0	–
B	4	A-B
C	5	A-B

Node B		
Dst	Cost	Outg
A	6	B-C
B	0	–
C	1	B-C

Node C		
Dst	Cost	Outg
A	5	C-B
B	1	C-B
C	0	–

  Note that we have a routing loop now!

- Time t_2: node B informs node C via update message that its new costs to A is 6, node C re-calculates costs and route to A (which is via B), as:

Node A		
Dst	Cost	Outg
A	0	–
B	4	A-B
C	5	A-B

Node B		
Dst	Cost	Outg
A	6	B-C
B	0	–
C	1	B-C

Node C		
Dst	Cost	Outg
A	7	C-B
B	1	C-B
C	0	–

- Continuation:
  - Node C informs node B about its new cost (which is now 7) and subsequently node B re-calculates its cost to 8
  - Node B informs node C about its new cost (which is now 8) and subsequently node C re-calculates its cost to 9
  - and so on, and so on
  - The procedure stops when node B announces costs of 50, then leading C to adopt the direct link C-A to C
- This behaviour, where it may take many rounds until the right route is discovered, is known as the **count-to-infinity** problem

One cure for the count-to-infinity problem is the split-horizon approach: when transmitting an update message on a link, we only include updated information for nodes for which this link is *not* the next-hop link! Please convince yourself that this solves the previous example. However, it does not solve the problem in general! The RIP routing protocol [20] uses a variant of split-horizon that is called "split-horizon with poisoned reverse".

Problem 3.1.1 (Another problem with DV protocols)**.**
Suppose you are an evil or incompetent person and have root access to a router running a DV protocol. Can you imagine a way in which, by sending well-formed DV messages, you can corrupt routing in parts of the network?

3.2 Basics of Link-State Protocols

We start by motivating and describing link-state protocols on the highest level, without yet referring to the specifics of an individual protocol like OSPF. We will supply a lot more details in later sections.

As observed in the previous section, distance vector protocols converge only slowly ("good news travels slow") and therefore are prone to routing loops or the count-to-infinity problem. Note furthermore that these problems are exacerbated by the fact that routers only know the identifier of a destination, its least cost to it and the next-hop router, but know nothing about the precise topology in other parts of the network or the actual path to the destination.

In contrast, in link-state routing each router possesses a local copy of a **link-state database**, which contains information about all routers, all IP networks / IP prefixes reachable through these routers, and the status or current cost of all links between the routers. In other words, the link-state database reflects the entire physical topology of the network. After each change to the link-state database a router performs a local shortest-path calculation from itself towards all destinations (based entirely on the local link-state database), for example using the Dijkstra algorithm. Furthermore, each router monitors the status (up/down) or the cost changes of the direct links to each neighboured router and directly attached IP subnetworks / prefixes, and subsequently disseminates this link-state information into *the entire network* (using a relatively fast **flooding** mechanism), and not only to neighboured routers. Such a piece of link-state information is commonly known as a **link-state advertisement** (LSA). When a router receives an LSA from one neighbor, it forwards it quickly to its other neighbours (to continue the flooding process), extracts its contents, updates its own link-state database accordingly, and performs a new routing calculation. Note that the router stores the LSA as a whole. Note furthermore that here the router can forward the LSA **before** it performs its own calculations, so that the flooding process can conclude as quickly as possible.[2] Routers send LSAs both periodically and upon changes in the link status or its cost.

Excursion (Flooding)

I assume that you already know what flooding (in a network) really is. If not, then here goes: In flooding, a packet originating at a single node is to be disseminated into the **entire** network (i.e. to all nodes). The process starts by the source node sending the packet to each of its neighbours. A node receiving the packet over some link will, if it sees the packet for the first time, forward it to all its links except the one the packet has been received on. With this approach the packet will eventually reach every node in the network.

Note that a node must keep track of which packets it has already seen, e.g. by storing information about the source of the packet and some packet sequence number (unique with respect to the source) in a local cache. Without applying such a filter criterion the flooding process might never end.

Problem 3.2.1 (Flooding done wrong)**.**
Find an example of a network where the flooding process. run without a local cache of recently flooded packets in each router, does not stop.

In link-state protocols a router keeps two different data structures for routing information: its routing/forwarding table (see also Sections 2.1.3 and 2.1.5) and its link-state database. The forwarding table is the main output of all interior routing protocols. OSPF uses the link-state database to compute the contents of the forwarding table.

[2] A router in a distance-vector protocol must process incoming information before it can disseminate it any further.

A key advantage of the operation of link-state prototols is that LSAs are flooded very quickly into the network, and any inconsistencies in the local link-state databases of different routers (which are the root cause for routing loops and other problems!) persist only for a very short time.

A second fundamental advantage of link-state protocols over distance-vector protocols is that in link-state protocols routers know the *entire network*, whereas in distance-vector protocols routers only know the destination addresses and their best cost to them. This enables link-state routers to immediately calculate new routes when the link towards the next-hop router goes down. And not only that: a link-state router has enough information to calculate *several alternative routes* towards a destination. When these alternatives have different costs, then they provide a list of candidates when the next-hop link / router fails. When there are several alternative routes with the same minimal cost but with different next-hop routers then even more becomes possible: a router can enter all these different optimal alternatives into its forwarding table and use these different paths in a round-robin fashion, achieving some **load balancing** within the network. In the OSPF context this facility is called **equal-cost multipath routing**.

A more specific advantage of OSPF over RIP is that OSPF allows more flexibility in the choice and meaning of link cost metrics – they can refer to anything the administrators like (e.g. delay, link capacity, monetary costs) and are only constrained to be (non-negative) 16-bit integers. Note that this 16-bit restriction only applies to the link costs, the cost of an entire path is not restricted that way. In contrast, RIP only counts hops and the space of path costs is restricted to the range $\{1, 2, \ldots, 15\}$, which also limits the maximum number of hops in a network.

The probably most serious disadvantage of link-state protocols compared to distance-vector protocols is their much higher complexity. As a case in point, just compare the size of the RIP RFC [20] to the size of the OSPFv2 RFC [23].

3.3 OSPF Overview

In the OSPFv2 protocol [23], or OSPF for short, neighboured routers exchange OSPF messages with each other. Logically, the OSPF protocol sits on top of the IP layer, i.e. OSPF messages are encapsulated into IP datagrams. This has the advantage that OSPF can use features of the IP protocol, for example fragmentation and reassembly. An OSPF packet has value 89 in the IP `ProtocolType` field. OSPF routers exchange OSPF messages (in particular messages with LSAs) only with immediate neighbours, and it is the neighbour's responsibility to continue the flooding process. For some types of networks with a physical-layer broadcast facility (e.g. Ethernet), a router sends such a message to all interested neighbours *at once*, using a facility called **link-local IP multicast**. Two link-local IP multicast addresses are used for OSPF, see Section 3.4.

Excursion (IP Multicast)

In general, multicast refers to a communication pattern "between unicast and broadcast". As you might remember from COSC264, in unicast a station talks to exactly one other station, whereas in broadcast a station talks to *all* other stations.

Multicast is in the middle, a station wants to talk to a subset of all stations, which are members of a **multicast group**. When a station sends a packet to a multicast group, then things should be arranged such that all (self-declared) members of the group receive the packet, whereas all other stations do not receive it and do not have to spend any resources on it.

With IP, multicast groups use a special address range, from `224.0.0.0` to `239.255.255.255`. In general, within the IP protocol stack multicast across different IP subnetworks requires special router support and specialized protocols (like IGMP, PIM-SM), which are not universally available.

There is, however, a variant called link-local IP multicast, where transmission and reception of packets is limited to one IP subnetwork and which does not require specialized router support. There is a particular range of multicast addresses allocated to this facility,

from `224.0.0.0` to `224.0.0.255`, and routers never forward packets destined to one of these addresses to other IP subnetworks. Stations in a subnet can "subscribe" to such a link-local multicast group and this way receive all IP multicast messages sent to this group address.

Excursion (Ethernet Multicast and IP Multicast over Ethernet)

When an Ethernet network adapter receives a correct frame it usually checks:

- Is it a unicast frame and does the MAC destination address match the adapters hardware address? If so, the frame is delivered to higher layers.
- Is the frames destination address the Ethernet broadcast address `0xFF:FF:FF:FF:FF:FF`? If so, frame is delivered to higher layers.
- Otherwise the frame is dropped.

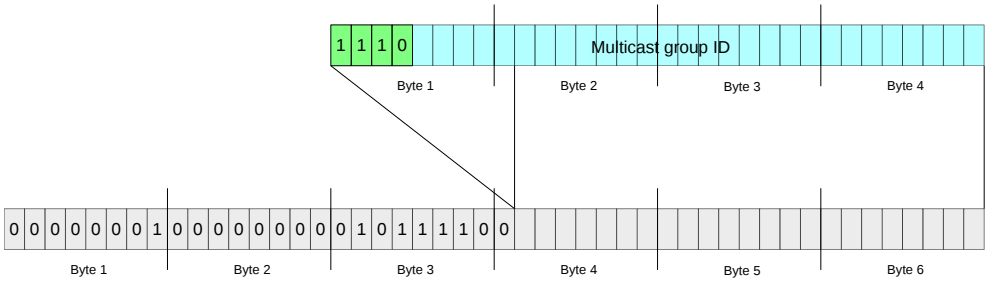
This address filtering is usually done in hardware (to avoid burdening the processor with this work) or in the low-level driver for the Ethernet adapter.

Ethernet also supports **Ethernet multicast**. The Ethernet multicast address range extends from `0x01:00:5E:00:00:00` to `0x01:00:5E:7F:FF:FF`, therefore, the last **23 bits** are available for Ethernet multicast group addresses.

IP multicast groups / addresses are mapped to Ethernet multicast groups / addresses. In particular, when a host wants to send to an IP multicast address on an Ethernet, it sends an Ethernet frame such that:

- The frames `SourceAddress` is the Ethernet hardware address of its network adapter.
- The frames `DestinationAddress` is chosen from the Ethernet multicast address range by mapping the IP multicast address to an Ethernet multicast address.

The mapping is shown in the following figure (compare [22, Fig. 2.5])



Here:

- The upper part of the figure shows a 32 bit IP multicast address, the lower part a 48 bit Ethernet address.
- The last 23 bits of the IP multicast address are mapped one-to-one to the last 23 bits of the Ethernet multicast address.
- The remaining five bits of the IP multicast address and the class-D prefix `1110` are dropped.
- The Ethernet multicast frame has prefix `0x01:00:5E`, followed by one zero bit.

Hence, this mapping is not unique, 32 different IP multicast addresses are mapped to the same Ethernet multicast address. An Ethernet station receiving an Ethernet frame sent to an Ethernet multicast address behaves as follows:

- The higher layers at a multicast receiver configure both:
  - the local IP stack, and

  - the local low-level network driver or adapter hardware

  with the IP multicast addresses it wishes to receive.
- The low-level network driver maps these to an Ethernet multicast address.
- Incoming frames with Ethernet multicast destination address are checked:
  - First by hardware / low-level driver whether the Ethernet destination address is one of the Ethernet multicast addresses the host is interested in – if not, the packet is dropped.
  - Next the IP layer checks whether the packets IP destination address coincides with one of the multicast addresses the higher layers are interested in – if not, the packet is dropped.

In OSPF each router has an identification number, which needs to be unique within the OSPF network. We will refer to this number as the **router id** of this router. There is no general requirement as to how this number is chosen, except that it is a 32-bit number. They can be administratively assigned, or a router could simply choose one of the (supposedly unique) IP addresses of one of its network interfaces as its router id, e.g. the lowest or the highest IP address.

When in the following we talk about the network the OSPF protocol operates on, we will use the more official term **OSPF domain** for this. An OSPF domain refers to the largest set of OSPF routers and IP prefixes / subnetworks sharing the same configuration and authentication data. No OSPF packets are exchanged between different OSPF domains (but data packets clearly can, through OSPF routers attached to both domains). Within an AS, an OSPF domain can cover parts or all of the routers and subnetworks of this AS. The other parts can be covered by other OSPF domains or other interior routing protocols like RIP.

OSPF builds on the mechanisms we already have introduced: the replicated link-state database, and periodic and triggered flooding of different types of LSAs by every router. The link-state database is actually nothing else than a set of LSAs, which are stored on a router in their entirety. There are a number of further concepts which we quickly introduce here and discuss in more detail in later sections.

3.3.1 Adjacencies and the HELLO protocol

Two OSPF routers are physically neighboured when they are attached to the same IP subnetwork (e.g. an Ethernet) and can reach each other on the link layer. However, in OSPF more than just physical connectivity is required for two routers to cooperate.

To add a base level of security, OSPF supports authentication between routers, and two routers only exchange routing information when they can authenticate to each other. Furthermore, there is a range of configuration data on which two routers need to agree before they should talk to each other (see Section 3.6.1).

Therefore, in OSPF the concept of an **adjacency** is introduced, and routers only accept information from (physical) neighbours with whom they have established such an adjacency. Without a lot of detail for now, an OSPF router X sends special `Hello` packets on a given link to a link-local IP multicast address (`224.0.0.5`, aka `AllSPFRouters`), to which all OSPF routers subscribe. These `Hello` packets include, amongst others:

- the router id of X,
- a range of fields related to configuration and authentication, and
- the *neighbour list* of X.

When a physically neighboured router Y receives such a `Hello` packet from router X, it checks whether its own configuration and authentication data match the values sent by X. If not, then X's `Hello` packet is simply

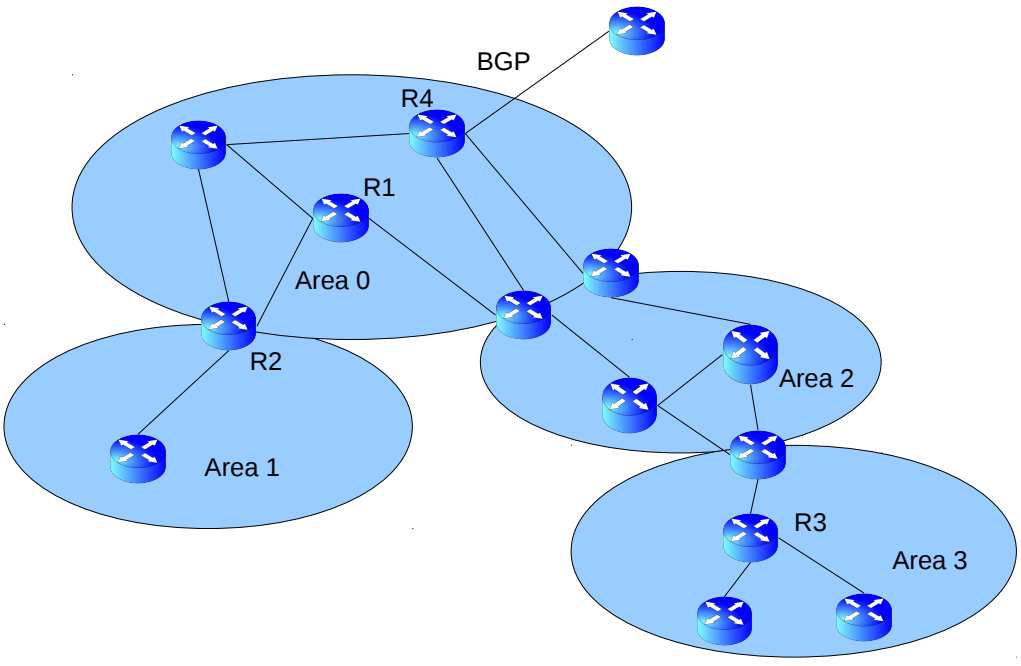
dropped, otherwise Y establishes an adjacency to X and includes X's router id into the neighbour list of its own `Hello` packets. When finally X receives a `Hello` packet from Y with X's IP address in the `Neighbor` list, then (after checking Y's configuration and authentication data) X knows that the link is bi-directional and also establishes an adjacency to Y.

One other piece of functionality is also seen as part of establishing an adjacency. When an OSPF router has just been switched on, it will need to build its link-state database. In the absence of triggered updates the router would normally have to wait until it receives the periodic LSAs generated by the other routers. However, in the OSPF default configuration a router only generates periodic LSAs once every 30 minutes, which means it may take up to 30 minutes for the new router to build its database. To speed up this process, a simple idea is used: the new router asks one or more of its neighbours to transfer their link-state database to it. To support this and similar use cases, OSPF provides a procedure allowing neighboured routers to synchronize their link-state databases, called **database synchronization**, see Section 3.6.

3.3.2 OSPF Areas

When we have compared link-state and distance-vector protocols at the start of this chapter, I have argued that DV protocols have a scalability problem, as the DV update messages can become rather long, whereas an LSA router only generates information about its local environment, i.e. its attached IP subnetworks and its neighboured OSPF routers. Describing an IP subnetwork in an OSPF LSA takes only a few tens of bytes, similarly for a neighbour, and considering that usually routers do not have more than a few dozens to a few hundreds of interfaces the total volume of data a router generates for all its links / interfaces appears moderate.

However, LSAs are periodically flooded, and when the OSPF domain is really large (say: thousands of routers, tens of thousands of links between these) then the aggregate of all LSAs (periodic or triggered) can consume significant network resources. To help with this scalability problem, another level of hierarchy is introduced. An OSPF domain is sub-divided into a number of **OSPF areas** or simply **areas**, and OSPF performs **hierarchical routing** with these areas. We show an illustration in the following figure, where for simplicity we assume that the OSPF domain encompasses an entire AS.



An area is identified by a 32-bit value, called its **area id**. The area id's of different areas must differ. An OSPF area consists of a number of OSPF routers and IP subnetworks – each IP subnetwork in an OSPF domain belongs to exactly one area. Routers belonging to two different areas are called **area-border routers** (e.g. router R2 in the figure), whereas the other routers are called **internal routers** (e.g. router R3). In general, the number and size of areas is not prescribed by the OSPF specification, but there always has to be one area with area id 0, called the **core area** or **backbone area**. Routers belonging to this area are also called **core routers** or **backbone routers** (e.g. R1 and R4).[3] The other areas are called **low-level areas**. The core area is the only area in which one can also have BGP border routers, which in OSPF terminology are called AS boundary routers (ASBR), e.g. router R4 in the figure. ASBRs provide routing information about IP prefixes owned by other AS. Therefore, only the core area is attached to other AS.

Importantly, an OSPF area is a boundary for flooding, and each area has its own separate link-state database. The link-state database of an internal router contains only the routers and IP subnetworks belonging to the same area (this could also be aggregated IP prefixes if one internal router advertises them this way). An area-border router has as many link-state databases as it has areas in which it is a member. For example, router R2 has two separate link-state databases, one for area 1 (to which it has one interface) and one for area 0 (into which it has two interfaces). All LSAs issued within an area A are confined to this area, an area-border router does not forward them to other areas. Instead, the area-border router advertises "internal" prefixes from area A to another area B as reachable through itself (this is called **summarization**), but it does not give any detailed topology information about area A into the other area B. This "firewalling" limits the scope of flooding to one area. For a given area X an area-border router of X is then responsible for advertising the IP subnetworks / IP prefixes belonging to area X to other areas as reachable through itself. Conversely, the area border router advertises IP prefixes from other areas (or possibly even other AS or other interior routing protocols) into area X as reachable through itself. It can do this by either explicitly advertising these foreign IP prefixes into area X as reachable through itself, or it can become the default router for area X.

With the concept of sub-dividing an OSPF domain into areas and linking these areas through area-border routers the question arises what the allowed area topologies actually are and how paths between areas are formed. In OSPF the area topology is limited: **all low-level areas must be connected to the core area and not to other low-level areas** – this is also called a "hub and spoke" topology.[4] Remember that an area border router R_X for area X represents the IP subnetworks of X to the other areas as reachable through itself, *without exporting link-state or other topological information from within area X!* Therefore, other area border routers only see that IP prefix P_X (belonging to area X) is reachable through R_X, and R_X's advertisements of P_X into the core area contain the information that *his* costs to reach P_X are C_X. Any other router R_Y in the core area will have to calculate its cost to prefix P_X by adding up its own cost $c_{Y,X}$ to reach R_X (which is obtained from the link-state database of the core area) plus R_X's cost to reach P_X, i.e. C_X. So we are back to a situation that is more similar to a distance-vector protocol, where routers only advertise their total costs towards a destination, but not the detailed topological information to get to that destination (all the involved links and their costs). This sounds like a paradoxical situation: we want to use the link-state approach to get rid of the disadvantages of distance-vector protocols, but through the area concept and the concept of summarization by area-border-routers the distance-vector concept sneaks in again! This is true, but this is also the reason why the restriction that all low-level areas must only be connected to the core area (and no other area!) has been introduced. This way we get a "hub-and-spoke" (or star-shaped) area topology, **and in this simple kind of topology the problems of distance-vector protocols are manageable!**

[3] The term "core router" or "backbone router" is again an overloaded term. Inside OSPF, it refers to a router belonging to area 0. Outside OSPF, it usually refers to a router that has a complete, internet-wide routing table and which has no default router, but is itself the ultimate default router for other routers.

[4] You will notice that the figure above seems to suggest otherwise, but see the concept of virtual links in Section 3.3.3.

Problem 3.3.1 (Distance-vector protocols in a hub-and-spoke architecture)**.**
Why can't routing loops occur in a "hub-and-spoke" architecture, where we have one "central" node to which all other nodes are attached, and where no non-central node is connected to any other non-central node.

Having seen why low-level areas are only connected to the core area, one next realizes that this requirement might be hard to satisfy in practice if the only way to connect two areas is to have one area-border router being physically connected to both of them. Therefore, low-level areas can be connected to the core area either physically or through "virtual links" (compare area 3 in the figure, see Section 3.3.3). A virtual link is established between two non-adjacent routers by creating a tunnel.

3.3.3 Network Types, DRs and BDRs

OSPF can support five different types of IP subnetworks:

- In **point-to-point networks** OSPF routers are connected through point-to-point links, i.e. only two routers share a transmission medium, and whenever one of them sends there is only one receiver. Examples include dial-up lines or optical links / optical circuits between routers. In this type of networks it is trivial to discover a neighboured OSPF router. It is not even necessary to allocate an IP network address to the point-to-point link.
- In **broadcast networks** several OSPF routers are attached to an underlying IP subnetwork with MAC-layer broadcast capability, like for example an Ethernet.[5] In such a network, a packet transmission by one router to a MAC-layer broadcast or multicast address is heard by *all* other routers attached to the same subnetwork. The discovery of all neighboured OSPF routers is easy on broadcast networks.
- In **non-broadcast multiaccess networks** (NBMA) several OSPF routers are attached to the same IP subnetwork and can reach each other, but this subnetwork does not have a broadcast facility. Examples of this type are ATM and frame relay networks. Due to the absence of a broadcast facility it is not easy to discover neighboured OSPF routers, this usually requires additional configuration.
- **Point-to-multipoint network** are similar to NBMA networks, i.e. several routers are attached to a network without broadcast facility. Furthermore, it is not required that on such a subnetnetwork each router can reach each other router, i.e. partial connectivity is allowed here. This can happen for example with wireless networks.
- With **virtual links** it is possible to connect two non-neighbored OSPF area-border routers through other routers in an intermediate area. Intuitively, these two routers establish a tunnel over the intermediate area, and again the discovery of the "neighboured" router in a virtual link requires configuration.

These network types differ in the methods by which routers discover and reach their neighboured routers.

Problem 3.3.2 (Tunneling)**.**
What is tunneling (or encapsulation)?

[5]Note that in Ethernet a station will throw away packets that are not destined to itself, to the Ethernet broadcast address or to an Ethernet multicast address to which this station has been subscribed, this is called **address filtering**. In broadcast Ethernet address filtering is done by a receiving station, in switched Ethernets it is done by the switches. However, frames sent to the broadcast or to a multicast address are indeed received by **all** stations.

Out of these five types of networks there are three (broadcast, NBMA, point-to-multipoint) in which several OSPF routers are attached to the same IP subnetwork and (in the case of broadcast and NBMA networks) can reach each other directly with a link-layer transmission. In this type of subnetworks there is one particular scalability problem, which I will explain by example of Ethernet: suppose that there are n OSPF routers attached to the network that share the same configuration / authentication data and can establish adjacencies with each other. If we do that, then:

- Each router would create $n-1$ adjacencies for this network (to each other router) and the total number of adjacencies would be $n \cdot (n-1)$.
- Each router would have to go through $n-1$ database synchronization procedures.
- Each time a LSA is flooded, a router would forward it to all of its $n-1$ neighbours, which in NBMA networks can amount to $n-1$ separate transmissions of the packet containing the LSA, plus acknowledgements (which are needed for reliable flooding, see Section 3.7). In an Ethernet one transmission of an LSA packet would suffice (because of the broadcast capability of Ethernet) but there would still be $n-1$ acknowledgements, and each router would initiate one separate transmission.
- Each router would report $n-1$ neighbors in its LSAs, and each router would report the network itself, leading to a lot of redundancy and overhead.

The solution to this scalability problem essentially consists of electing one particular OSPF router as the representative of this subnetwork. This router is called a **designated router** (DR), and it is dynamically elected during the neighbour / adjacency establishment process in broadcast and NBMA networks. The designated router represents and advertises the subnetwork to other routers outside the subnetwork and participates in the flooding process for LSAs generated outside the network. All the other routers in this subnetwork establish an adjacency only to the DR and perhaps another special router called the **backup designated router** (BDR). All routers receive and process LSAs, but the other routers only establish an adjacency (and synchronize their databases) with the DR and the BDR (if any).

However, in this setup the DR becomes a single point of failure: when the DR fails, the flooding of LSAs packets would fail as well, since the DR plays a key role in this. Therefore, a BDR is elected as a standby in case the DR fails. An individual OSPF router can be DR in one broadcast subnetwork and can be non-DR/BDR in another subnetwork. The DR/BDR mechanism requires two sub-protocols (which we will not discuss any further):

- An election protocol for selecting the DR and BDR in a subnetwork.
- A mechanism for failure takeover by the BDR, once it has discovered that the DR is non-operational anymore. One ingredient of this takeover mechanism is based on `Hello` packets, see Section 3.6.1.

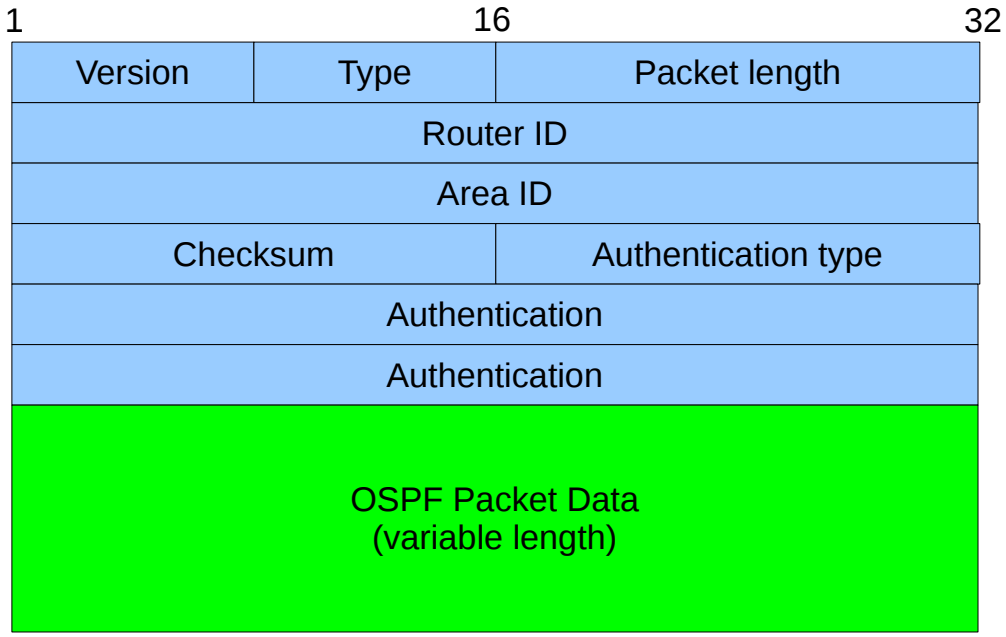
Having both a DR and a BDR in place, there are only $2n$ adjacencies instead of $O(n^2)$.

3.4 OSPF Packets and Messages

As introduced above, OSPF packets are encapsulated into IP packets as payload, i.e. they do not use the extra services available from UDP or TCP. The `Protocol` field in the IP header is set to 89 for OSPF packets, the `SourceAddress` field is set to the IP address of the OSPF router on the interface the packet is sent, and the `DestinationAddress` field is, depending on circumstances, one of two well-known link-local multicast addresses:

- `224.0.0.5`, also known as `AllSPFRouters` (all OSPF routers of the same OSPF domain)
- `224.0.0.6`, also known as `AllDRouters` (referencing all (backup-)designated routers)

The intention of using link-local multicast addresses is to make sure that OSPF packets are always only transmitted to immediately neighboured routers and not any further.[6] As another safeguard to ensure this, for most of the OSPF packets the `TTL` field in their IPv4 header is set to only one.[7] All OSPF routers have to subscribe to the `AllSPFRouters` multicast group, in broadcast and NBMA networks the designated router and the backup designated router also subscribe to the `AllDRouters` group.



All OSPF packets have a common header, shown in the preceding figure. This header consists of the following fields:

- `Version` refers to the OSPF protocol version (here: 2).
- `Type` refers to OSPF packet type. OSPF distinguishes five different types of packets.
- `Router-ID` is the router id of the OSPF router generating the packet (see Section 3.3).
- `Area-ID` is the identifier of the OSPF area to which the interface on which the originating router sends the packet (or the subnetwork the packet is being sent to) belongs. In the core area this value is 0.
- The authentication-related fields allow routers to prove to each other that they are who they claim they are. We will not discuss this any further.
- `Checksum` extends over the whole OSPF packet. A receiving router must check this and throw the packet away if the checksum is wrong.

[6] And on subnets of broadcast type, using a link-local multicast is also more efficient, as it allows to reach **all** attached routers with just one transmission, instead of requiring a separate transmission for each neighbored router.

[7] You do remember what the `TTL` field is good for, do you?

The OSPFv2 protocol knows five different types of packets:

- `Hello` packets (`Type=1`): used to establish and maintain neighbor adjacencies.
- `Database-description` packets (`Type=2`): there is a sub-protocol allowing two neighbored OSPF routers to exchange their routing information (more precisely: to synchronize their link-state databases) while establishing an adjacency, this is useful for re-synchronization after a router crash. Recall that the link-state database is nothing else than the union of all LSAs, and the `database-description` packets contain **summaries** of these LSAs.
- `Link-state-request` packet (`Type=3`): the `database-description` packet includes only **summaries** of link-state entries – when either party during adjacency setup notices that the peer has more recent data, it requests it using a `link-state-request` packet.
- `Link-state-update` (LSU) packet (`Type=4`): provides a container for several link-state-advertisement (LSA) records, sent upon link state changes, periodically or during database synchronization in the course of an adjacency establishment.
- `Link-state-acknowledgement` packets (`Type=5`): sent in response to successful reception of LSU packet, used to make flooding reliable.

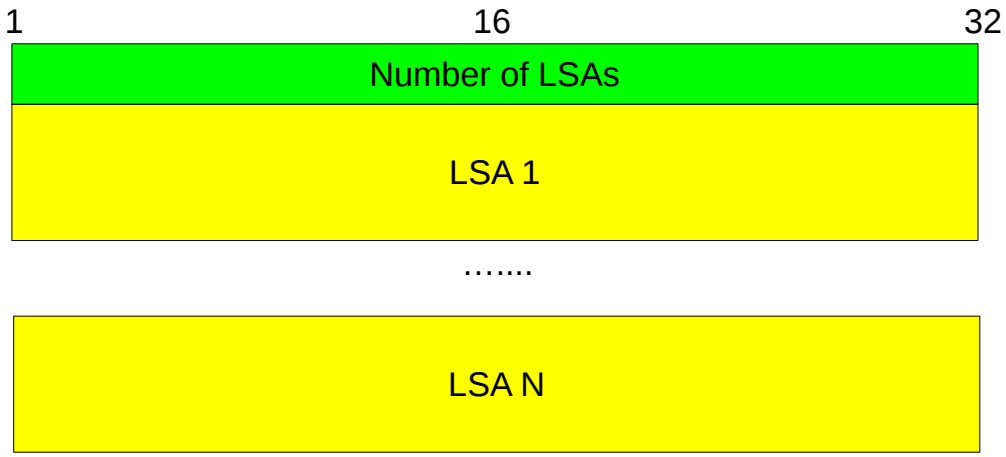
The first three types of packets will be discussed in Section 3.6, the `link-state-acknowledgement` is used in the flooding process (Section 3.7) and the LSU / LSA packets are discussed now.

Be careful: There is the risk of confusing link-state-advertisements (LSAs) as core part of a link-state routing protocol, several of which are stuffed into a `link-state-update` packet in OSPF) and link-state acknowledgements. When referring to LSAs, we always mean the advertisements (or the LSA records in OSPF terminology).

3.5 LSA Records

We now discuss LSA records in some more detail.

As explained above, a LSU packet (link-state-update) is simply a container for one or more LSA records. Therefore, the structure of a LSU packet (following the common header) is simple:

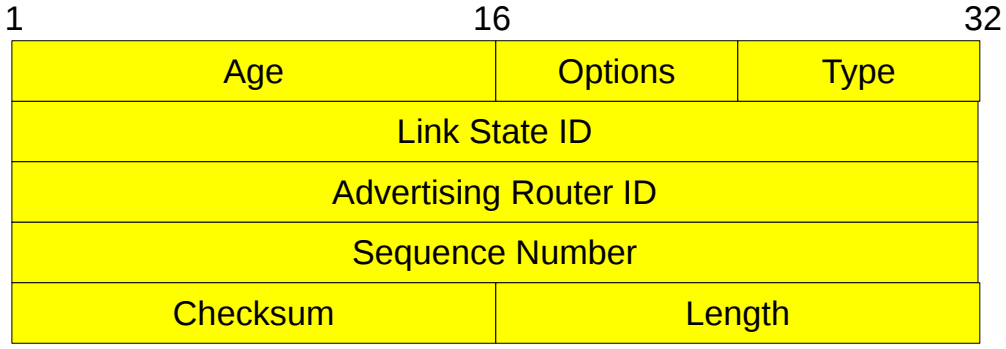


The first 32 bits indicate how many LSAs are contained in the LSU packet, and then the LSAs follow. Note that LSU packets are carried as payload of IP packets, and the IP layer can carry out fragmentation and reassembly when the LSU becomes longer than the maximum length of a packet on the underlying IP subnetwork.

Problem 3.5.1 (Why all the alignment?)**.**
You can notice in this and in all following packet formats that, wherever sensible, fields and records are nicely aligned to 32-bit boundaries. Can you explain why?

3.5.1 Common LSA Header

The OSPFv2 specification distinguishes eleven (!!) different types of LSAs, furthermore there can be vendor-specific types. Some of these are irrelevant for us, as they refer to functions outside the scope of this lecture – for example to OSPF-based multicast, traffic management and others. We will discuss only some of the LSA types in more detail. However, before doing this, we first show the **common header** of all OSPF LSA types:



The meaning of some of these fields is straightforward:

- The `AdvertisingRouterID` carries the router id of the router originating this LSA.
- The `Type` field indicates to which of the eleven types of LSAs the present LSA belongs. Examples:

  - `Type=1` is a **router LSA**, with which a router advertises itself, all immediately attached subnetworks and, depending on the circumstances, neighboured OSPF routers.
  - `Type=2` is a **network LSA**, generated by a DR for the IP subnetwork it represents, it furthermore lists the BDR and all the other routers in that subnetwork.
  - `Type=3` is a **summary LSA**, generated by an area-border router to summarize the available IP subnetworks / IP prefixes in one area to routers in another area. The router only lists these subnetworks and gives its own costs to reach them, but does not provide any information about the detailed topology within the area (see Section 3.3.2).
  - `Type=4` and `Type=5` are concerned with routing information that has been gained from other routing protocols (e.g. RIP within the same AS or BGP across AS) and is injected into the OSPF domain. We will not consider this any further.

  These five types must be supported by every OSPF router. It is generally possible to add new LSA types, and some vendors have done just that. When an OSPF router does not support a particular LSA type, it drops the LSA.

- The `Options` field is not relevant for our purposes. Its main usage is to mark an LSA as "special", e.g. to express that any IP prefixes announced in it belong to external AS. One particular bit, the `E` bit is set when the originating router is part of the core area and sends the LSA into this area, it should also be set if the advertised routes are external to the own AS. This is for informational purposes only and does not affect routing calculations.
- The `Length` field gives the length of the entire LSA in bytes.

The other fields deserve some more explanation and justification, see below.

In some places we will need to make a distinction between a (generic) LSA and one of its (recurring) instances. A generic LSA is characterized by its `Type` field, the `AdvertisingRouterID`, and the `LinkStateID`. The values of these fields are common across all the different **instances** of an LSA, and different instances can have different `Age` fields, `Sequence number` fields, metric values, neighbour lists and the like.

The `Checksum` field

For a link-state protocol it is of fundamental importance that the local link-state databases of all nodes are consistent and identical most of the time. The flooding process is used to help keeping the times of inconsistency short. To further maintain the integrity of an LSA while it resides in the memory of a router, it is protected with its own checksum. A router periodically re-calculates the checksum of an LSA stored in its database and compares it against the `Checksum` field of this LSA. If there are differences, then the LSA is purged from the database. The checksum protects against errors introduced in two different ways:

- Errors introduced during transmission of an LSA: packets transmitted on physical media can be corrupted due to noise, interference, low signal strength and other reasons. Link-state protocols (which typically operate on the network or the transport layer) cannot choose the underlying link-layer technology and cannot know whether the link-layer has own packet checksums (most have, though). Note, however, that this type of errors will also be detected by the checksum field in the enclosing OSPF common header.
- A LSA can become corrupted in memory while being stored in the link-state database, for example due to software bugs, randomly flipped bits, etc.

There are further uses of the checksum field, not discussed here.

The `Age` field

The `Age` field is used by an aging or **soft state** mechanism by which routers can remove outdated or stale LSAs for which they have not received any update in some time. The `Age` field indicates the time (in seconds) that has passed since an LSA has been originated.

When a router originates a new LSA it sets the `Age` field to 0. During the flooding process other routers might take some time to process the LSA and update the `Age` field accordingly. A router periodically traverses its link-state database to increase the `Age` fields of all the LSAs stored in the database (remember that routers store received LSAs *completely*). When the `Age` field of an LSA reaches the value `MaxAge` (which is set to one hour) the LSA is considered stale and is purged from the database. To make sure that the other routers also drop the LSA, a router on which the `Age` field reaches `MaxAge` re-floods the LSA with the `Age` field set to `MaxAge`, causing the other routers to drop the LSA from their database as well. By this, when a router crashes, all the LSAs generated by it will be purged after at most one hour. Note that within one hour a router will originate an LSA at least twice, since the generation period is normally 30 minutes.

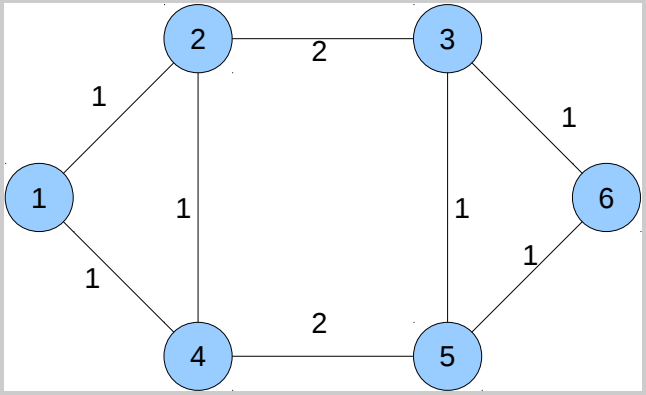
Now suppose that indeed a router crashes. Then an important question is whether this means that all the other routers will continue to use paths involving the crashed router for another hour. The answer to this question is: the path can still be in use for some time, but not for an entire hour. This is made sure by the following two mechanisms:

- A router includes a link in a path only when the routers at *both* ends of the link report the link as alive.
- When a router crashes, it will cease to send periodic `Hello` packets, which are normally generated every ten seconds. A peer router, having established an adjacency with the crashed router, will notice that there have been no `Hello` packets for some time. This time is configurable and called the **router dead interval**, a typical default value is 40 seconds. When the router dead time has passed without receiving `Hello` packets, the peer router will issue a new LSA about this link which announces it as down.

The `Sequence Number` field

The `Sequence Number` (or `Seqno` field) allows a receiving router to distinguish older information for a particular LSA from more recent information about the same LSA, and to drop the older one. Since LSAs are flooded, and in particular when a router generates two different instances of an LSA in quick succession (e.g. because of fast fluctuation of the link costs) then it may well happen that another router receives multiple copies of the same or different LSA instances within a short time, over different paths and with different delays, and perhaps not in the order in which they have been generated at the originating router. Sequence numbers have been introduced to ensure that routers can drop anything but the most recent LSA instances. In particular, routers accept an LSA instance only when its sequence number is "more recent" than the sequence number of the corresponding LSA instance stored in its local link-state database. Otherwise, the router assumes that the received LSA instance is out-dated and drops it.

Problem 3.5.2 (What can go wrong without sequence numbers?)**.**
Consider the following network:



and suppose that in some link-state routing protocol the LSA's consist only of the `SrcNode`, `DstNode` and cost `Cost` fields, but no `Seqno` field. At some time the link between routers 1 and 2 fails. Give a timeline indicating which router sends / forwards which LSA that leads to receiving confusing information at node 4.

In more detail, a router i maintains a sequence number s_i for a particular LSA it generates. Periodically, or when the cost of some link $i \mapsto j$ has changed (requiring a triggered update), router i sends a new LSA instance, includes the current sequence number s_i in the `Seqno` field, and increments s_i afterwards. Any other router k receiving a LSA instance for link $i \mapsto j$ stores the received LSA instance and its sequence number. When router k receives a LSA instance from i for $i \mapsto j$, it checks whether the `Seqno` field of the received LSA instance is strictly larger than the `Seqno` field of the corresponding LSA instance stored in the link-state database. If so, the new LSA instance is accepted and stored in the link-state database (plus a new routing computation is carried out when necessary). If the new LSA instance has the same `Seqno` as the one stored in the database, it will be accepted when it has a smaller `Age` value. Otherwise, the new LSA instance is dropped. Note that there is no coupling between sequence numbers at different nodes (not even when sharing the same link).

One design aspect is the actual sequence number space. A sequence number consists of a finite number of bits. Main design options include:

- Linear sequence number space: In this option the initial sequence number is 0 and the sequence number is simply incremented until it reached its maximum (where it then stays). When the sequence number range is large enough, say 32 bits, and a router always starts with a sequence number of 0, then it will be practically impossible for a router to ever reach the maximum, even if it generates LSAs once every second. This is the design option chosen in OSPF, which has 32-bit sequence numbers (but uses a different number range). With this approach, testing two sequence numbers for which is the more recent one is simple: when $s_1 < s_2$, then the latter number is more recent.
- Circular sequence number space: such a sequence number space has no natural start or end: a sequence number is always incremented modulo the maximum sequence number plus one. For example, with a three-bit sequence number we can represent sequence numbers from 0 to 7, the maximum number plus one is 8, and the successor of a sequence number s then is computed as $(s + 1) \mod 8$. In particular, $(7 + 1) \mod 8 = 0$. In this setting, designing a test for two sequence numbers s_1 and s_2 to figure out which is more recent is harder and not possible to do unambiguously (see Problem 3.5.4).

Problem 3.5.3 (Small linear sequence number spaces)**.**
Suppose we are using OSPF with a linear sequence number space, but the maximum sequence number is chosen to be so small that it can be practically reached. What could the router do when it has reached the end of the sequence number space and wants to send an update that is accepted as new by the other routers? Explain the problem and give one or two design options.

Problem 3.5.4 (Circular sequence number spaces)**.**
To illustrate the problem of devising a useful test which of two sequence numbers is more recent, simply consider the case of a three-bit sequence number space (numbers $\{0, 1, 2, \ldots, 7\}$) and two sequence numbers $s_1 = 3$ and $s_2 = 7$. It is not really easy to say with confidence which is more recent.
With a sequence number space of n distinct sequence numbers, design a decision scheme which for two sequence numbers a and b decides which is more recent.

As already mentioned, the main purpose of the `Seqno` field is to allow a router to distinguish more recent from older LSAs. What happens if the router has to compare two LSAs with the same value in the `Seqno` field, e.g. a recently received one and the one in its link-state database? Normally a router accepts the LSA with the smaller age, assuming it is more recent. An exception to this is when the `Age` field is set to `MaxAge`, which then indicates that the LSA actually should be dropped from the database.

Another design issue with sequence numbers is what happens when a router crashes and has no memory of the last sequence number it has sent. After re-initialization it would be problematic for the router to start from the beginning of the sequence number space, for reasons you have figured out in Problem 3.5.3. To solve this problem, a router, intuitively speaking, asks his neighbour what has been the last sequence number s_i it has sent (with any LSA), and then uses $s_i + 1$ as its starting sequence number. More formally, this happens during the database synchronization procedure, see Section 3.6.2.

The `LinkStateID` field

To quote from the RFC [23, Sec. 12.1]: "This field identifies the piece of the routing domain that is being described by the LSA": The usage of this field depends on the type of the LSA:

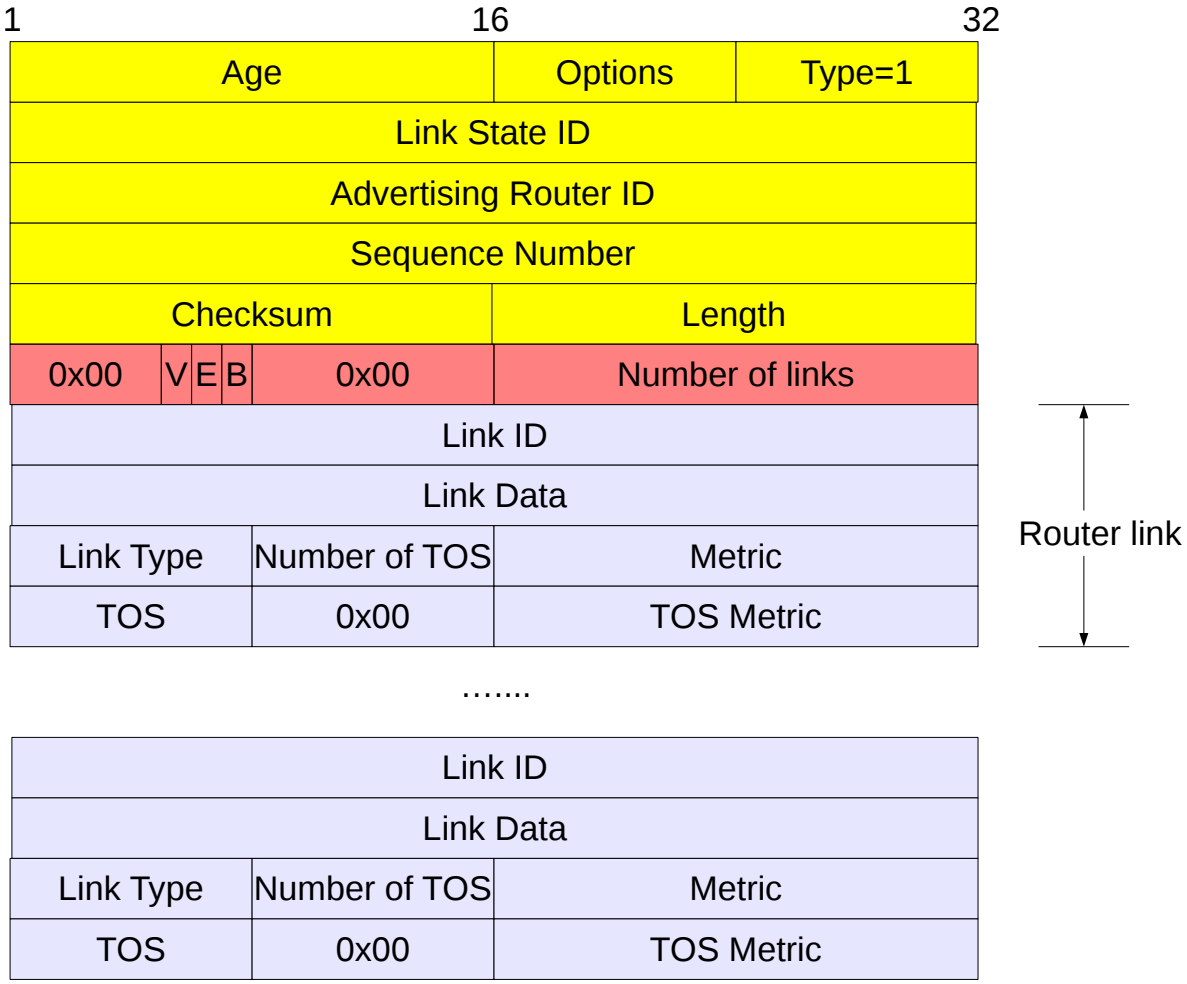
- For LSA type 1 (a router LSA, in which a router describes all the IP subnetworks to which it is attached and all its neighboured routers) the `LinkStateID` field contains the router id of the originating router.
- For LSA type 2 (generated by a DR, describing the IP subnetwork for which it is the DR) this field contains the DRs IP address on the subnetwork on which it is the DR. This is not necessarily identical to the router id.
- For LSA type 3 (generated by an area border router, to summarize networks from one area into another) it contains the IP address of the advertised network.

The usage of the `LinkStateID` field in the further LSA types is not described here.

3.5.2 Router LSA (LSA `Type=1`)

Router LSAs are generated by every OSPF router for each area it belongs to, and these are respectively flooded into the area the router LSA is associated with. In this LSA a router includes all directly attached IP subnetworks and neighboured routers that **belong to the same area** the router LSA is generated for, and for these it includes information like the cost to reach them. It also includes information about itself. If an OSPF domain is made up of one single area consisting only of point-to-point links then router LSAs are the only ones ever transmitted.

The layout of a router LSA is shown in the following figure:



The router LSA starts with three flags:

- `V`-bit (virtual bit): this is set if the router is an end-point for one or more virtual links for which the current area is a transit area.
- `E`-bit (external bit): set when the originating router is an AS boundary router.
- `B`-bit (border bit): set when the originating router is an area border router.

The NumberOfLinks field specifies the number of **router links** that this LSA includes – a router must include router links for all its attached links / subnetworks (belonging to the same area) into its router LSA, and possibly one router link describing itself.[8] Note: **the originating router has to report all its attached subnetworks / all router links (belonging to the same area) in the same LSA**, sharing one common LSA header.

One **router link** entry consists of the following fields:

- The LinkType field indicates the type of link:
  - LinkType=1 is a *point-to-point link*, like for example a serial connection between two routers. Often, the link itself is **not** an IP subnetwork, i.e. it does not have its own IP network address and the two routers attached to the point-to-point link do not have IP addresses on such a link! However, it is nonetheless legal to assign a separate IP network address to a subnetwork formed by the point-to-point link.
  - LinkType=2 is a *link to a transit network*. A transit network is an IP subnetwork (usually of broadcast or NBMA type) to which other OSPF routers are attached, and which hence has a designated router.
  - LinkType=3 is a *link to a stub network*. A stub network is an IP subnetwork to which no other OSPF routers are attached. Note that broadcast or NBMA networks can be stub networks.
  - LinkType=4 is a virtual link.
- The Link ID and Link Data fields broadly describe the other end point of the link, together with auxiliary information. They are used in conjunction with the LinkType field as follows:

LinkType	Description	LinkID	LinkData
1	Point-to-Point link	Neighbors router-id	for point-to-point links having an IP subnetwork address: Interface IP address of originating router; for p2p links without IP address: interface Index
2	Link to transit network	Interface IP address of DR	Interface IP address of originating router
3	Link to stub network	IP Prefix / network address	Netmask
4	Virtual link	Router-id of virtual link end-point	Interface IP address of other virtual link end-point router

- The Metric field is the actual link weight or metric of this link. The field is 16 bits long and contains an unsigned integer. Note that the OSPF specification does not prescribe any specific interpretation of the cost values, but individual vendors can choose otherwise (for example, CISCO routers often assign link weights based on their bitrate).
- The fields Number of TOS, TOS and TOS Metric are no longer relevant, they have only been included for backward compatibility. Note that the field Number of TOS indicates how many 32-bit words (each made up of the fields TOS, 0x00 and TOS Metric, see Figure) are contained. When Number of TOS is zero, then no such field is present.

[8] See the Examples in Section 3.5.5, but this is not universally done by all implementations, e.g. the ospfd that is part of the quagga distribution appears to not do this.

Examples will be shown in Section 3.5.5.

One of the more important details pertains to the following question: suppose we are a router attached to a broadcast or NBMA network, and we are not the DR of this network. Do all the other routers on the network have to be mentioned with their own router link in a router LSA? The answer is no. The rule is that each router in such a network only includes a router link for the network as a whole (this router link, which will be reported as a link with `LinkType`=2, contains information about the IP prefix of the network and the DR), but does not list all the other routers in the network individually. This is done in another type of LSA, the network LSA. The network LSA (discussed next) is generated only by the DR and lists all the routers attached to the network.

Excursion (TOS fields)

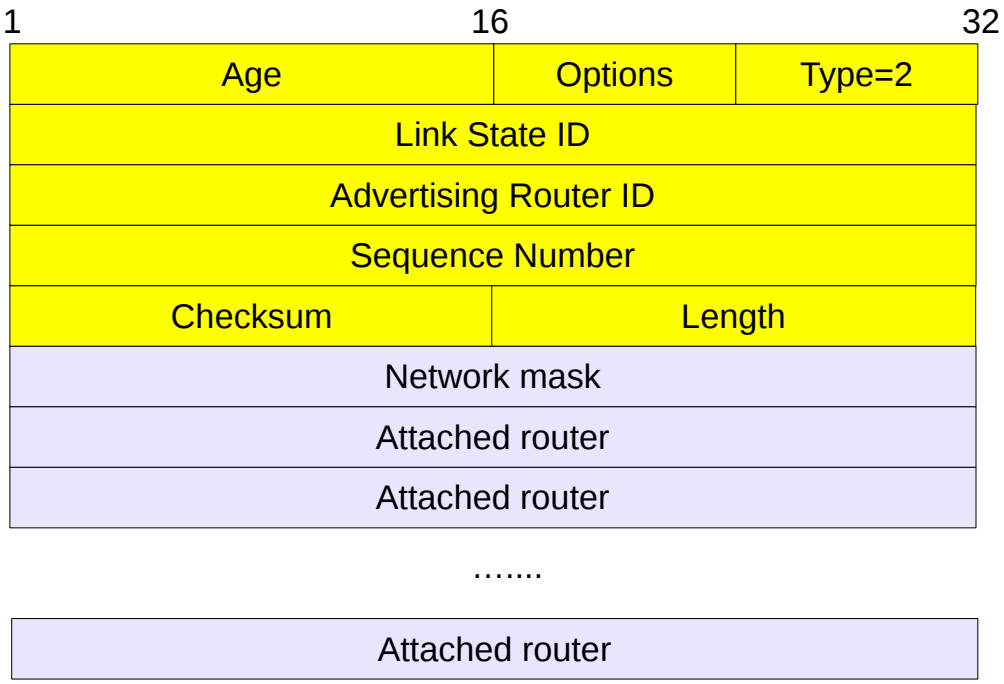
The `TOS` and `TOS Metric` fields in an individual router link are related to the `TOS` (Type Of Service) field in the IPv4 header. This IPv4 header field is nowadays either obsolete or used for different purposes, there are only very few routers in the wild which support the original IP TOS idea.

The idea itself actually makes some sense: it is sometimes important for an IP router to give different forwarding treatment to different types of applications (keyword: Quality-Of-Service, QoS). Packets belonging to a voice-over-IP conversation are time-critical and require quick and reliable forwarding. In contrast, packets making up an email or longer file transfer (like a software update) are not time-critical. The idea of the TOS facility was to classify the packets into different forwarding treatments (like for example "delay sensitive"), to have routers sort these packets into different queues and serve the different queues accordingly – for example a router could always give priority to a queue full of delay-sensitive packets and process other packets only when that queue is empty. Or time-critical packets could be sent over other routes than the other ones.

The TOS facility failed because it has not been well supported by router vendors and because initially there was no precise understanding or agreement on what the different forwarding classes and their associated forwarding behaviours should be. Nonetheless, the need to have some more meaningful QoS support in the Internet has lead to the creation of the differentiated services and integrated services frameworks [28], [18], [8] , [6], [29], [4].

3.5.3 Network LSA (LSA `Type=2`)

A network LSA is generated for every broadcast- or NBMA-type IP subnetwork X that is also a transit network (i.e. to which indeed several OSPF routers are attached). It is generated by the designated router (DR) for network X and describes all the other routers attached to X. The network LSA is only flooded within the same area the subnetwork X belongs to. The format of a network LSA is shown in the following figure:



The `LinkStateID` field of the LSA common header does not give the router id, but instead gives the originating routers IP address on IP subnetwork X. The `Network mask` field of the network LSA gives the network mask for subnetwork X, and, together with the contents of the `LinkStateId` field, any other router can calculate the network address of X. The remaining entries (`AttachedRouter`) list the router id's of all the other routers attached to subnetwork X.

3.5.4 Network Summary LSA (LSA `Type=3` or `Type=4`)

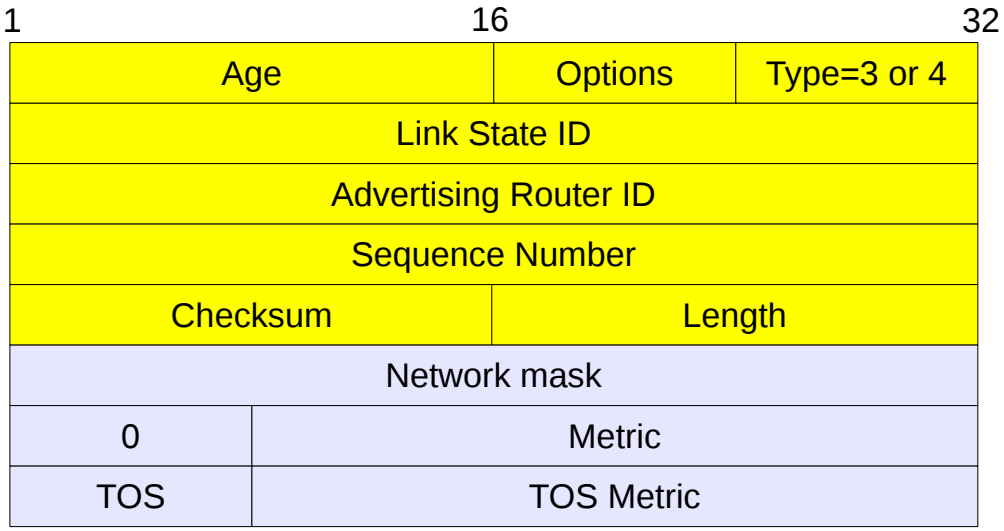
Network summary LSAs are originated by area border routers (ABR), i.e. routers which have interfaces towards at least two different areas, for example an ABR between the core area 0 and a low-level area A. Network summary LSAs play a key role in the hierarchical routing approach of OSPF, and are used for two different purposes:

- The ABR advertises an IP subnetwork or an aggregated IP prefix belonging to a low-level area A into the core area and all core routers, so that information about prefixes of A is available in the core (and subsequently in other low-level areas as well).
- Conversely, the ABR advertises any IP prefixes / subnetworks belonging to the core area or learned from other ABRs through their summary LSAs (which they have flooded into the core area) into its low-level area A (again using flooding), so that routers in A are aware of destinations outside of A.[9] Note that it is not necessary to advertise IP prefixes external to area A into area A, as the ABR can advertise itself as a default router into area A (by advertising IP prefix `0.0.0.0/0` through a network summary LSA into area A). However, when a low-level area has several ABR's it can make sense to have each of them advertise all external IP prefixes (i.e. prefixes from the core area or other low-level areas) explicitly so that internal routers can pick the shortest route.

[9] As the core area also houses AS border routers, it is also possible that the ABR advertises external routes learned from an ASBR into a low-level area. But this is not common.

Note that a network summary LSA can be of either LSA `Type=3` or `Type=4`. The former (`Type=3`) is concerned with IP prefixes / subnetworks internal to the OSPF domain, whereas `Type=4` is used when AS border routers are involved. We will not discuss the latter any further.

The format of a network summary LSA is shown in the following figure:



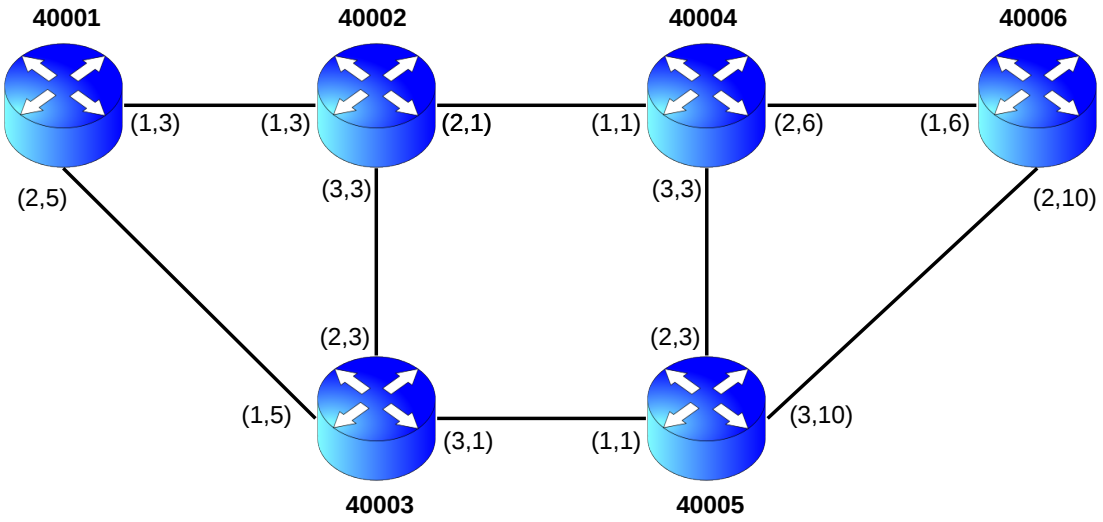
The `LinkStateID` field contains the network address of the IP prefix to be advertised, and the `Network mask` field the corresponding network mask. The `Metric` field gives the cost or weight of reaching the advertised IP prefix through the ABR. The `TOS`-related fields are not relevant to us.

3.5.5 Example Networks and their LSAs

In this section we will look at different networks and for each network we will discuss the shape of the link-state databases and the LSAs that are being exchanged.

A Single-Area Point-to-Point Network

Consider the following network (taken from [24, Sec. 4.3]):



All the links in this network are point-to-point links (e.g. serial lines) and are **not** IP subnetworks. So, none of the interfaces of any router has an IP address, instead the routers simply number their interfaces as $\{1, 2, \ldots\}$ (similar to the way some Linux hosts number their Ethernet interfaces as `eth0`, `eth1` and so on). The router ids are given by the numbers `4000x`. An individual router interface is marked by a pair (x, y) where x is the interface number with respect to the closest router, and y is the link cost. All the routers are in the same area 0 and there are no further routers or end hosts. None of the routers is an AS border router or an area border router, all are just ordinary individual routers. Note furthermore that in this example all link costs are symmetric, but this does not have to be true in general.

As there are no broadcast- or NBMA networks and no further areas, the routers will only send router LSAs. Each router will generate one router LSA describing all its links, and the overall link-state database will then consist of six router LSAs. Let us construct a new router LSA originated by router `40001` (which will be wrapped into one LSU OSPF packet carrying just one LSA, compare Section 3.4):

- The `Age` field will be set to `Age=0`, as the router generates a fresh router LSA.
- The `Area-Id` will be zero, as all routers are in the core area.
- The `Type` field is set to `Type=1`, indicating a router LSA.
- The `LinkStateID` is set to `40001`, i.e. to the router id of the originating router.
- The `AdvertisingRouterID` field is also set to `40001`, as this is the router which originates the router LSA.
- The `Sequence number` and `Checksum` fields are set to some values that are of no concern here, the `Length` field gives the length of the entire router LSA in bytes.
- Remember that the next fields in a router LSA are three flags (`V`, `E`, `B`, see Section 3.5.2) followed by the `Number of links` field. In this particular scenario the router is a vanilla router and none of the three flags is set. The `Number of links` field shows a value of three, i.e. there are three router links: one for each of `40001`'s neighbours and one for `40001` itself.
- We will go through the router link for neighbour `40002` (ignoring the `TOS`-related fields):
  - The `LinkID` field is set to the router id of the neighboured router, i.e. to the value `40002`.

  - The `LinkData` field is set to the number of the interface which router `40001` uses to get to router `40002`, which is its local interface number 1.
  - The `LinkType` field is set to 1, indicating a point-to-point link.
  - The `Metric` field is set to 3.
- The second router link towards neighbour `40003` is reported in a similar fashion, with `40003` in the `LinkID` field, interface number 2 in the `LinkData` field and a `Metric` of 5.
- To announce itself, router `40001` adds a third router link as follows:
  - The `LinkID` field is set to its own router id `40001`.
  - The `LinkData` field is set to `255.255.255.255`.
  - The `LinkType` field is set to 3, indicating a stub network.
  - The `Metric` field is set to 0.

Problem 3.5.5 (Further router LSAs)**.**

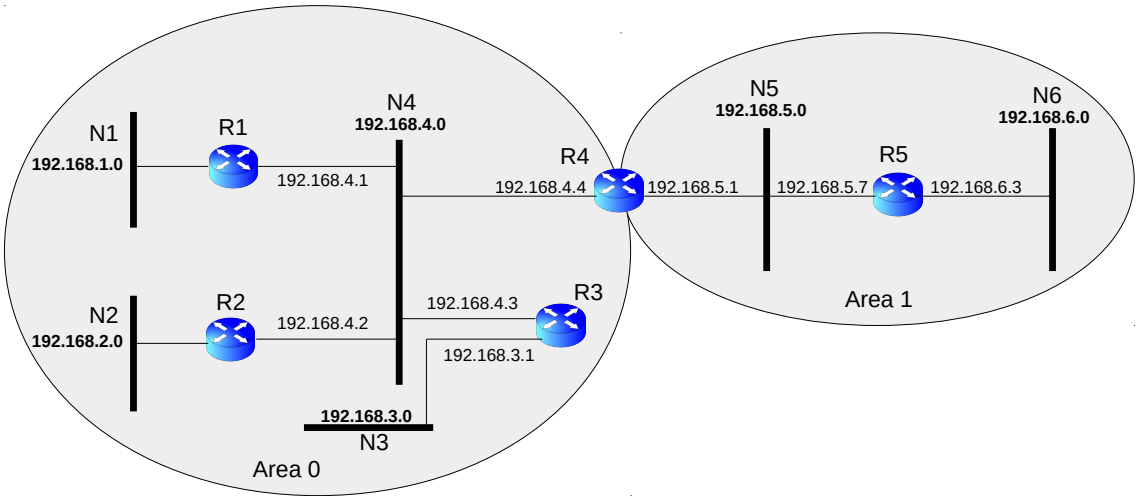
- Give the router LSA for router `40004`.
- Give the router LSA for router `40006`.

Ignore the `Sequence number`, `Options`, `Checksum`, `Age` and `TOS`-related fields.

Problem 3.5.6 (Length of a router LSA)**.**
Suppose a router in a purely point-to-point network as above has n neighbours. In the absence of any `TOS` support, what is the total length in bytes of the generated router LSA? Give an expression. Refer to Section 3.5.2.

A More Involved Example

Consider the following example OSPF domain:



This example has a number of additional features. First, there are two areas, area 0 and area 1. Secondly, there are a number of broadcast subnetworks (just imagine these as plain Ethernet networks), `N1` to `N6`, and multiple routers are attached to two of these (`N4` and `N5`). For subnetworks `N4` and `N5` router `R4` is the designated router (DR). These broadcast subnetworks all have distinct `/24` network addresses. In OSPF terminology, subnetworks `N1`, `N2`, `N3` and `N6` are **stub networks**, to which only one OSPF router is attached, and furthermore all packets a stub network handles either originate in it or are destined to it, and no other packets pass through. In contrast, subnetworks `N4` and `N5` are called **transit networks**. Note furthermore that router `R4` is an area-border router. All routers choose their router id as the highest IP address of any of its interfaces. For simplicity, all link weights are one. Note that the IP address of `R1` on `N1` and of `R2` on `N2` are immaterial. Furthermore, no address aggregation is applied.

Problem 3.5.7 (What is in the link-state database)**.**
Which router, network and summary LSAs are contained in the link-state database for area 0? Same question for area 1. For the summary LSAs you can assume that no address aggregation is used.

Let us first look at the router LSA generated by router `R5` in area 1 (here shown in a tabular form):

Header field	Value	Comment
Area-Id	1	because R5 is in area 1
Type	1	indicates router LSA
LinkStateID	192.168.6.3	R5's router id
AdvertisingRouter	192.168.6.3	R5's router id
E-bit	0	not an AS boundary router ...
B-bit	0	... and not an area-border router
NumberOfLinks	3	
LinkID	192.168.5.1	IP address of DR R4 for network N5
LinkData	192.168.5.7	R5's interface towards N5
LinkType	2	connects to a transit network
Metric	1	
LinkID	192.168.6.0	IP network number / prefix
LinkData	255.255.255.0	network mask for N6
LinkType	3	connects to stub network
Metric	1	
LinkID	192.168.6.3	R5's router id
LinkData	255.255.255.255	network mask for self router link
LinkType	3	connects to stub network
Metric	0	

Note here that router R5 is not the designated router in network N5, so it only generates a router LSA.

Problem 3.5.8 (Router LSA for router R1)**.**
Give the router LSA for router R1. Does this router generate a network LSA?

Next we consider the router LSA generated by router R4 for area 0. It looks as follows:

Header field	Value	Comment
Area-Id	0	because R4 sends this into area 0
Type	1	indicates router LSA
LinkStateID	192.168.5.1	R4's router id
AdvertisingRouter	192.168.5.1	R4's router id
E-bit	0	not an AS boundary router ...
B-bit	1	... but an area-border router
NumberOfLinks	2	
LinkID	192.168.4.4	IP address of DR R4 for network N4
LinkData	192.168.4.4	R4's interface towards N4
LinkType	2	connects to a transit network
Metric	1	
LinkID	192.168.5.1	R4's router id
LinkData	255.255.255.255	network mask for self router link
LinkType	3	connects to stub network
Metric	0	

Note three things in R4's router LSA: first, since the router LSA is sent into area 0, it only includes information about network N4 (which is in area 0) but not about network N5 (which is in area 1). Secondly, R4 is the designated router for N4, and the LinkID of the router link for N4 has to report the IPv4 address *that the DR*

has within network N4. Thirdly, router R4 does **not** list all the other routers in N4 as router links, this is instead done by the network LSA issued by router R4.

Problem 3.5.9 (Router LSA for router R4)**.**
Give the router LSA of router R4 for area 1.

We next look at the network LSA which router R4 generates for N4 (for which it is the DR). Note that this is only flooded in area 0 as N4 belongs to this area. It looks as follows:

Header field	Value	Comment
Area-Id	0	because R4 floods this into area 0
Type	2	indicates network LSA
LinkStateID	192.168.4.4	IP address of DR for N4
AdvertisingRouter	192.168.5.1	R4's router id
NetworkMask	255.255.255.0	
AttachedRouter	192.168.5.1	Router id of R4
AttachedRouter	192.168.4.3	Router id of R3
AttachedRouter	192.168.4.2	Router id of R2
AttachedRouter	192.168.4.1	Router id of R1

Note that in the list of AttachedRouter's only routers attached to N4, including R4 itself, are listed.

Problem 3.5.10 (Network LSA for router R4)**.**
Give the network LSA of router R4 for network N5. In which area is it flooded?

We also take the opportunity to see network summary LSAs (type=3) in action for the first time. Since router R4 is an area border router, it will:

- send network summary LSAs (type=3) about the IP subnetworks contained in area 1 into area 0, to let routers in the latter area 0 know that the networks in area 1 are reachable through itself (at the indicated cost), and
- conversely send network summary LSAs about the IP subnetworks contained in area 0 into area 1.

Note that router R4 will send one separate network summary LSA for each advertised IP prefix/subnetwork. It is furthermore possible (but not done in the following examples) that R4 is configured to **not** send network summary LSAs about the IP subnetworks of area 0 into area 1, but rather to announce itself as the default router by sending a summary LSA announcing the default route (0.0.0.0/0) into area 1. As an example, we consider a summary LSA sent by router R4 into area 1 advertising network N1:

Header field	Value	Comment
Area-Id	1	because R4 floods this into area 1
Type	3	indicates network summary LSA
LinkStateID	192.168.1.0	IP address of advertised network N1
AdvertisingRouter	192.168.5.1	R4's router id
NetworkMask	255.255.255.0	The /24 network mask of advertised network
Metric	2	Total cost from R4 to N1

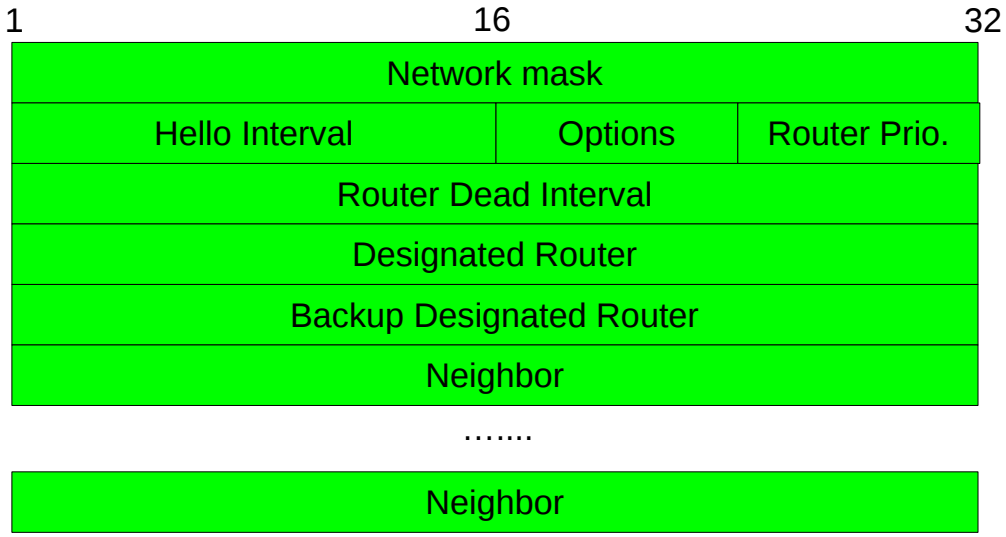
Problem 3.5.11 (Network Summary LSA)**.**
Give the network summary LSA (`type=3`) that router `R4` uses to advertise network `N6` into area 0.

3.6 Adjacencies and Database Synchronization

We describe the method which OSPF uses to establish and maintain adjacencies with neighboure routers (the Hello protocol) and explain how neighbours achieve initial synchronization of their link-state databases.

3.6.1 The Hello Protocol

OSPF routers periodically send `Hello` packets on each of their interfaces, usually every ten seconds. `Hello` packets are sent to the link-local multicast address `224.0.0.5`, aka `AllSPFRouters`. A `Hello` packet (without the common header for all OSPF packets, see Section 3.4) looks as follows:



The fields have the following meaning:

- `Network Mask` is the network mask valid for the interface on which the packet was sent, this is used to check that two routers on the same subnetwork have the same subnetwork configuration.
- `Hello-Interval`: is the nominal period (in seconds) of sending `Hello` packets between two adjacent neighbors – both neighbors must have same value. The default value is ten seconds, but this is configurable.
- `Options`: indicates further capabilities of OSPF router, e.g. which additional LSA types it understands.
- `Router-Priority` determines the routers eagerness to become DR or BDR router on the subnetwork, and this field plays a role in the DR election protocol. If it is set to 0, then the router will not become DR/BDR.

- `Router-Dead-Interval` indicates how many seconds a router will wait at most for a `Hello` packet before declaring the neighbors death.
- `DesignatedRouter` is the IP address of the designated router on the network (non-DR/BDR routers only establish adjacencies with DR/BDR routers). Note that this reflects the *current view* of the sender, and might change during an election process.
- `BackupDesignatedRouter`: IP address of backup designated router.
- `Neighbor` lists all OSPF neighbors (their router-id's) on the subnetwork to which OSPF packet is sent for which the router has received an "acceptable" `Hello` packet during the last `RouterDeadInterval` seconds.

A router X starts sending `Hello` packets after it has been switched on. At that time it has not yet established any adjacency and the `Neighbor` list will be empty. Another router Y receiving a `Hello` packet:

- Checks whether all his own pre-configured parameter values for `Area-ID`, `NetworkMask`, `Authentication`, `HelloInterval`, `RouterDeadInterval`, and `Options` are the same as in the received packet.
- If not, the packet is dropped.
- Otherwise, Y establishes an **adjacency** with X, and Y includes the router id of X in the neighbor list of its subsequent `Hello` packets.
- When X receives a `Hello` packet from Y with X's router id in the `Neighbor` list and an agreeable set of configuration/authentication data, then X knows that the link is bi-directional and also establishes an adjacency to Y. In general, only bi-directional links are considered in routing calculations. If any router finds that one particular link is only reported by one of the involved routers and not the other (in their respective router LSAs), it will not be used.

After router X has established an adjacency with router Y, it expects to receive `Hello` packets from Y periodically (and vice versa). If X receives no `Hello` packet for a time of at least `Router-Dead-Interval` (typical default value is 40 seconds), then X concludes that neighbour Y is dead and will issue a (triggered) router LSA in which the router link to Y is not listed anymore. It will furthermore re-calculate its routes to ensure that Y is not used as a next-hop anymore.

Problem 3.6.1 (Bi-directional links)**.**
OSPF insists that all links which are used in any forwarding path are bi-directional. In contrast, this was not ensured in the first version of the RIP routing protocol (RIPv1).
Find an example of a RIPv1 network where something goes wrong when a link is only uni-directional.

Excursion (Uni-directional links)

Do uni-directional links really occur anywhere?

In wired networks (Ethernet, optical networks) it is indeed extremely unlikely to ever meet one, unless you have faulty connectors or the like, and in these cases usually some repair is carried out.

In wireless networks, however, they actually occur quite often. One of the many reasons for this is that very often the antennas used in wireless systems are not truly omni-directional, but they radiate higher levels of signal power into some parts of space than into others, either by design (directional antennas) or because of obstacles blocking wave propagation. When directional antennas are

used at two stations, they need to be oriented towards each other. If the antenna of one station is oriented towards the other station but the other stations antenna is oriented elswhere, then a uni-directional link can occur. Measurements described in [11] show that in a deployment of many wireless nodes about 15% of the links can be uni-directional. Another reason is the choice of different transmit powers,

3.6.2 Database Synchronization

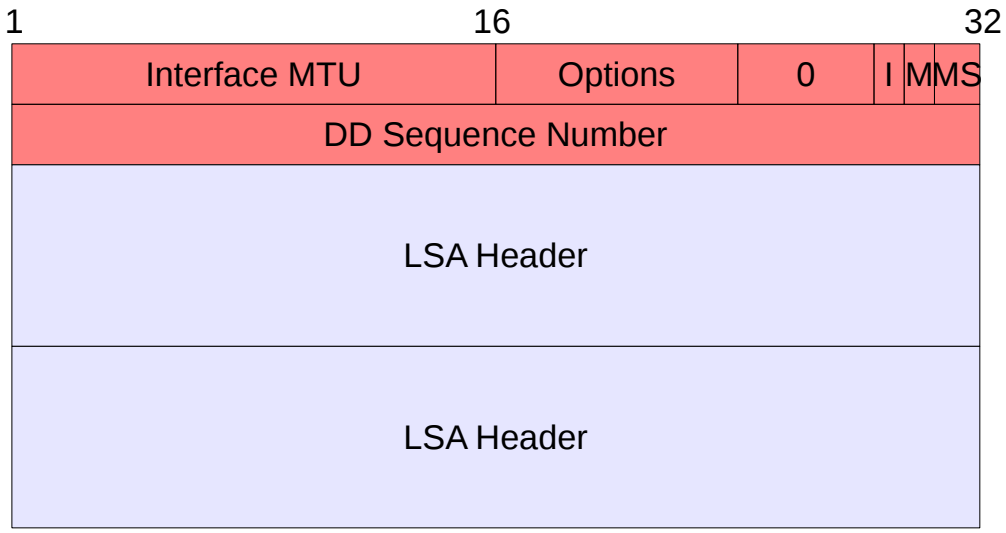
In OSPF (and any other link-state protocol like IS-IS) it is very important that all routers in an area have the same link-state database and calculate consistent, loop-free routes. The amount of time where the databases are not consistent should be as small as possible. As discussed above (Section 3.3.1), two different mechanisms are used to achieve this:

- All link-state changes are flooded quickly.
- A router that has just been switched on will ask neighbours for their link-state databases to avoid the lengthy wait for the periodic LSA transmissions (every 30 minutes). Only when the router has synchronized its link-state database with its neighbours will the router be used as a forwarder.

The second point can be phrased more generally as saying that two neighboured routers must have synchronized databases before they establish any paths through one another.

The database synchronization protocol between two neighboured routers is built on the `database-description`, `link-state-request` and `link-state-update` packet types (see Section 3.4). Broadly speaking, the approach is not to exchange the complete database immediately, but rather to first send summary information about the stored LSAs, and only if one neighbour realizes it misses an LSA (or only has it in an outdated version) will the full LSA be exchanged. This is called a **database exchange**.

Neighbours connected via a point-to-point link always synchronize their databases with each other, whereas in broadcast- and NBMA-type networks all the non-DR nodes only synchronize their database with the DR and BDR. When two neighbours indeed have to synchronize with each other, they start so immediately after they have established that they are adjacent through the Hello protocol. This is also the point in time where they will start flooding LSAs to each other. One of the two routers is designated to become the **master**, the other one will be the **slave**. The master will send one or more `database-description` packets (`Type=2`) to the slave, the format of which looks as follows:

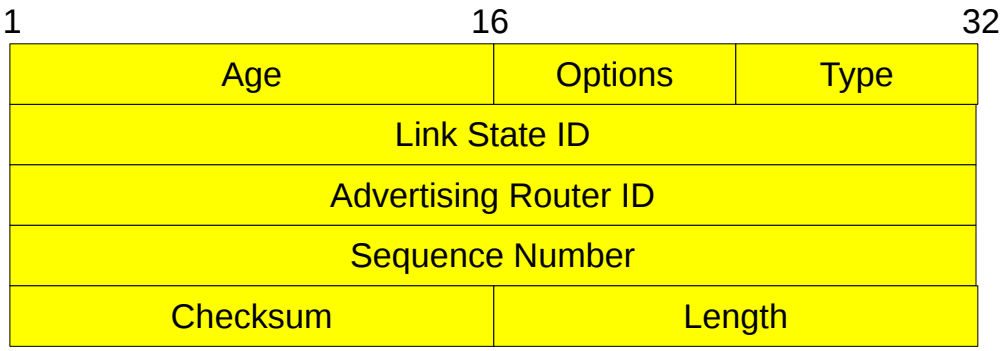


.....

The fields are as follows:

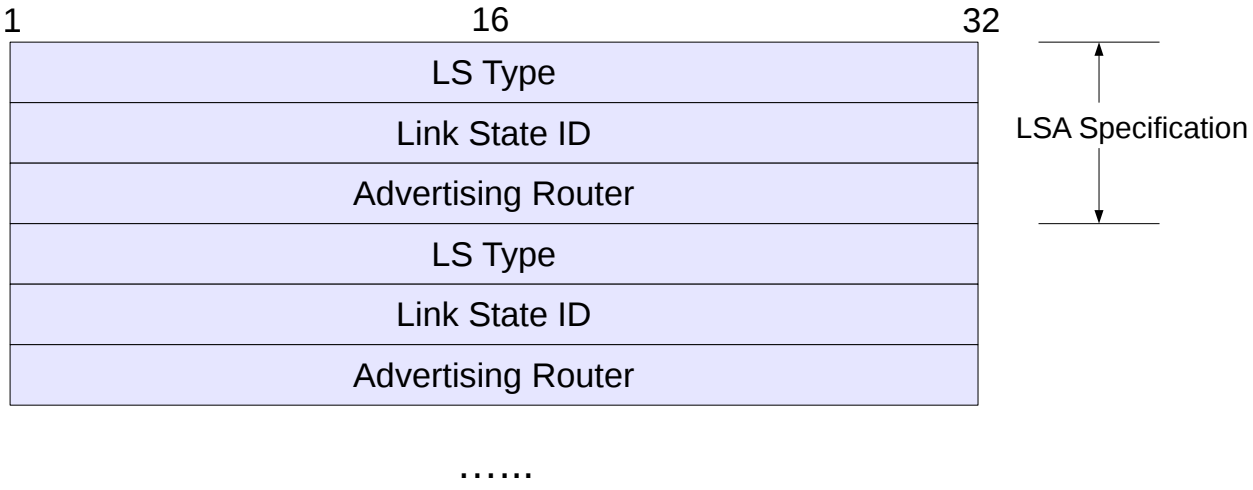
- The `Interface MTU` field gives the maximum link-layer packet size that can be used on the subnetwork / point-to-point link connecting the two routers. This allows to calculate the size of the largest IP datagram that can be sent without invoking IP fragmentation.
- The `Options` field describes router capabilities (e.g. which optional LSAs it understands), it is not relevant for our discussion here.
- The `I` bit is called the **init bit** and is set when this packet is the first in a sequence of `database-description` packets sent by the master.
- The `M` bit is called the **more bit**, and indicates that more `database-description` packets will follow.
- The `MS` bit is called the **master/slave bit**. When set, the originating router has the role of the master, otherwise it is the slave.
- The `DD Sequence Number` is used to number the `database-description` packets, and for the slave to acknowledge their proper receipt (also using `database-description` packets).

The LSA headers are simply the common headers for all the different LSA types, which for reference we repeat here (see Section 3.5.1):



The master sends the LSA headers for its entire link-state database using one or more `database-description` packets. The slave has to acknowledge each packet separately (also using a `database-description` packet) and then the master sends the next one. So, the protocol that the two follow for exchanging the link-state database is similar to the alternating-bit (or send-and-wait) ARQ protocol you have discussed in COSC264.

Once the slave has received the entire database description from the master, it can check which LSAs it misses or has only in outdated versions. The slave will then specifically request these using one or more `link-state-request` packets (which have `Type=3`). These look as follows:

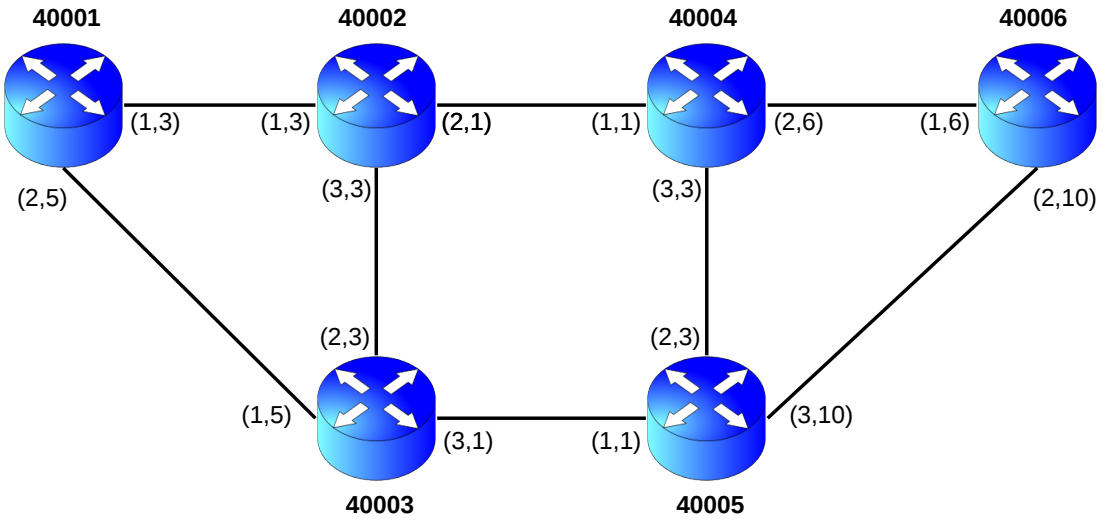


i.e. they consist of a number of LSA specifications. One such LSA specification only consists of the `Link State Type`, `Link State ID` and `Advertising Router` fields from the common LSA header and does notably not include other fields like the `Age` or `Sequence Number`. The slave asks to get the *most recent instance* of the specified LSA and does not ask for any specific instance. When the master receives `link-state-request` packets it responds by sending the requested LSAs (by wrapping them in `Link-State-Update` packets).

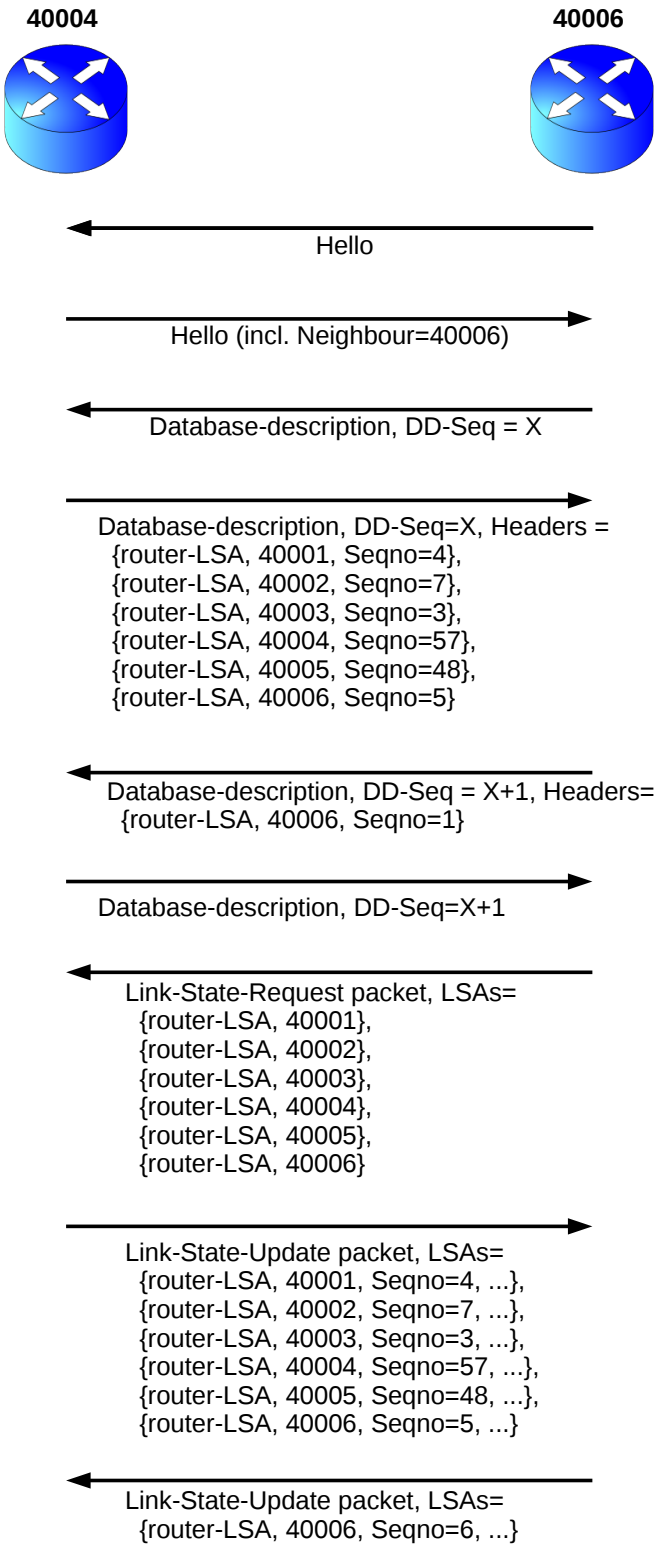
In the moment the slave router has received all requested LSAs, it declares itself as being fully operational and starts to include the master in its router LSAs.

An Example

We will go through an example of the database synchronization procedure (taken from [24, Chap. 4]). Consider again the point-to-point network from Section 3.5.5:



and assume that after the network has converged router `40006` crashes. Its neighbours `40004` and `40005` realize this and drop `40006` from their router LSAs, so these two and all other routers re-calculate their routing table and eliminate all routes involving `40006`. However, the router LSA of router `40006` remains in the link-state databases for up to an hour. Router `40006` re-starts well before this time and will enter the database synchronization procedure, for example with neighbour `40004`. A possible dialogue between these two routers is shown in the following figure:



With the first `database-description` packet (sent as slave), router `40006` starts the database synchro-

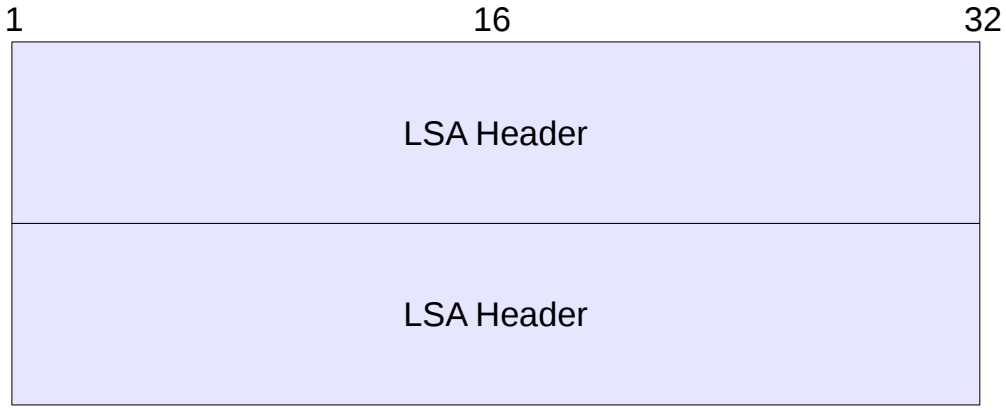
nization process and proposes an initial `DD Sequence Number=X`. The master router `40004` responds with a (master) `database-description` packet with `DD Sequence Number` X, in which it lists all the LSAs it has (only router-LSAs for this point-to-point network), including the old router-LSA from router `40006` with sequence number 5 (this is the sequence number from the LSA common header). Router `40006` acknowledges this with another `database-description` packet, now with the next `DD Sequence Number = X+1` and including a new router LSA for itself, in which it uses the LSA sequence number one (assigned to itself freshly after restart). This packet is in turn acknowledged by `40004` by sending an empty `database-description` packet with `DD Sequence Number = X+1`. Following this, router `40006` asks for all the router LSAs (using a `link-state-request` packet) and gets them (through a `link-state-update` packet). Finally, router `40006` has learned that apparently the last LSA sequence number it sent before the crash has been 5, so it picks the next sequence number 6 and issues a fresh router LSA with the new sequence number.

3.7 Flooding

The reliable (and fast!) flooding of LSAs is a crucial element of link-state protocols to keep the link-state databases synchronized. There is one key property of link-state protocols (in particular OSPF) that helps to make flooding potentially fast: an intermediate router who just has received an LSA does not need to perform any calculations on the new LSA before it continues the flooding process, it is sufficient if new route calculations are carried out in parallel or shortly after the router further floods the LSA.

How is the flooding process being made reliable and at the same time reasonably efficient? Suppose one router wants to generate a new instance of an LSA, which needs to be wrapped into a `link-state-update` (LSU) packet. The router starts the flooding process by sending the LSU packet to all of its interfaces that are attached to the area the LSA shall be flooded into (remember that flooding is limited to areas). When the interface is towards a point-to-point link, the packet will be sent directly to that. When the interface is towards a broadcast/NBMA network, it will be sent to one of the two link-local multicast addresses `AllSPFRouters` or `AllDRouters`, depending on the circumstances.

Before explaining the flooding process in more detail, we show the format of the `link-state-acknowledgement` packet type (`Type=5`), which plays a role in the process:



So, with a `link-state-acknowledgement` message the receiving router confirms a list of individual LSAs.

A neighbour which receives the LSU packet first checks the checksum in the OSPF packet header and throws the packet away if this is wrong. Otherwise, the router goes through the following list of steps (incomplete, leaving out a number of special cases):

- It checks for each LSA in the LSU packet its individual checksum and the LSA type. If the checksum is wrong or the LSA type is unknown to this router, the LSA is dropped, the remaining ones are processed further and we will call them the *surviving LSAs* in the remainder.
- Check each of the surviving LSAs whether it is not yet contained in the routers database or whether it is more recent (sequence number, age) than what the router has currently in its database. Such an LSA will be called a *new LSA* in the following.
- Create a `link-state-update` (LSU) packet including all the new LSAs and flood this out on all interfaces which:
  - belong to the same area as the new LSAs belong to (which is indicated in the `Area ID` field in the header of the OSPF packet containing the LSU), and
  - are different from the interface on which the incoming LSU packet has been received.

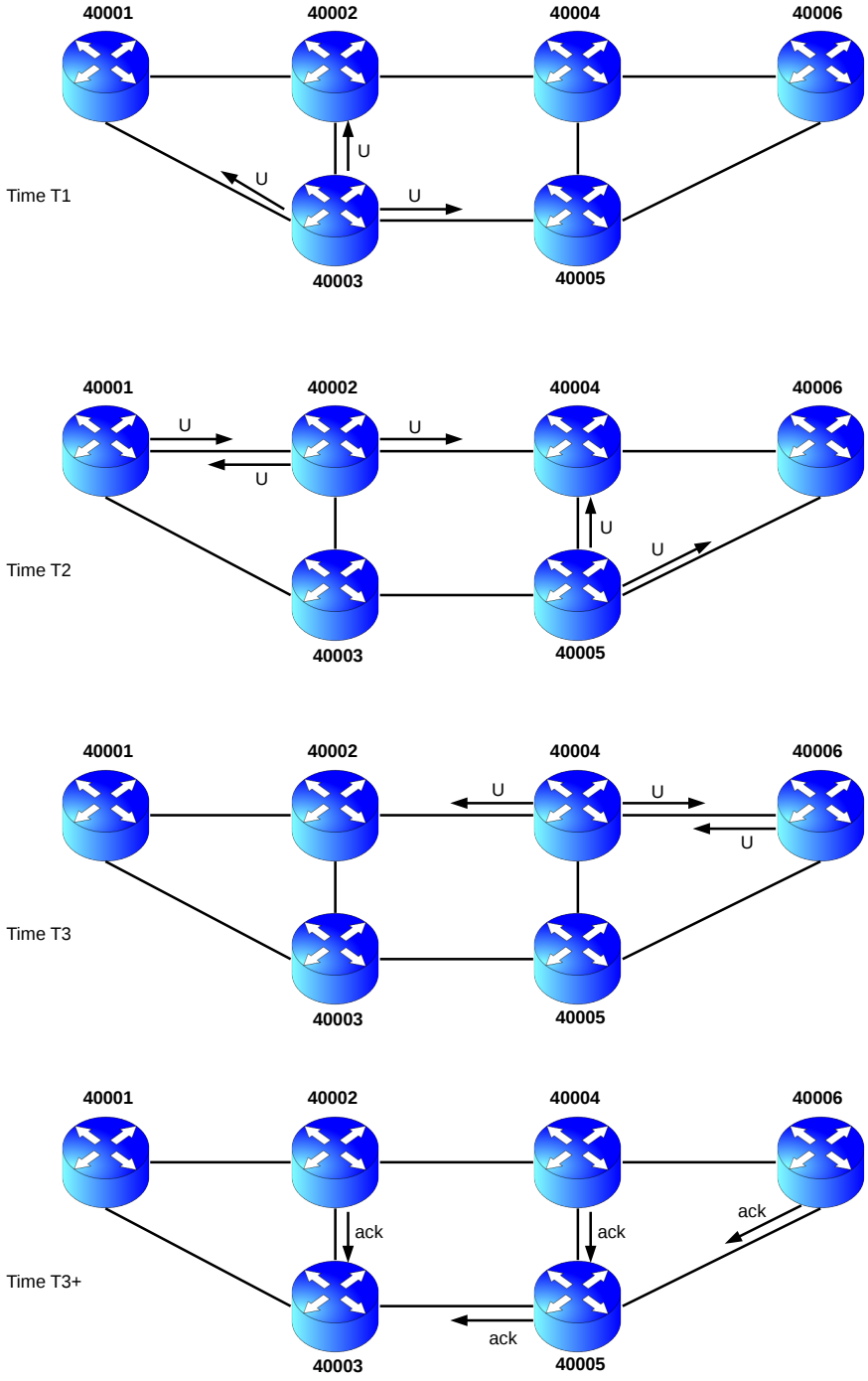
  This way the flooding process iterates through the entire area.
- When a new LSA has not been in the link-state database before, it is added, otherwise it replaces the previous instance of the same LSA. If necessary the routing calculation is started.
- All the surviving LSAs are acknowledged, with some exceptions. This can happen either by sending a `link-state-acknowledgement` packet listing all the headers of the surviving LSAs, or by including the LSAs into the next own LSU packet. If a `link-state-acknowledgement` packet is used, then this is sent out to the interface on which the triggering LSU has been received. The `link-state-acknowledgement` packets can have a random delay, which is useful for different reasons:
  - If several routers on a broadcast subnet receive and process a LSU at the same time and send an acknowledgement packet immediately, they can collide with each other on the transmission medium, and the collision resolution procedure might incur time and packet losses (in particular on wireless networks). Having every router choose a random delay allows to smooth out the acknowledgements.
  - During the waiting time further LSUs can be received and the router can actually aggregate the LSA acknowledgements for all the different LSUs into one `link-state-acknowledgement` packet. It may also happen that during this waiting time the router will send a LSU of its own, into which any new LSAs would be included. This new LSU generated by the receiving router could then be used as an implicit acknowledgement for the originating router.

  One of the exceptions to the rule that all surviving LSAs are acknowledged is when the link-state database has just very recently (within a few seconds) been updated with another instance of the LSA that is, but for minor differences in the `Age` field, essentially the same instance. This can happen if a router receives the same LSA over different interfaces. In this case the new copy of the LSA is dropped as it has already been added before.
- The router originating a LSU packet will periodically re-transmit the LSU until it has received acknowledgements for all the LSAs from all its neighbours it has established an adjacency with (and which belong to the same area).

In a broadcast- or NBMA-type network it can happen during the flooding process that a non-DR/BDR router X receives a LSU packet from another network. Router X will then send this LSU to the DR and BDR router in the network by sending it directly to their configured addresses (in an NBMA network) or to the `AllDRouters` link-local multicast address in a broadcast network. Both the DR and BDR then receive the LSU packet, and the DR is responsible for sending it to all the other non-DR/BDR routers in the subnetwork – either by sending it to each other router individually (in an NBMA network) or by sending it to the `AllSPFRouters` multicast address (in a broadcast network).

An Example

We illustrate the flooding process by an example (taken from [24, Chap. 4]), again using the point-to-point network introduced in Section 3.5.5).



At time $T1$ router `40003` generates a new router LSA for itself, wraps this into a LSU packet and sends this to all of its interfaces. At time $T2$ the direct neighbours of `40003` re-flood the LSU on all interfaces but the one on which they have received the LSU, and at time $T3$ the flooding process is completed.[10] Note that at time $T3$ router `40004` only sends the LSU to router `40002` and `40006`, but not to `40005`. This is because of the

[10]Note that in reality the neighbours would not normally re-flood the LSUs at *exactly* the same time, due to differences in processing times etc.

implicit assumption that at time $T2$ router `40004` first processes the LSU received from `40005` before the one from `40002` and thus treats the interface towards `40005` as the incoming interface, whereas the LSA received from `40002` is dropped since it is received/processed within very short time after the arrival of the LSA from `40005`. After time $T3$ all the routers start the process of sending acknowledgements.

Note that this example is relatively simple as we are only using a point-to-point network and all routers follow the same behaviour. Things are more complicated in a broadcast- or NBMA network, as the different routers behave differently, depending on whether they are the DR or not.

3.8 OSPF Areas

In large OSPF domains with a large number N of different IP subnetworks scalability problems arise. Without sub-dividing the domain into areas, each router would have N different destination networks in its routing / forwarding table, which requires a lot of memory and can slow down routing table lookups. Furthermore, each router would have to keep a link-state database for the entire domain, and after each change in link costs or metrics the actual routing computation would have to operate on a large network graph.

The concept of OSPF areas as sketched in Section 3.3.2 introduces **hierarchical routing**. An OSPF domain is broken down into areas, and a router strictly internal to some area only knows the detailed topology information (IP subnetworks, links, routers) for its own area – represented in a link-state database. On the other hand, area border routers keep information about at least two areas – they have one link-state database per area they are attached to. To reach any subnetwork outside its own area, a router will have to turn to an area-border router, which will then deliver packets via the core area.

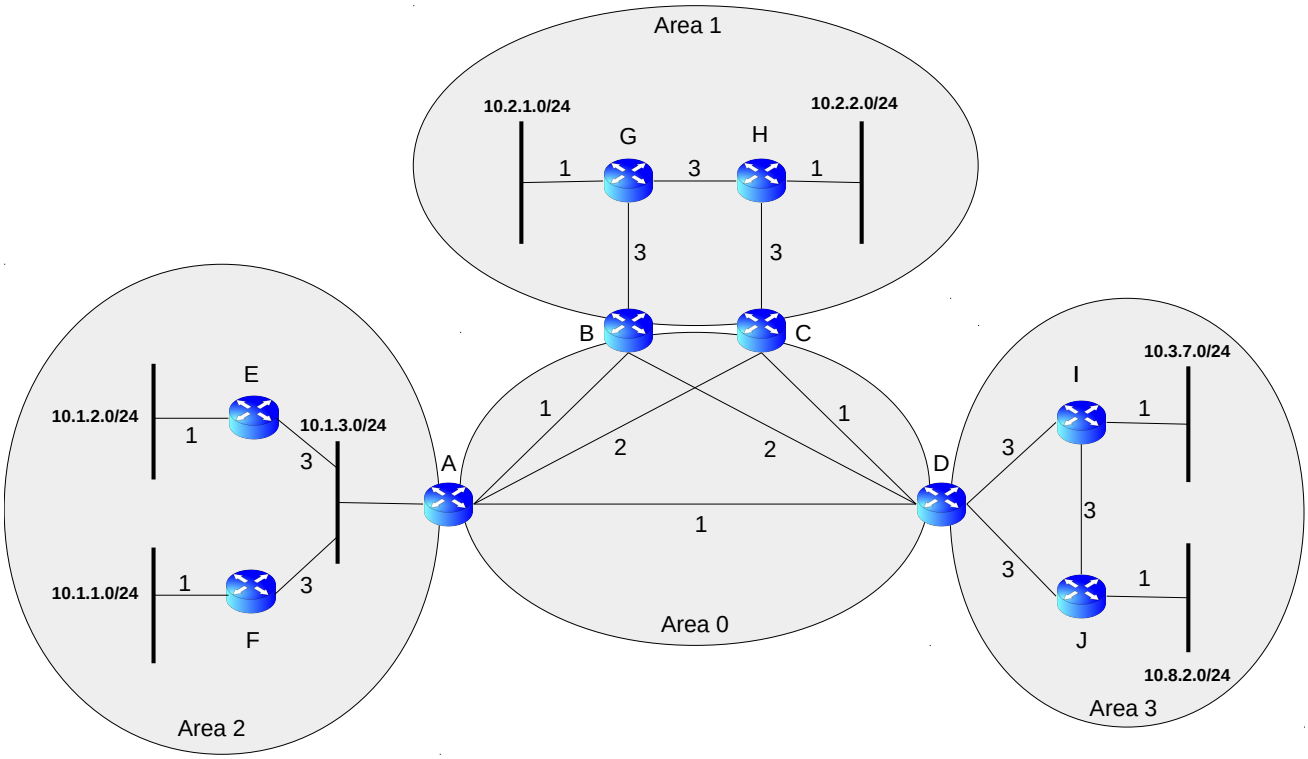
Problem 3.8.1 (Tradeoff in hierarchical routing)**.**
With hierarchical routing as introduced here we indeed have the benefit that the routers within an area can have smaller routing tables (less memory, faster lookup). What is the price to pay?

An OSPF area is assigned a unique 32-bit identifier (area id) and can consist of several IP subnetworks connected by routers. There is one special area with area id 0, called the **core area**. Each area has its own link-state database, maintained by routing- and network-LSAs generated by the routers of this area and reflecting the precise physical topology of the area. These two types of LSAs are never flooded outside an area.

Areas are connected through area-border routers and the allowed area topology is a star topology: all areas have to be attached (through area-border routers and by either physical or virtual links) to the core area. The area-border routers provide information about a low-level area to the core area, and in turn forward routing information received from the core area to the low-level areas they are attached to. For this purpose network summary LSAs are used (see Section 3.5.4).

3.8.1 An Example

Consider the following example OSPF domain (from [24, Chap. 6]):



In this figure the thicker lines correspond to Ethernet networks, and their network address / network masks (all `/24`) are given in the figure. Thinner lines are point-to-point links or attachments of routers to Ethernets. Furthermore for each point-to-point link / network attachment the routing metric is indicated, and point-to-point links do not have an IP network address. There are four areas, including the core area 0.

Problem 3.8.2 (Link-state databases of different areas)**.**
Please give the number of router- and network LSAs and the originating routers for each of the four areas in the figure.

Notice that the networks `10.2.1.0/24` and `10.2.2.0/24` in area 1 share the first 16 bits of their network address, and the area border routers of this area (routers `B` and `C`) are configured to *aggregate* these two networks into one single network with a shorter prefix and advertise only that to the other areas. In this particular case, routers `B` and `C` will advertise the network `10.2.0.0/16` in summary LSAs. Similarly, router `A` will advertise the aggregated prefix `10.1.0.0/16` outside of area 2. In contrast, router `D` will have to advertise the two networks `10.3.7.0/24` and `10.8.2.0/24` separately in its summary LSAs, as these two cannot be aggregated into a `/16` prefix.

Be careful to distinguish *aggregation* from *summarization* here – sometimes people also use the term summarization for what we call aggregation (of IP prefixes). Aggregation refers to advertising a group of IP subnetworks or IP prefixes under a more general prefix, just as router `B` is doing in our example. Aggregation is completely independent of any routing protocol. In contrast, summarization is specific to OSPF and refers in a narrow sense to the fact that an IP prefix (aggregated or not) is put into a summary LSA, and more generally to the fact that `B` advertises just the cost of this prefix without giving away the detailed topological fine structure of area 1.

A summary LSA sent by router `B` into the core area could look as follows:

Header field	**Value**	**Comment**
`Age`	0	because `B` is originating router
`Area-Id`	0	because `B` floods this into the core area
`Type`	3	indicates summary LSA
`LinkStateID`	10.2.0.0	Network address of advertised prefix
`AdvertisingRouter`	b.b.b.b	router `B`'s router id
`NetworkMask`	255.255.0.0	advertising it as `/16` prefix
`Metric`	7	Cost of 7

As you can see, router `B` indeed advertises the aggregated network `10.2.0.0/16` (indicated in the `LinkStateID` field for the network address and the `NetworkMask` field for the netmask) at a cost of 7. This cost value is the worst-case cost by which router `B` can reach any host in the two aggregated networks `10.2.1.0/24` and `10.2.2.0/24` (namely, a host in the latter network).

This summary LSA is flooded in area 0, and the other border routers `A`, and `D` advertise the prefix `10.2.0.0/16` into the areas 2 and 3, respectively. Note that router `C` advertises the same prefix `10.2.0.0/16` into the core area. When router `D` wants to calculate a route towards prefix `10.2.0.0/16` it has two different options (since it has received two different summary LSAs from `B` and `C`):

- through router `C`, which router `D` can reach at cost of 1, or
- through router `B`, which router `D` can reach at cost of 2.

Therefore, router `D` will choose the advertisement from router `C`, which gives the lower total cost of $1 + 7 = 8$ towards the prefix. If the core area is larger and routers `B` and `C` were no direct neighbours of router `D`, the latter would still be able to calculate its best cost to the prefix `10.2.0.0/16`, as it knows from the area 0 link-state database its own best costs (and next hop) towards routers `B` and `C`. Once router `D` has calculated its best cost (8) to prefix `10.2.0.0/16`, it will advertise this into area 3 using a summary LSA, so that all the other routers in area 3 will know how to reach this prefix.

In this example network we can also see that summarization and aggregation can lead to sub-optimal routing decisions. As explained above, router `D` will send every packet for which the destination address matches the prefix `10.2.0.0/16` to router `C`. However, for the particular destination address `10.2.1.7` it would be more cost-efficient to go through router `B` (convince yourself!). But router `D` has insufficient information to do this.

3.8.2 Stub(by) Areas

Normal OSPF areas enjoy all the features of OSPF (some of which we have not discussed in detail), in particular, the routers in a normal area can learn routes towards many different destinations in other areas or even external to the OSPF domain, coming from other interior routing protocols or from BGP.

Sometimes, however, it is not desirable to have this wealth of information available in a router, in particular when the router is rather resource-limited. In a **stub area** a much simpler setup is used. In particular, information about prefixes from other areas or AS-external routes are not flooded into the stub area (e.g. through summary LSAs), but instead the area border router advertises itself as the default router into this area.

Problem 3.8.3 (How to become a default router?)**.**
How can an area border router of a stub area advertise itself as a default router? Only use the mechanisms discussed in this chapter.

Stub areas cannot be the end-points of virtual links and also cannot host any AS boundary routers.

There are different types of stub areas, e.g. totally stubby areas, not-so-stubby areas and others. These differ in the actual amount of summary information that area-border routers advertise into the area.

3.8.3 Area Sizes and Other Considerations

One important design question is how large an area actually should be, i.e. how many routers should an area have. There is no hard and fast rule about this, but typical practice appears to be to restrict an area to between 10 and 100 routers [9]. Another design guideline is that the amount of control traffic in an area (router, network and summary LSAs, `Hello` packets) should not take away more than 5% of transmission bandwidth in heavy-load cases (e.g. when a freshly restarted router receives the link-state database from a neighbour), better only 1% on average for the periodically generated LSAs.

Some other benefits and effects of sub-dividing a network into areas include [24]:

- Increased robustness: routing failures within one area do not propagate to other areas, leaving them un-affected.
- Hidden prefixes: an area-border router can be configured to *not* advertised some internal prefixes of a low-level area into the core area, making these prefixes not accessible from the outside. This can be used to protect these prefixes.

Chapter 4

BGP

In this chapter we give a short introduction to the BGP routing protocol, where BGP stands for **Border Gateway Protocol**. BGP is at least as complex as OSPF, but we will not go into the same level of detail and our discussion here is much more sketchy.

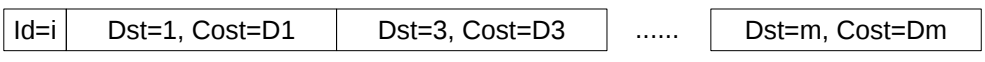
BGP is an exterior routing protocol, i.e. it enables routing across different autonomous systems (AS). BGP (or more specifically BGP version 4, specified in RFC 4271) is practically the only exterior routing protocol that is used in the Internet. The behaviour of BGP has global impact on Internet routing, and whenever you read about a spectactular routing hiccup in the Internet you can be almost certain that the BGP protocol has something to do with it.

Some references for BGP include its specification [27], the books [15], [21, Chap. 8] and [13]. A discussion of BGP security issues can be found in [5].

Recall that in Section 3.1 we have spent some time to discuss the weaknesses of distance-vector protocols. Interestingly, BGP does not build on the link-state approach but uses a variant of distance-vector protocols which comes under the name of **path vector routing**, and which eliminates some of the problems that other protocols from the DV family like RIP have. We first introduce the idea of path vector routing and then turn our attention towards some of the basics of BGP's operation.

4.1 Path Vector Routing

Recall that in distance-vector routing protocols there is no network-wide flooding, but routers only ever send routing information to immediately neighboured routers. More specifically, a router sends to its neighbour a table (or vector) listing all the destinations it knows of (by giving their identification, e.g. IP network address) and its own best cost to reach this destination. As an example, a distance vector message sent by some router i could look as follows:



Router i sends such a message to all of its neighbours. It does not transmit any information about individual links and a distance-vector message reveals almost nothing about the overall topology of the network. We have also seen (Section 3.1 and [20]) that distance-vector protocols are prone to problems like routing loops or the count-to-infinity problem. We have looked in detail at an alternative class of routing protocols, namely

link-state protocols (and in particular OSPF).

However, all is not lost with distance-vector routing and BGP follows a method called **path-vector routing**, which is an extension of distance-vector routing. Here we discuss path vector routing in a generic network made up of nodes and links, not with specific reference to exterior routing protocols and the AS topology on which they operate.

In path-vector routing a router i sends to each immediate neighbour a message which includes:

- The identification of i
- For each destination d known to i:
  - the identification of d,
  - i's best cost towards destination d, and
  - **the entire path** towards d which has just the indicated costs.

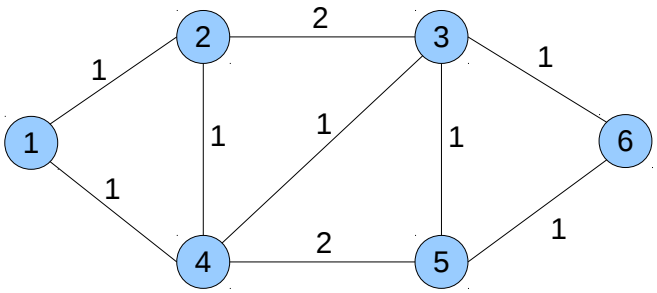
When station i sends such a message to a neighbour k, node i prepends its own identification i to the paths towards all destinations it lists. When k receives such a message, it processes it in a way that is very similar to message processing in distance-vector protocols: k checks whether its current best cost route towards d can be improved upon by going through i, where the cost of going through i is calculated as the sum of the cost of going directly from k to i, plus the cost indicated by i for getting to d. When k decides that indeed its best path to d goes through i, then node k:

- stores the new best cost for destination d, and
- calculates its path towards d by prepending its own address to the path (to d) it received from i.

This way, the information about a given destination d propagates through the network and each router advertising it prepends its own identification to a path, so that the path gets gradually built up.

Problem 4.1.1 (Routing loops)**.**
How can this approach help to detect and remove routing loops or deal with the count-to-infinity problem? (Compare also Section 3.1)

We will go through an example (based on [21, Sect. 3.5]). Consider the network shown in the following figure:



Suppose that immediately after startup and discovery of neighbors, node 3 will send the following path vector message to node 2:

Destination	Cost	# nodes in path	Path
3	0	1	(3)
4	1	2	(3,4)
5	1	2	(3,5)
6	1	2	(3,6)

Suppose furthermore that immediately after initialization and subsequent discovery of nodes 1 and 4, and before it receives anything from node 3, node 2 has the following path table:

Destination	Cost	Path
1	1	(2,1)
4	1	(2,4)

Now node 2 receives the path vector message from node 3 and updates its path table as follows (its direct path to 4 is cheaper than the one through node 3!):

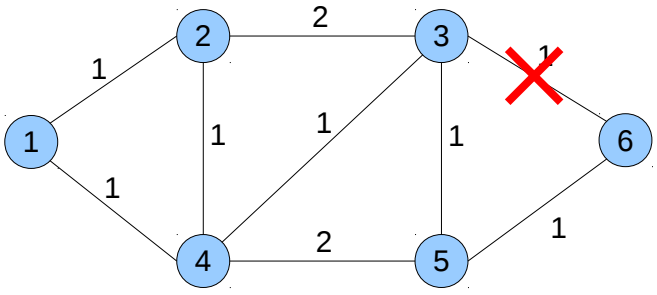
Destination	Cost	Path
1	1	(2,1)
3	2	(2,3)
4	1	(2,4)
5	3	(2,3,5)
6	3	(2,3,6)

When updating a table entry, node 2 adds its own id to received paths. It then sends the following path vector message to 1, 3, 4:

Dst=1 ; Cost=1; #nodes=2; nodes=2,1	Dst=2 ; Cost=0; #nodes=1; nodes=2	Dst=3 ; Cost=2; #nodes=2; nodes=2,3
Dst=4 ; Cost=1; #nodes=2; nodes=2,4	Dst=5 ; Cost=3; #nodes=3; nodes=2,3,5	Dst=6 ; Cost=3; #nodes=3; nodes=2,3,6

This is repeated at other nodes, entire paths are disseminated.

Next we consider a link failure, indicated in the following figure:



We assume that the network is in a converged state. The path table entries from all nodes to node 6 are shown in the following table:

From node	To destination	Cost	Path
1	6	3	(1,4,3,6)
2	6	3	(2,3,6)
3	6	1	(3,6)
4	6	2	(4,3,6)
5	6	1	(5,6)

Now the following sequence of events unfolds:

- Time t_0: link between 3 and 6 fails.
- Time t_1: node 3 notices and sends `unreachable` message concerning link (3,6) to its neighbors 2, 4 and 5.
- Time t_2: when receiving this message, node 2 removes its entry for destination 6, since its path to node 6 uses link (3,6).
- Time t_3: node 2 sends an unreachable-message to its neighbors 4 and 1.
- Time $t_4 \approx t_3$: node 4 will effect similar actions as node 2 after receiving node 3's unreachable-message.
- Time t_5: when node 5 receives node 3's unreachable-message, it checks that it can reach node 6 without using link 3 - 6 on its path and sends a path-vector message to node 3, which computes new path to node 6 (through 5).
- Time t_6: After receiving the path vector update message from node 5, node 3 sends again a path vector update message to its other neighbors 2 and 4, indicating its new path to node 6.
- Time t_7: nodes 2 and 4 will update their path vector tables (What is the end result? What will node 4 do?).

This example shows that failing links are treated in a very sensible way.

As an extension of the path vector approach, a router can for a given destination d store all the paths that his neighbours advertise to it, so that backup paths are available quickly.

Problem 4.1.2 (Scalability of path vector routing)**.**
The Internet has thousands of autonomous systems (AS) and path-vector routing (as part of BGP) is applied to the AS topology. Qualitatively compare the scalability of path vector routing to that of a link-state protocol (think OSPF with just one huge area). Consider in particular that BGP does not use periodic updates (argue why BGP might have dropped these) and transmits only changes, when they occur.

4.2 BGP Overview

The BGP protocol is used between neighbored AS to exchange routing information. For the following remember that each AS is identified by a 16- or 32-bit number, its AS identifier or **AS number**, abbreviated as ASN.

BGP is based on the path vector approach, and routing information is exchanged between neighboured AS (through AS border routers, which in BGP parlance are called **BGP speakers**). Two AS are considered as

neighboured when they have peering relationship, either using private peering or a public exchange point (see Section 2.2.4). Each involved AS has one or more **border routers** (BR) running the BGP protocol (i.e. they are BGP speakers), and neighboured BRs establish a **communication session** over which they exchange BGP messages. BGP is encapsulated as payload into TCP, and a border router listens on the well-known TCP port 179. With this, border routers do not necessarily have to be physical neighbours but can be separated. When a BGP session (or more precisely: the underlying TCP connection) breaks, then a BR removes any information obtained from the lost BGP peer and does not use it any further.

Excursion (BGP, OSPF and Layering Principles)

If you recall the layering principles from COSC264, then one of the ways you can tell that a protocol belongs to layer N is if its protocol messages (or Protocol Data Units – PDUs) are encapsulated as payload into PDUs of layer $N - 1$.

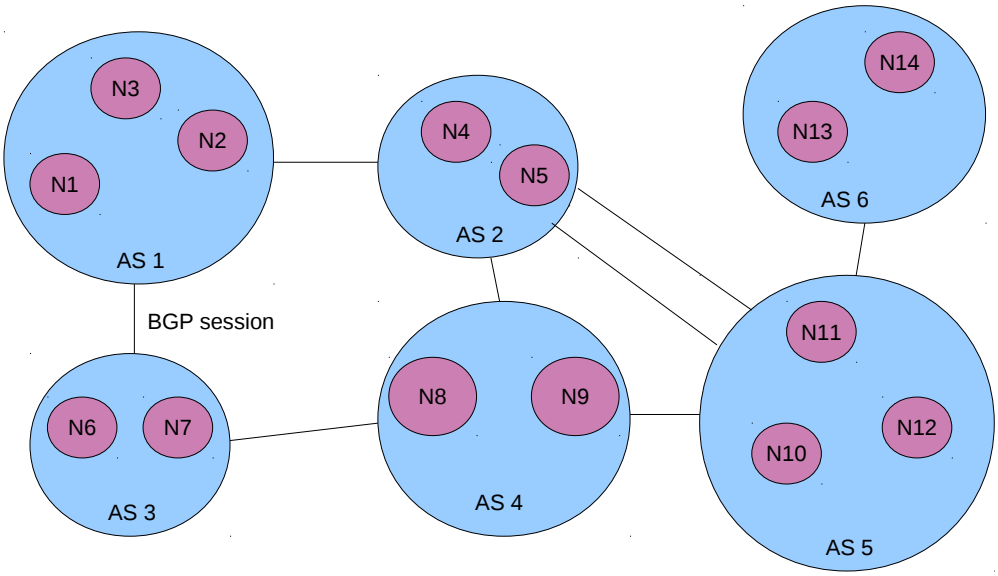
Now BGP is encapsulated into TCP and with respect to the TCP/IP reference model this would make BGP an application layer protocol. Similarly, OSPF is embedded into IP which technically would make OSPF a transport layer protocol. But both BGP and OSPF exchange routing information, which really is only relevant to the network layer and "morally" belongs to it. This is one example where reality and layering principles are harder to reconcile.

The routing information exchanged between two BGP speakers includes the following essential information:

- IP prefixes (including network masks).
- The AS path to an IP prefix, which is made up of ASNs.
- A number of **attributes**, which play a key role in BGP and tell for example how BGP (in the originating AS) has actually learned about the prefix, for example through an interior routing protocol or through manual configuration.

It is important to re-iterate the following point: BGP applies path vector routing with entire AS as its "nodes". The "links" between nodes are assumed to have unit weights and represent peering relationships (see Section 2.2.4). Thus the resulting routes minimize the number of AS hops between two endpoints, not the number of routers!!

The following example gives a crude illustration of how path vector routing is applied in BGP. We consider the following network:



where we have six different AS and 14 different IP prefixes `N1` to `N14`. Each AS owns some subset of these prefixes. In this example, the shortest AS path between AS1 and AS6 is AS1 $\mapsto$ AS2 $\mapsto$ AS5 $\mapsto$ AS6. AS1 learns about this the following way:

- AS6 announces via BGP to AS5 that it is home of networks N13, N14. Specifically, AS6 sends the following record to AS5: $(\text{AS6}|\text{N13}, \text{N14})$
- When AS5 forwards this record further to AS2, the record becomes $(\text{AS5}, \text{AS6}|\text{N13}, \text{N14})$
- In the end, AS1 receives $(\text{AS2}, \text{AS5}, \text{AS6}|\text{N13}, \text{N14})$.

Excursion (Expressing policies)

BGP allows that an AS entry in a record is repeated, e.g. AS1 could send a record $(\text{AS1}, \text{AS1}, \text{AS1}|\text{N1}, \text{N2}, \text{N3})$ to AS2 while sending record $(\text{AS1}|\text{N1}, \text{N2}, \text{N3})$ to AS3.

In this case AS4 would receive two different records: it receives $(\text{AS2}, \text{AS1}, \text{AS1}, \text{AS1}|\text{N1}, \text{N2}, \text{N3})$ from AS2 and $(\text{AS3}, \text{AS1}|\text{N1}, \text{N2}, \text{N3})$ from AS3.

Through this "hack" AS1 can influence the routes that would be chosen in, say, AS4 for any packets destined to network N1. In fact, AS4 would choose a route through AS3 for these packets as fewer AS hops are involved. This is for example useful when AS1 prefers to receive packets from AS4 through AS3 instead of AS2, perhaps due to a more advantageous pricing.

4.3 Some BGP Details

First, a note on terminology: when we talk about a route, then we actually mean an AS route (or AS path) towards an IP prefix, e.g. $(\text{AS2}, \text{AS5}, \text{AS6}|\text{N13}, \text{N14})$ describes the AS route AS2 $\mapsto$ AS5 $\mapsto$ AS6 to IP prefixes N13 and N14. Note that, while not shown here, such an AS route also includes a range of attributes.

4.3.1 BGP Sessions

As stated above, BGP routers having a communication session with each other do not need to be immediate physical neighbours. So, these routers need to be *configured* with the identity of their peer router.

To establish a BGP session, the two involved routers (BGP peers) first establish a TCP connection between them. Using TCP ensures that routing information is reliably transmitted. When a BGP router has several BGP peers, it needs to establish several TCP sessions in parallel. Once the TCP session has been established, the BGP peers start exchanging BGP messages with each other, see Section 4.3.2. In particular, they identify their peers, agree on parameters and establish trust through authentication mechanisms.

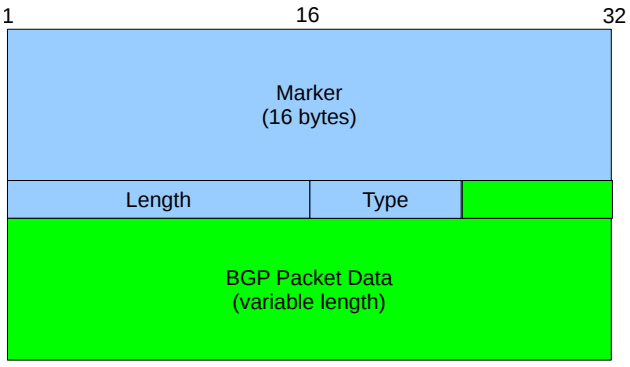
Once the initial steps (TCP connection establishment, mutual negotiation of parameters, authentication) have been carried out and the session has been properly established, the speakers start exchanging actual routing information. Immediately after session establishment a BGP speaker sends **all** the routes it knows to its peers, after that it only sends updates. **There is no periodic exchange of whole routing tables**.

The BGP peers monitor the health of the underlying TCP session by frequently exchanging particular BGP messages called KEEPALIVE when there are no other messages to be sent.[1] If a BGP peer receives no message at all for a certain amount of time (larger than the KEEPALIVE period), then the BGP session is considered broken and dis-established. The BGP session is also dis-established when the underlying TCP protocol reports an abrupt termination of the connection, and both involved speakers remove all information and all routes they have learned through this session. A BGP speaker might attempt to re-establish the BGP session – in that case there would again be a complete exchange of routing tables at the beginning.

A key aspect of BGP (which we do not discuss in any detail) is that BGP supports **policy-based routing**: A BGP speaker might deliberately decide to throw away some routes it learned from its peer, it might decide to not advertise some routes it knows about to its peer, it might modify the AS path and so on.

4.3.2 BGP Messages and Common Header

There are four basic BGP messages (and further optional ones). All BGP messages share a common header, shown here:

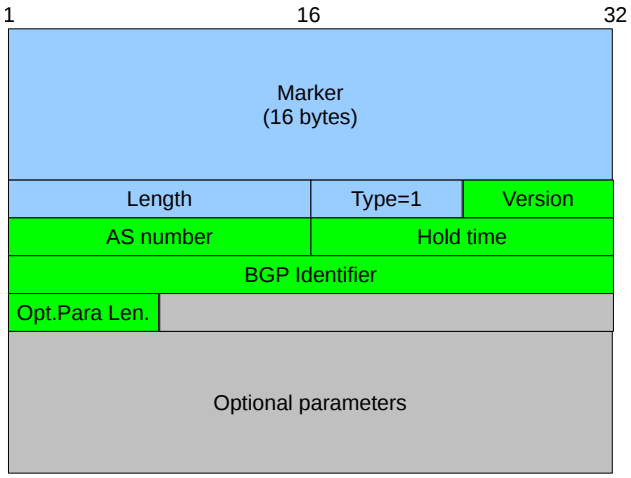


The header includes the following fields:

[1] TCP normally does not have a keepalive feature by which an otherwise idle but established connection is frequently checked for continuing availability. Some TCP versions support such a feature though, and two TCP peers can agree to use it (through certain options exchanged during TCP connection setup). However, as this feature is not available in all TCP connections, the BGP protocol implements this functionality itself.

- The `marker` consists of 16 bytes `0xff` (unless authentication is used, in which case it consists of authentication-related data).
- The `length` field gives the total length of the BGP message in bytes.
- The `type` field specifies the type of BGP message. There are four basic types and some further optional types.

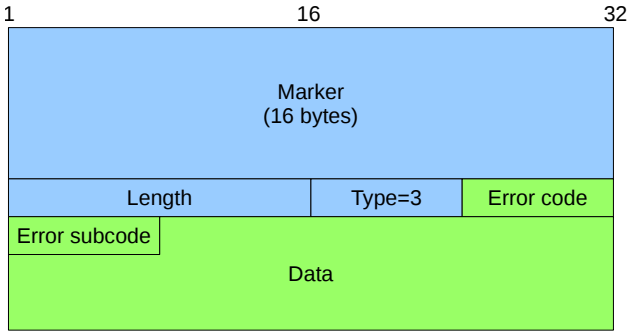
The first type of message (`Type=1`) is the OPEN message, which is only used during establishment of a BGP session to help both speakers agree on some parameters. It looks as follows:



The `version` field specifies the BGP version number. If the two BGP peers do not agree on this, the TCP connection is immediately closed. The `AS number` field specifies the AS number (ASN) of the sending speaker. The other BGP peer needs to be configured with the ASN of the sending peer, and if the value is wrong then the TCP connection is closed. The `HoldTime` field specifies the maximum time which is allowed to pass without receiving any UPDATE or KEEPALIVE message before the session is declared to be broken. Both peers send an OPEN message during session establishment, and the smaller of the two exchanged `HoldTime` values is accepted as the actual hold time. Finally, the `BGPIdentifier` indicates the router-id of the sending speaker. To each BGP speaker such an identifier is assigned – it has to be unique, but there is otherwise no particular requirement on what the identifier should be. A BGP speaker is configured with the identifiers of its peers, and if a received OPEN message does not contain a known identifier, it is dropped and the TCP connection is closed.

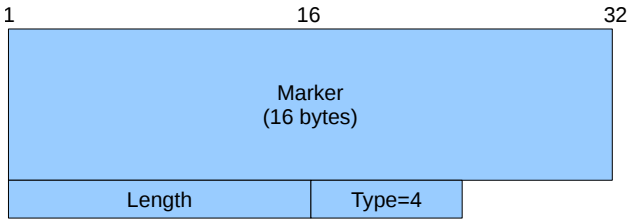
The second type of message (`Type=2`) is the UPDATE message type, which carries actual routing information. We discuss this in more detail in Section 4.3.3.

The third type of message (`Type=3`) is the NOTIFICATION message type. When one of the BGP speakers detects an error (e.g. a mal-formed BGP message sent by the peer), it sends a NOTIFICATION message including an error code and then closes the TCP connection (and thus the BGP session). The message looks as follows:



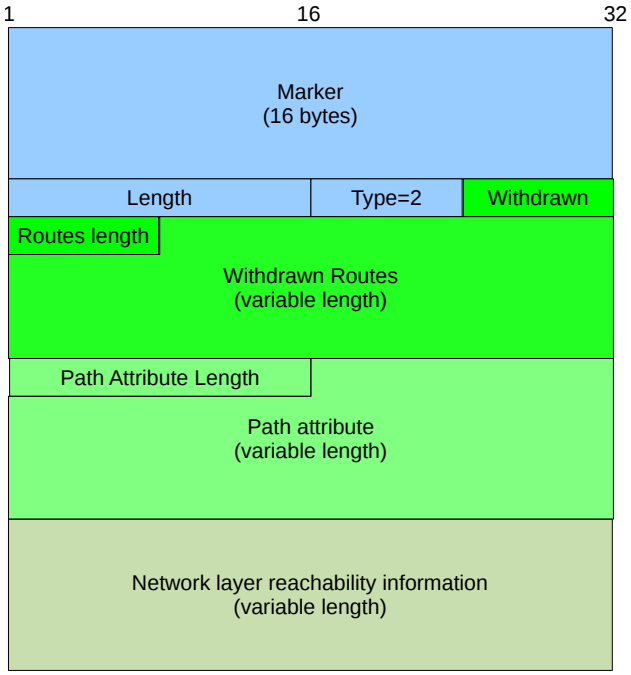
The `Error code` and `Error sub-code` fields together specify the error condition.

Finally, the fourth type of message (`Type=4`) is the KEEPALIVE message. These messages are sent when no other message (in particular an UPDATE message) has been sent for some time to check the liveness of the TCP connection. They are sent periodically (default: every 30 seconds, generally one third of the value of `HoldTime`) in a session to verify liveness of the peer BGP speaker. It carries no further data beyond the common header:



4.3.3 `Update` Messages and Route Selection

We now look at some of the aspects of the operation of BGP. We start by looking at the format of the UPDATE message (`Type=2`):



UPDATE messages are sent by a speaker whenever it becomes aware of a change in reachability information, e.g. when a new optimal route has been determined for an IP prefix, or when a prefix is not reachable anymore. When speaker A notices that an IP prefix P is not reachable anymore (or it just wants to stop forwarding packets to P), it informs peer B with a route withdrawal (there can be several in one message) – a withdrawn route is specified as (netmask/prefix length, IP prefix) pair. When speaker B's best route to P is going through A's AS, then it removes the route and informs further speakers in other AS. The network layer reachability information field (NLRI) is a list of reachable IP prefixes and to all of which the given path attributes apply. An NLRI entry is for example generated when speaker A has just learned about a prefix P, or the AS path to P has changed.

The key elements of the UPDATE message are the NLRI and the path attributes (PA) field. The PA field consists of one or more attributes, and each attribute is specified as a (type, length, value) record: the first entry specifies the attribute type, the second the length of the data in the value field, and then the value field follows. All the attributes listed in the PA field apply to **all** prefixes listed in the NLRI field. Some important attributes include:

- `AS-PATH`: this specifies the AS path or route through which all of the prefixes listed in the NLRI are reachable. Every BGP speaker sending an UPDATE to a peer prepends its own AS number.
- `ORIGIN`: identifies how an IP prefix has been injected into BGP. For example, a BGP router could have learned a prefix from an interior routing protocol running in its AS, or a prefix could have been manually configured.
- `NEXT-HOP`: this specifies the router in the speakers AS that routers in the neighboured AS should actually use to forward packets to in order to reach an advertised prefix. This can be the same as the speaker, but can also be a different machine.
- `MULTI-EXIT-DISCRIMINATOR`: two AS can actually be connected by more than one pair of routers. For the sake of example assume that two AS are connected by two such pairs. In some situations a destination prefix might be more preferably reached by one of the two router pairs. This attribute can be used to express such a preference.

There are several attributes specified, which differ in whether they are well-known (i.e. understood by every BGP speaker) or not, or whether they are transitive (i.e. they will be forwarded to BGP peers in other AS) or not – well-known attributes are always forwarded. When a BGP speaker has prefixes to which different attribute sets apply, then these must be sent in different UPDATE messages.

A BGP speaker (which can have sessions with different peers at the same time) receives routes from its peers, filters out some of the incoming routes which for some reason it does not want to make available to its own AS or re-advertise to others, and re-advertises routes to other BGP peers. To support this, a BGP speaker keeps different databases:

- **Adjacent-RIBs-In** database: stores for each BGP peer the complete AS routing information (prefixes and their attributes) that this peer has sent and which the receiving speaker has chosen not to filter out. The abbreviation RIB stands for "Routing Information Base". To carry out input filtering on a received UPDATE message, the BGP speaker performs for each contained IP prefix the following (and further) steps:
  - It filters out IP prefixes that are not supported.
  - It filters out private IP prefixes.
  - It filters out routes with private AS numbers.
- The **Loc-RIB** (or "local RIB") database is the own AS-level routing table which for each IP prefix stores the best cost and AS path leading to it.
- The **Adjacent-RIBs-Out** database stores for each BGP peer the information that is actually advertised to them with UPDATE messages. This can in general be a subset of the Loc-RIB, as the speaker might choose not to advertise some routes.

With this setup, incoming UPDATE messages are filtered and stored in Adjacent-RIBs-In. Furthermore, a BGP speaker might also participate in an interior routing protocol like OSPF running in its own AS and learn IP prefixes this way. A key step is to use all this input information to select the best routes to a given destination and populate the Loc-RIB. This is a complex decision process, and we only mention a few of the steps involved:

- If import filtering indicates an unwanted prefix, discard it.
- If the IP prefix belongs to the own AS, the speaker will prefer routes determined by its own interior routing protocol over routes learned through BGP.
- If there are several AS routes available to the destination prefix, keep the ones with the fewest number of AS hops listed in the `AS-PATH` attribute. If only one route survives, then take this.
- If there is more than one route with the same (smallest) number of AS hops, look at the `ORIGIN` attribute, prefer routes where the prefix has been learned from an IGP over manually configured prefixes.
- If there are still several candidate routes, choose the one having the highest preference (compare for example the `MULTI-EXIT-DISCRIMINATOR` attribute).

Further criteria based on additional path attributes exist.

Part II

Labs

Chapter 5

Labs

The labs for this module center around configuring routing protocols under Linux and performing various experiments to "vivisect" their operation. You will work under Linux on a set of virtual machines which are configured as routers or end hosts, and which are interconnected by virtual networks. These virtual networks look to the virtual machines like perfectly normal Ethernet networks. You will not have a GUI available on the virtual machines but rather work on the Linux command line, so you are expected to be familiar with this.

You will use the `quagga` routing suite (see `www.quagga.net`), which is a key component of a range of commercial linux-based router products.[1] You will find a pdf version of the `quagga` users manual on the learn website for this course, and you will find some basic information about `quagga` and other networking tools (`ping`, `traceroute`, `ifconfig`, `route`) in Appendix B. This appendix repeats lab material from COSC 264 in a slightly revised manner, you are expected to be familiar with the operation of these networking tools. Make sure you understand how `ping` and `traceroute` are implemented, i.e. which IP/ICMP facilities they use.

It is important that you read this chapter **before** the actual lab, since otherwise you will risk to have insufficient time to get everything done!! However, you should not expect that the notes below will contain *everything* you need – in fact, you are expected to figure out several details on your own.

This chapter contains all the information and exercises about the lab work for this module. I expect that you will need between two or three weeks to complete the exercises. There is no need to finish a certain amount of exercises in any one week, but at the end of this module you should have completed the labs.

5.1 Lab Setup

You will be working with a number of virtual machines (VM) under `Oracle VM VirtualBox`, interconnected by a number of virtual networks. More precisely, you will be getting ten VMs. In each of these runs the Linux operating system, and there are two different types:

- Some VMs (`alpha`, `beta`, `gamma`, `delta`, `epsilon`) will run as **end hosts**. These are only connected to one network and are not supposed to actively participate in any routing.

[1] However, many of the Internet routers, especially on the backbone, use proprietary hardware architectures and operating systems. Examples include CISCO routers (and their IOS operating system) or Juniper routers.

- The other VMs (`christchurch`, `auckland`, `dunedin`, `hamilton`, `oamaru`) will run as **routers**. They are connected to multiple networks.

Each machine runs an instance of the Ubuntu 16.04 Server LTS operating system which includes the relevant tools. The test network does not have a DNS service configured. This means that you will have to identify all machines / interfaces by their IP addresses. Furthermore, the test network does not have access to the real Internet.

Before starting with the labs you will need to carry out some one-time configuration steps. You will first create the host-only networks and then import the VMs into your home directory.[2] Here is a step-by-step guide for setting up your work environment:

- Open a shell, start the VirtualBox software

  ```
  $ virtualbox
  ```

- Make sure there is no existing host-only network
  - Choose menu `File -> Host Operations Manager...`
  - Remove the existing host-only network if there is any
- Open another shell, create the host-only networks by running the following script

  ```
  $ /netfs/share/bin/create_vbox_vnets
  ```

- In the same shell, now import the VMs by running the following scripts one by one

  ```
  $ /netfs/share/bin/cosc364vm-alpha
  $ /netfs/share/bin/cosc364vm-auckland
  $ /netfs/share/bin/cosc364vm-beta
  $ /netfs/share/bin/cosc364vm-christchurch
  $ /netfs/share/bin/cosc364vm-delta
  $ /netfs/share/bin/cosc364vm-dunedin
  $ /netfs/share/bin/cosc364vm-epsilon
  $ /netfs/share/bin/cosc364vm-gamma
  $ /netfs/share/bin/cosc364vm-hamilton
  $ /netfs/share/bin/cosc364vm-oamaru
  ```

- Start all VMs.
- Log into all VMs as user `student` and type the valid password. After you have logged in, please enter the command

  ```
  /sbin/ifconfig
  ```

  which displays the list of the available interfaces. The interface names are printed on the left side. You will very likely see an interface named `lo`, which is the so-called **loopback interface**. You should also see at least one (and on the routers two or three) interfaces named `enpxsx` (like `enp0s3`, `enp0s8` etc), which refer to Ethernet interfaces. Here, `en` stands for Ethernet which followed by a `p` for PCI slot and `s` for hotplug PCI-E slot. This naming scheme incorporates physical/geographical location of the connector of the hardware which makes the interface names predictable.

Now you are good to go.

[2] More precisely: you will create what can be called a *linked clone*: there are some "mother copies" of the VMs stored on the hard disk of your lab machine, and by importing the VMs into your home directory you do not create a full copy but rather a "link" to the mother copy, and subsequently in your home directory only the parts of the VM are stored which differ from the mother copy.

5.2 Discovering Network Topology and Initial Setup

The goal of this set of exercises is to let you explore the topology of our test network and carry out some basic configuration of end hosts and routers. It is assumed that at the beginning of these exercises you are working on a fresh copy of the virtual machines and have not modified them before. If this is not true, then some of the exercises below might not work out as intended.

Problem 5.2.1 (Network topology)**.**

- Switch on all virtual machines (hosts and routers), wait until the last machine has booted before making the next step
- Use `ifconfig` in all machines to discover the topology of the network:
  - How do you use IP addresses and netmasks in this process?
  - Draw a diagram of the topology showing all subnetworks, all hosts and routers, which interface of each host is attached to which subnetwork, and the IP address of this interface.

  You might need to use the invocation `ifconfig | more` to read the output of `ifconfig` one screenful at a time.
- Use `ping` between each pair of hosts connected by the same network to check direct connectivity. It is likely that the round-trip time for the first packet is somewhat larger than for the remaining packets. Why is this?
- Log into host `alpha` and ping the IP address of host `epsilon`. What happens? And why?
- Shut down all virtual machines. To do this, use from within the machine the command `sudo shutdown -P now`.

Problem 5.2.2 (End hosts)**.**

- Re-boot all virtual machines so that they start "empty"
- First consider the five end hosts `alpha` to `epsilon`. On each one do the following:
  - Use the `route` command to display the forwarding table. Make sure you understand the different fields/columns in the output of the `route` command.
  - Add a default router. You can do this for the current session using the `route` command, to make this change permanent you need to edit (with root permissions, i.e. with `sudo`) the file `/etc/network/interfaces`. In both cases you need to figure out how to do this.

  For host `delta` please pick router `dunedin` as the default router.
- On host `alpha` send again a `ping` to host `epsilon`. You will be getting a different reaction to the one you got in Problem 5.2.1. What is happening here and why?
- On host `alpha` send a ping to IP address `144.10.64.39`. What happens, and why?
- Continue with the next problem without re-starting the hosts or routers.

Problem 5.2.3 (Initial setup of routers)**.**
After we have successfully configured the end hosts, we turn our attention to the routers.

- Log into router `christchurch` and display the forwarding table. What do you see?
- How to enable IP forwarding on a router? Make sure that IP forwarding is enabled on all routers and that this is a permanent change. What is the meaning of this step? Under Ubuntu 16.04 you will require invocations of the `sysctl` command, and you will need to edit the file `/etc/sysctl.conf`.:wq

5.3 RIP Routing

In this set of lab experiments you will configure the RIPv2 dynamic routing protocol and carry out a number of experiments.

Problem 5.3.1 (RIP Routing – Get it running)**.**
We now set up and analyze RIP routing. The configuration steps need to be carried out on the routers only.

- Change into the directory with all `quagga` configuration files: `cd /etc/quagga`. Edit the file `daemons` to enable the `zebra` daemon and the `ripd` (by setting the entries to "yes"[a]).
- Next create and edit the file `zebra.conf` on all routers (do not forget to give the right file permissions, see Appendix B). You need to find out how, see the `quagga` manual for the available options. It may be a good idea to let the `zebra` daemon create a logfile. If you decide to do so, it is suggested to place the log file into the directory `/var/log/` (you can create an empty file with the command `touch`, see its `man` page) and give it sufficient access rights by the command `chmod 666 /var/log/zebra.log`, assuming you have chosen to call your logfile `/var/log/zebra.log`.[b] After any change to one of the configuration files you need to restart the `quagga` daemons (by using the command `/etc/init.d/quagga restart`). In the following we will not remind you anymore about this.
- Now create and edit the RIP configuration file `ripd.conf` on all routers to enable RIPv2 routing (do not forget to give the right file permissions, see Appendix B). You need to find out how. I suggest to do this one at a time: start with `christchurch` then `hamilton`. Use `vtysh` commands (see below) in `christchurch` before and after activating `hamilton` to see how things change.
- Test your setup with `ping` and `traceroute` for full connectivity. In particular, use `traceroute` or `ping` on all the end hosts to check connectivity to all the other end hosts.
- Log into the routers and open the command line interface to the `zebra`/`ripd` daemons. One way to do this is to call (with `sudo`) the command `vtysh`. Some of the interesting commands available in this shell are the following:
  - `show ip route`: shows the contents of the forwarding table currently used by the kernel.
  - `show interface`: shows all the interfaces and their properties (basically the output of the `ifconfig` command).
  - `ping` and `traceroute`: these are essentially similar to the command line versions of these commands.

  These commands are universally applicable. Some of the RIP specific commands are:
  - `show ip rip`: show routes learned through RIP and shows RIP-specific information, e.g. metrics.
  - `show ip rip status`: displays a range of status information about RIP operation.

  To show information (e.g. to respond to the `show ip rip` command) `quagga` relies on an external program to actually do the display output. The default external program requires you to press `q` at the end.

[a] In the manual you will find that you actually can assign priorities here. `zebra` can run different routing daemons in parallel and can help with distribution of routes between these. The priority mechanism in the daeomons file simply tells in which order to start different daeomons (e.g. start bgp well after ospf so that bgp already can use full knowledge).

[b] Note: this is not how you would do it in real life, where you would limit read and write access to specific users and groups. But for our purposes this will do.

Problem 5.3.2 (RIP Routing – Try to break it)**.**
By now RIP routing should be running without problems. We will now introduce a number of failure situations and see how RIP reacts.

- Log into host `alpha` and start a `ping` command towards `epsilon` and let it run (make sure you can observe its output). Log into router `oamaru` and de-activate `oamaru`'s interface towards network `144.10.65.0/24` (this interface will have a name of the form `enp0sx` with `x` being a number; you can find out the right interface name from `ifconfig` and then you enter the command `sudo ifdown enp0sx`). After doing this, wait a few minutes and observe the output of the `ping` command running on `alpha`. What happens here? Explain.
- Still on `oamaru`, give the command `sudo ifup enp0sx` (with `x` replaced by the number you have used in the previous step) to re-activate the link and bring the network back to its original state.
- Next, on `alpha` start a `ping` command towards host `delta`. After that, log into router `dunedin` and de-activate interface `dunedin`'s interface towards `77.14.6.0/24`. What happens here? Explain. And suggest a protocol mechanism which can theoretically be used to solve this problem.
- After this experiment shut down and restart all routers.

5.4 Single-Area OSPF Routing

Problem 5.4.1 (OSPF Routing – Get it running)**.**
We next consider OSPF routing so that all routers / subnetworks are in the same area.

- Before you start, describe what you expect to be in the link-state database: how many router LSAs, how many network LSAs and how many summary LSAs.
- Re-start all the router VMs, stop `quagga` by giving (with `sudo`) the command `/etc/init.d/quagga stop`, and edit the `daemons` file to not use `ripd` anymore and using `ospfd` instead. Do not yet re-start `quagga`.
- Next create a config file for `ospfd`, which has to come by the name `ospfd.conf`. You need to figure out how to do this and what to put in (use the `quagga` manual). Requirements:
  - All routers / subnetworks belong to the same area `0.0.0.0`
  - The router id's must be unique.

  For reasons of readability I suggest to not use IP addresses as router id's but allocate them manually, for example: `christchurch=0.0.0.1`, `hamilton=0.0.0.2`, `auckland=0.0.0.3`, `dunedin=0.0.0.4`, `oamaru=0.0.0.5`. When you are not sure about an option then don't specify anything.
- Restart `quagga` on all routers. Check connectivity between all end hosts using `ping` and `traceroute` as usual.
- Wait for at least one minute after routers have been re-started and then find out which routers are the designated routers on the transit networks. Guess a rule why those routers became DR.
- You can inspect the status of `ospfd` and in particular the link-state database with the following commands in a `vtysh`:
  - `show ip ospf`: shows summary information about OSPF routing on a router, including how many interfaces it uses, how many router / network / summary . . . LSAs there are in the link-state database, timer settings and the like.
  - `show ip ospf database`: shows summary information about the router / network / etc LSAs in the database.
  - `show ip ospf database router`: shows all router LSAs in detail
  - `show ip ospf database router <r-id>`: shows the router LSA of router id `<r-id>`in detail
  - `show ip ospf database network`: dito for network LSAs
  - `show ip ospf neighbor`: shows information about neighboured OSPF routers
  - `show ip ospf route`: shows the OSPF routing table

  Inspect the LSAs and make sure you understand their contents.

Problem 5.4.2 (OSPF Routing – Test it)**.**
At the start of this exercise OSPF should be running without problems and the network should have full connectivity. Let us see how OSPF reacts to failures:

- Log into host `alpha` and start a `ping` towards `epsilon`, make sure you can observe its output. Shut down `oamaru`'s interface towards network `144.10.65.0/24`. What happens here? Explain.
- Stop `ping` on `alpha` and run the `traceroute` command `alpha`. How many hops does `alpha` need? Now switch `oamaru`'s interface towards network `144.10.65.0/24` back on, go back to `alpha` and do `traceroute` repeatedly until it changes. How long did this take?

5.5 Multi-Area OSPF Routing

Problem 5.5.1 (OSPF Routing – Several Areas)**.**
In this final exercise we want to sub-divide our OSPF routing domain into several areas.

- Stop `quagga` on all routers.
- Shut down `oamaru`'s interface towards network `144.10.65.0/24`.
- On all routers copy the existing OSPF configuration files to backup files before modifying them, e.g. using the command `cp ospfd.conf ospfd-singlearea.conf`
- Edit the OSPF configuration files `ospfd.conf` on all routers such that:
  - Subnetworks `144.10.64.0/24`, `144.10.65.0/24` and `144.10.66.0/24` belong to area `0.0.0.1`
  - Subnetworks `77.14.6.0/24` and `77.14.7.0/24` belong to area `0.0.0.2`
  - Subnetworks `10.2.0.0/24` and `68.22.1.0` belong to area `0.0.0.0`
  - `hamilton` and `dunedin` are area-border routers.
  - Do not yet apply any route aggregation.
  - On router `oamaru` remove network `144.10.65.0/24` from the list of OSPF networks
- Restart `quagga` on all routers, wait a bit and then check for complete connectivity.
- Now we inspect some link-state databases:
  - How many link-state databases will `hamilton` have now? And how many has `christchurch`?
  - Why are there now fewer router LSAs in `christchurch` than there were in the single-area case?
  - How many summary LSAs do you expect `christchurch` to have, which ones and why? Check.
- We finally want to arrange things such that routers `hamilton` and `dunedin` only advertise aggregated routes about areas `0.0.0.1` and `0.0.0.2` into the core area `0.0.0.0`. More specifically:
  - Note down the number of summary LSAs in routers `christchurch` and `oamaru`. Also note the length of the routing table on `auckland` (with the `route` command).
  - Configure router `hamilton` such that the three subnetworks `144.10.64.0/24`, `144.10.65.0/24` and `144.10.66.0/24` are advertised as a single network `144.10.64.0/20`.
  - Configure router `dunedin` such that the two subnetworks `77.14.6.0/24` and `77.14.7.0/24` are advertised as a single network `77.14.0.0/16`
  - Restart both routers after you have made these changes and wait a few seconds.
  - Now check whether the number of summary LSAs in routers `christchurch` and `oamaru` has changed and what precisely is summarized. Check the output of `show ip route`. Have a look at the length of the routing table of `auckland`, too.

Part III

Appendices

Appendix A

Shortest-Path Routing

In this appendix a very brief review of some key notions of graph theory and shortest-path algorithms is given. This is meant to serve as a reference, not as an introduction. If you lack background in graph theory then I recommend that you consult introductory textbooks, e.g. [1].

In the following when X is a finite set then $|X|$ denotes the number of elements in X.

A.1 Networks as Graphs

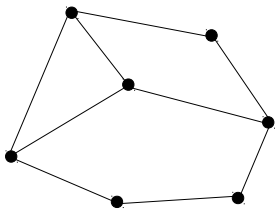


Figure A.1: An example of an undirected graph

We start by remembering a few definitions (see [17]):

- An **undirected graph** is a triple (V, E, Ψ) where V and E are finite sets (called the *vertex set* or *node set*, and the *edge set*, respectively) and the mapping $\Psi : E \mapsto \{X \subset V | |X| = 2\}$ indicates for a given edge e the two vertices it is connecting (these can be listed in any order). We typically write an edge e as $e = \{x, y\} = \{y, x\}$ with x and y being two vertices. Intuitively: when there is an edge e between two vertices x and y then one can reach node y from node x and vice versa. An example of an undirected graph is shown in Figure A.1.
- A **directed graph** (or digraph) is a triple (V, E, Ψ) as before, but $\Psi : E \mapsto \{(v, w) \subset V \times V | v \neq w\}$, i.e. an edge is an ordered pair $e = (x, y)$, indicating that it is possible to go from node x to node y.
- Note that these definitions allow to have two edges e and e' connecting the same vertices x and y, these are then called parallel edges. A graph without parallel edges is called **simple**. Simple graphs can be

written in a simplified way as (V, E) by identifying an edge directly (and uniquely!) with the pair $\{x, y\}$ or (x, y) involved.[1]

- If there exists an edge $e = \{x, y\}$ or $e = (x, y)$ between two nodes x and y then the two nodes are called **adjacent**.
- We write $\langle x, x \rangle$ when two nodes x and y are adjacent but we do not care whether the graph is directed or undirected.
- Be $G = (V, E)$ a simple graph and $x \in V$ a node. If the graph is undirected then the set $\mathcal{N}_x = \{y \neq x | \{x, y\} \in E\}$ of nodes adjacent to x is the **neighbourhood** of x. The quantity $|\mathcal{N}_x|$ is the **degree** of node x.
- Be $G = (V, E)$ a directed or undirected simple graph. A **path** of length n from a node x to a node y is given by a sequence of edges $e_0 = \langle v_0, v_1 \rangle$, $e_1 = \langle v_1, v_2 \rangle$, $e_2 = \langle v_2, v_3 \rangle$, ..., $e_{n-1} = \langle v_{n-1}, v_n \rangle$ such that all edges e_i are in the edge set E, the ending node v_i of edge e_{i-1} is identical to the starting node v_i of edge e_i, the first vertex v_0 coincides with x, the last vertex v_n coincides with y, and all "inner" vertices $v_1, \ldots, v_{n-1}$ are pairwise distinct (i.e. no inner vertex is visited twice). If we give up this last restriction then we speak of a **walk**.
- An directed or undirected simple graph $G = (V, E)$ is called **connected** (or **strongly connected** for a directed graph) if for each pair of nodes x and y there exists a path between x and y.

Often communication networks are modeled as directed or undirected graphs. We will only consider simple graphs as there is rarely more than one direct physical link between two adjacent stations, and if there is, there is rarely any need to actually distinguish them and **not** treat them as one aggregated link. Furthermore, in communication networks with a wired topology it is usually true that if two stations are adjacent to each other (i.e. connected to each other on the physical layer with a direct link) then this link operates in both ways and offers the same data rate in both directions, hence such networks can often modeled as undirected graphs.

A.2 Shortest-Path Problems

Broadly speaking, routing is concerned with finding *good* paths between each pair of nodes in a network modeled as a (directed or undirected) simple graph $G = (V, E)$. To quantify this, we need to assign to a path P between two nodes x and y a quantity measuring its "goodness". In the Internet it has become customary to express the "goodness" of a path through a scalar cost value, and given two different paths P_1 and P_2 between nodes x and y with cost values $c(P_1)$ and $c(P_2)$, we prefer the path with the numerically smaller cost. The precise nature of the cost (whether it expresses delay, monetary cost, ...) depends on the circumstances but does not really matter for path-finding algorithms.

A second decision has been made for path-finding in the Internet (at least for the routing protocols we will be discussing). Namely, in the graph $G = (V, E)$ a scalar cost value is assigned **to each edge** $e \in E$ individually, and the cost of an edge e is denoted as $c(e)$. When we specify a path P through the sequence of edges it uses, i.e. $P = (e_0, e_1, \ldots, e_{n-1})$ then the cost of the path $c(P)$ is defined as

$$c(P) = \sum_{i=0}^{n-1} c(e_i)$$

In this context we can pose three different problems.

[1] In other words, the mapping $\Psi(\cdot)$ really is only necessary to distinguish several edges between the same pair of nodes.

Definition A.1 (Single-Pair Shortest Path Problem - SPP)**.** *We are given a (directed or undirected) simple graph $G = (V, E)$ with edge costs $c : E \mapsto \mathbb{R}$ and two vertices $x \in V$ and $y \in V$. Then we want to find a path between x and y that has the shortest length among all paths from x to y (note that there can be several paths of the same shortest length) and the cost value of this path, or conclude that no such path exists. If such a path exists we call it a* ***shortest path****.*

A path can fail to exist in networks that are not (strongly) connected. Note also that we insist on finding a path and not a walk.[2]

Definition A.2 (Single-Source Shortest Path Problem - SSP)**.** *We are given a (directed or undirected) simple graph $G = (V, E)$ with edge costs $c : E \mapsto \mathbb{R}$ and a vertex $x \in V$. Then we want to find the shortest paths between x and all other vertices $y \in V$ and their costs, or conclude that no such path exists.*

Definition A.3 (All-Pairs Shortest Path Problem - APSP)**.** *We are given a (directed or undirected) simple graph $G = (V, E)$ with edge costs $c : E \mapsto \mathbb{R}$. Then we want to find the shortest paths between all pairs of distinct vertices $x \in V$ and $y \in V$ and their costs, or conclude that no such path exists.*

Routing protocols tend to be concerned with this last type of problem.

If for a given path P we denote by $v(P)$ the set of all vertices included in the path (including start- and end-vertex) then we can make a simple observation:

Remark A.1. *Let P be a shortest path from node x to node y and let w some node on the path, $w \neq x$ and $w \neq y$. Then the sub-path $P_{x,w}$ from x to w of path P is a shortest path from x to w. Furthermore is the sub-path $P_{w,y}$ of P a shortest path from w to y.*

Proof. If there were a shorter path $P'_{x,w}$ between x and w, then we could construct a new path P' from x to y by concatenating $P'_{x,w}$ and $P_{w,y}$, which would have a smaller total cost than path P, but then P can't have been a shortest path to start with, which is a contradiction. □

A.3 Shortest-Path Algorithms

We will briefly discuss two shortest-path algorithms, Dijkstra and Bellman-Ford.

We are given a network $G = (V, E)$ with $N = |V|$ stations, the network is undirected and simple. In the following we assume that nodes are numbered from 1 to N, i.e. $V = \{1, \ldots, N\}$. When i and j are two generic nodes from the network then we define the **direct distance** $d_{i,j}$ between i and j to be either:

- $d_{i,j} = c(\{i, j\})$, i.e. it equals the cost of the edge $\{i, j\}$ when i and j are adjacent. In the following we assume that all the edge costs $c(\cdot)$ are non-negative and finite.
- $d_{i,j} = \infty$ when i and j are non-adjacent nodes.

In the Bellman-Ford algorithm, $\widehat{D_{i,j}}$ represents the total cost of the minimum cost path from i to j, over one or multiple hops, according to i's current knowledge. The quantity $\underline{D}_{i,j}$ represents the same thing for Dijkstra's algorithm.

We illustrate this notation with reference to Figure A.2.

[2] If we were to admit walks then we must be careful if we have negative edge costs. In particular, if there exists a cyclic walk for which the sum of the edge costs is negative, then we can construct walks between x and y of arbitrarily low costs simply by including the cycle into the walk a sufficient number of times. We do not deal with pathological cases of this kind.

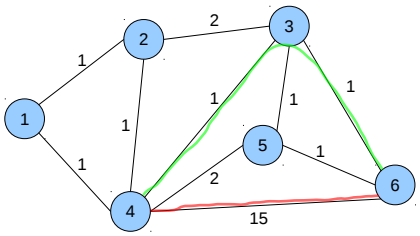


Figure A.2: An example to illustrate notations

- In this example we have $d_{4,6} = 15$, but $\widehat{D_{4,6}} = \underline{D}_{4,6} = 2$ (by choosing the path $4 - 3 - 6$)
- Furthermore, $d_{1,6} = \infty$, but $\widehat{D_{1,6}} = 3$ (by choosing the path $1 - 4 - 3 - 6$)
- We have $\mathcal{N}_5 = \{3, 4, 6\}$.

A.3.1 Bellman-Ford Algorithm

The Bellman-Ford algorithm solves the SSP problem and builds on the dynamic programming principle [2], [3]. Assume that we want to find a shortest path from source node $s \in V$ to destination node $d \in V$. For such a shortest path the following equations must be satisfied (compare Remark A.1):

$$\begin{aligned} \widehat{D_{s,s}} &= 0 \\ \widehat{D_{s,d}} &= \min_{k \in \mathcal{N}_d} \left\{ \widehat{D_{s,k}} + d_{k,d} \right\}, \qquad \text{for } s \neq d \end{aligned}$$

These equations can be understood as follows:

- The first equation says that the shortest path from the source node s to itself will always have zero cost (which is a trival observation).
- For the second equation suppose node s already knows its least costs $\widehat{D_{s,k}}$ to the neighbors $k \in \mathcal{N}_d$ of the destination d, then s's least cost to d is the minimum over all neighbors k of the costs $\widehat{D_{s,k}}$ plus the direct costs $d_{k,d}$.

In a very similar way we can establish the following equations in which we minimize over the neighbours of the source node s:

$$\begin{aligned} \widehat{D_{s,s}} &= 0 \\ \widehat{D_{s,d}} &= \min_{k \in \mathcal{N}_s} \left\{ d_{s,k} + \widehat{D_{k,d}} \right\}, \qquad \text{for } s \neq d \end{aligned}$$

In the Bellman-Ford algorithm the costs between a source node s and all destination nodes d are computed iteratively over all possible numbers of hops (of which there can be at most $N-1$), one version of the algorithm suitable for non-negative edge costs is given in Figure A.3. It maintains two different vectors: the vector of best-costs $\widehat{D_{s,d}}$ to all the destinations d and a vector `pred` of node identifiers. The algorithm initializes the best cost to itself as $\widehat{D_{s,s}} = 0$, the best cost to all other destinations d as $\widehat{D_{s,d}} = \infty$ and the `pred` array to all destinations with a `NULL` value. It then loops over all possible numbers of hops, and for each possible hop number loops over all edges $\{v, w\}$, and checks whether the current best cost from s to w (given by $\widehat{D_{s,w}}$)

```
// Computes for a fixed node s the distances and the routing
// tree to all other nodes.

// initialization
D̂_{s,s} = 0; pred[s] = s;
forall d with d ≠ s do
    D̂_{s,d} = ∞; pred[d] = NULL;

/* loop over all numbers of hops */
for h = 1 to N − 1 do:
    foreach {v, w} ∈ E do
        when D̂_{s,v} + d_{v,w} < D̂_{s,w} then
            D̂_{s,w} = D̂_{s,v} + d_{v,w};
            pred[w] = v
```

Figure A.3: One version of the Bellman-Ford algorithm

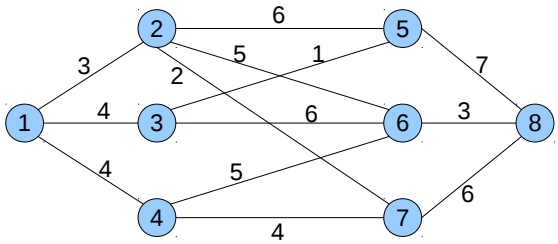


Figure A.4: An example network

can be improved by going from s through v (at cost $\widehat{D_{s,v}}$) and then from v to w. If so, the best cost $\widehat{D_{s,w}}$ are updated and the `pred` array is modified accordingly.

The runtime of this algorithm for a single source is $\mathcal{O}(|V| \cdot |E|)$, for a whole network it becomes $\mathcal{O}(|V|^2 \cdot |E|)$. It can handle only non-negative weights as shown here, but it is possible to extend the algorithm to cover negative weights as well (as long as no negative cycles are contained) Note that it is a centralized algorithm, complete network information ($d_{i,j}$) must be available at execution time. The proof that this algorithm indeed gives the shortest paths to all destinations is beyond the scope of this text but can be found for example in [17].

Problem A.3.1 (The `pred` array)**.** What exactly is the `pred` array in Figure A.3 good for? How can you use it after the algorithm terminates?

```
// Computes for a fixed node s the distances and the routing
// tree to all other nodes. Graph is assumed to be directed

// initialization
S' = V \ {s}
D_{s,s} = 0; pred[s] = s
forall d ∈ S' do
    D_{s,d} = d_{s,d}; pred[d] = NULL;
    when d_{s,d} < ∞
        pred[d] = s

// main loop
while S' ≠ ∅ do
    k = arg min_{m∈S'} D_{s,m}
    S' = S' \ {k}
    for j ∈ N_k do
        when D_{s,k} + d_{k,j} < D_{s,j}
            D_{s,j} = D_{s,k} + d_{k,j}
            pred[j] = k
```

Figure A.5: One version of the Dijkstra algorithm

Problem A.3.2 (A Bellman-Ford example)**.**
Consider the example network shown in Figure A.4.
Run the Bellman-Ford algorithm to find the minimum-cost routes from station 1 to all other stations. For each step (the initialization step and each iteration of the outer loop over h give:

- all values $\widehat{D_{1,d}}$
- the contents of `pred`

After termination of the algorithm, give the shortest-cost route for each destination (use the array `pred` to read the route off).

A.3.2 The Dijkstra Algorithm

Dijkstra's algorithm is restricted to graphs with non-negative weights It is **greedy**: in every situation it makes the choice that is currently the best, without regard to future situations. Here:

- The algorithm maintains a list $\mathcal{S}'$ of nodes that have not yet been considered.
- In each step it removes the node $k \in \mathcal{S}'$ to which the source s has the smallest known distance $\underline{D}_{s,k}$.
- For each neighbor x of k it is then checked if a path through k to x is shorter than the best so-far known path to x.

The actual algorithm is sketched in Figure A.5. This algorithm is sketchy and cannot handle non-negative link metrics. A real implementation would have to flesh out a number of details, for example the precise

implementation of the statement $k = \arg\min_{m \in \mathcal{S}'} \underline{D}_{s,m}$ – the data structures used here can have substantial impact on algorithm performance. The worst-case runtime of Dijkstra's algorithm for a single node is $\mathcal{O}(N^2)$, but can be better for sparse graphs.

Problem A.3.3 (A Dijkstra example)**.**
Consider again the network shown in Figure A.4 and run the Dijkstra algorithm to find the best paths from node 1 to all other nodes. For each step (including the initialization) give:

- the set $\mathcal{S}'$
- all values $\underline{D}_{1,d}$ (we will simplify notation and simply write this as $D_{1,d}$) and the contents of the array `pred`
- the selected node $k \in \mathcal{S}'$.

Appendix B

Some Lab Material from COSC 264

In this appendix some of the lab materials from COSC 264 are repeated in a slightly revised fashion.

B.1 Elementary Unix/Linux networking tools

You will need to familiarize yourself with a number of basic tools for diagnosing, printing and configuring network-related information under Unix/Linux. There are four popular tools that every network engineer should be familiar with: `ping`, `ifconfig`, `route` and `traceroute`.

The `ifconfig` tool can be used to configure network interfaces on a host and to print information related to these interfaces. Here we restrict to looking at information printed by this tool. Depending on the Linux distribution you use, the shell might print an error message after you have entered the `ifconfig` command. This is because the `ifconfig` executable is not stored in one of the directories which are included in a users search path.[1] In this case you will instead have to enter the command

```
/sbin/ifconfig
```

to see the existing interfaces and their main properties. In contrast, the `ping` and `traceroute` tools should be accessible without giving the `/sbin/` prefix.

Problem B.1.1 (Understanding `ifconfig` output)**.**
Make sure you understand the output of the `ifconfig` command for an Ethernet interface.

The `traceroute` tool displays a list of intermediate routers between your host and the final destination. Nowadays `traceroute` often fails to do this properly since many institutions run firewalls blocking the packets important for `traceroute` (the same is also true for `ping`). However, `traceroute` will be available in your test network.

The `route` tool displays the current contents of the forwarding table and also allows to add or delete routes from the command line.

[1] The search path is a list of directories in which the shell looks for executables when you submit a command. You can see the list of directories in which the shell searches for an executable by giving the command `printenv PATH`. Actually, `PATH` is an environment variable and can be changed. Consult a tutorial for the `bash` shell and look for the commands `setenv` and `export`.

Problem B.1.2 (`man` pages for networking tools)**.**

- Read the man page for `ping`.
- Read the man page for `traceroute`.
- Read the man page for `arp`.
- Read the man page for `route`.

Problem B.1.3 (Networking tools)**.**

- How is `ping` implemented, i.e. which IP facilities or protocol features does `ping` use? How do the related IP datagrams look like exactly?
- How is `traceroute` implemented? Describe the key approach of its implementation.
- What does a `traceroute` user assume about the displayed route when using this tool? Is this assumption always true? Please explain.

B.2 The `quagga` routing software

The `quagga` routing package (see `www.quagga.net`) provides a basic framework for IP routing plus several routing daemons for individual dynamic routing protocols, e.g. RIPv2 or OSPF. However, `quagga` also allows you to do static routing. The `quagga` routing package is pre-installed on all the router virtual machines (i.e. `christchurch`, `auckland`, etc.) but is not available on the end hosts (`alpha`, `beta`, etc.).

Generally speaking, when an IP router has decided to forward an IP datagram, it consults its **forwarding table**. The forwarding table contains several entries, one entry per IP prefix / IP subnetwork. An IP prefix here refers to a combination of an IP network address and netmask. For each IP prefix the forwarding table stores the outgoing interface (on a linux-based router e.g. `eth0`) and the IP address of the next-hop router, which must be reachable through a directly attached subnetwork (i.e. on the outgoing interface). The forwarding table is all that the router (or the Linux kernel) needs for forwarding IP datagrams. There are fundamentally two different ways of **populating** the forwarding table: static routing and dynamic routing.

Generally speaking, with static routing the IP forwarding table is configured manually (by editing configuration files). You ought to have done this in COSC 264 and probably found this to be a tedious and error-prone process. In contrast, with dynamic routing, on all routers a dedicated piece of software is running, a so-called **routing daemon**. Such a routing daemon implements a dynamic routing protocol (e.g. RIP, OSPF) which communicates with neighbored routers, exchanges reachability information, and, most importantly, automatically updates the forwarding table used for IP forwarding.

The `quagga` routing package contains first the `zebra` routing daemon, which is the sole manager of the Linux kernels forwarding table. On top of that, `quagga` can run one or more of the routing daemons that are part of the package, for example the RIP daemon, the OSPF daemon and so forth. These routing daemons modify the forwarding table through the services offered by the `zebra` daemon. In addition, the `quagga` routing package comes along with the `vtysh` command, which allows a user to "log in" to the routing package and to issue commands to the `zebra` daemon and any routing daemon in a command language that resembles the

language of CISCO routers running the CISCO IOS ("Internet Operating System"). This can be used to change the configuration of the routing daemon or to inspect internal data (e.g. the OSPF link-state database).

We will not give a comprehensive tutorial about the `quagga` software. You find a `quagga` manual under `www.quagga.net`, following the documentation link (the manual has also been placed on the learn platform). Furthermore, Google shows up `quagga` tutorials. Nonetheless, in the following you can find some important bits and pieces about the work with the `quagga` software suite:

- You can find all `quagga` configuration files in the directory

  ```
  /etc/quagga
  ```

  To get into this directory you can use the `cd` (change directory) command, thus entering

  ```
  cd /etc/quagga
  ```

  in the command line. The configuration files are all stored in ASCII format, so you can edit them with `vi` or `emacs`. When logged in as user `student` you can only read the configuration files but not modify them. If you want to modify a configuration file (e.g. the file `daemons`) you can use the command `sudo vi daemons`.

- Each routing daemon (including `zebra`) insists on finding its own configuration file in the directory `/etc/quagga`. For example, the `zebra` daemon expects a `zebra.conf` file to be present (even if it is empty), similarly the ripd (RIP daemon) expects file `ripd.conf`. To create an empty `zebra.conf` file you can give the command

  ```
  touch zebra.conf
  ```

  These files must have owner and group `quagga` and permissions `644`, which you can achieve with the commands

  ```
  sudo chown quagga:quagga filename
  sudo chmod 644 filename
  ```

  where of course `filename` is the name of the file you want to work with.

- Whenever you have modified any `quagga` configuration file you have to restart the overall `quagga` service (which is: all currently active `quagga` daemons). To do this, enter the command

  ```
  sudo /etc/init.d/quagga restart
  ```

  This command stops all currently running daemons and starts them afresh. Each daemon reads its configuration file after it started. When a daemon notices a syntax error in its configuration file, the startup process is stopped.

The `quagga` software suite offers an own command-line interface to the running daemons, the `vtysh` program.

To simplify your work, you can enter the command `sudo -s -H`. This gives you a **root shell** (which you can recognize by the changed prompt – `#` instead of `$`).

BEWARE: This is not for the faint-hearted, you have now the absolute power over the system, there is nothing that protects you against destroying or misconfiguring it. Be careful, especially when you find yourself typing `rm` on the command line. `rm` deletes files and it really does that, there is no trash bin from which you can recover deleted files. If you want to play it safe, don't perform this step but then prefix all the following commands with `sudo`.

Problem B.2.1 (Review problem)**.**

- Review the difference between routing and forwarding in the context of IP protocols.
- Based on this: what is the difference between a routing table and a forwarding table?
  - A note of caution here: these two notions are often exchanged or confused in the available literature / resources. For example, what has been called a forwarding table in the lecture is called a routing table in the `route` command, see Problem 5.2.2. It is important that you first understand the conceptual differences independently of any technology and then adapt to whatever wording the underlying software / technology uses.

Problem B.2.2 (Review problem)**.**

- Please familiarize yourself with the `quagga` users manual. You do not have to read it cover to cover, but you should know it well enough to find relevant pages quickly.

Bibliography

[1] Geir Agnarsson and Raymond Greenlaw. *Graph Theory: Modeling, Applications, and Algorithms*. Prentice Hall, Upper Saddle River, NJ, 2006.

[2] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control – Volume 1*. Athena Scientific, Belmont, Massachusetts, 3rd edition, 2005.

[3] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control – Volume 2*. Athena Scientific, Belmont, Massachusetts, 3rd edition, 2007.

[4] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus – A Theory of Deterministic Queueing Systems for the Internet*. Springer, Berlin, Heidelberg, 2001.

[5] K. Butler, T. R. Farley, P. McDaniel, and J. Rexford. A survey of bgp security issues and solutions. *Proceedings of the IEEE*, 98(1):100–122, January 2010.

[6] Brian E. Carpenter and Kathleen Nichols. Differentiated services in the internet. *Proceedings of the IEEE*, 90(9):1479–1494, September 2002.

[7] R. Coltun, D. Ferguson, and J. Moy. Ospf for ipv6. RFC 2740, December 1999.

[8] Constantinos Dovrolis, Dimitrios Stiliadis, and Parameswaran Ramanathan. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. *IEEE/ACM Trans. Networking*, 10(1):12–26, February 2002.

[9] Jeff Doyle. *OSPF and IS-IS*. CISCO Press, Addison Wesley, 2006.

[10] Jeff Doyle and Jennifer Carroll. *Routing TCP/IP*, volume Vol. 1. CISCO Press, Indianapolis, USA, second edition, 2006.

[11] Deepak Ganesan, Bhaskar Krishnamachari, Alec Woo, David Culler, Deborah Estrin, and Stephen Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. Technical Report UCLA/CSD-TR 02-0013, Computer Science Dept., University of California, Los Angeles (UCLA), 2002.

[12] J. J. Garcia-Luna-Aceves. Loop-free routing using diffusing computation. *IEEE/ACM Trans. Networking*, 1(1):130–141, February 1993.

[13] Sam Halabi. *Internet Routing Architectures – the definitive BGP resource*. CISCO Press, Indianapolis, USA, second edition, 2001.

[14] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930, March 1996.

[15] John W. Stewart III. *BGP4 – Inter-Domain Routing in the Internet*. Addison-Wesley, Boston, MA, 1999.

[16] ISO/IEC. *ISO/IEC 10589 – Information Technology – Telecommunications and information exchange between systems – Intermediate System to Intermediate System intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473)*, 2002.

[17] Bernhard Korte and Jens Vygen. *Combinatorial Optimization – Theory and Algorithms*. Springer, Berlin, third edition, 2005.

[18] Vijay P. Kumar, T. V. Lakshman, and Dimitrios Stiliadis. Beyond Best Effort: Router Architectures for the Differentiated Services of Tomorrow's Internet. *IEEE Communications Magazine*, 36(5):152–164, May 1998.

[19] James F. Kurose and Keith W. Ross. *Computer Networking – A Top-Down Approach Featuring the Internet*. Addison-Wesley, Boston, fourth edition, 2001.

[20] G. Malkin. RIP Version 2. *RFC 2453*, 1998.

[21] Deepankar Medhi and Karthikeyan Ramasamy. *Network Routing – Algorithms, Protocols, and Architectures*. Morgan Kaufmann, San Francicso, California, 2007.

[22] Daniel Minoli. *IP Multicast with Applications to IPTV and Mobile DVB-H*. John Wiley and Sons, Chichester, UK, 2008.

[23] J. Moy. Ospf version 2. RFC 2328, April 1998.

[24] John T. Moy. *OSPF – Anatomy of an Internet Routing Protocol*. Addison Wesley, Reading, Massachusetts, 1998.

[25] John T. Moy. *OSPF – Complete Implementation*. Addison Wesley, Reading, Massachusetts, 2001.

[26] Maria Rita Palattella, Nicola Accettura, Xavier Vilajosana, Thomas Watteyne, Luigi Alfredo Grieco, Gennaro Boggia, and Mischa Dohler. Standardized Protocol Stack for the Internet of (Important) Things. *IEEE Communications Surveys and Tutorials*, 15(3):1389–1406, 2013.

[27] Y. Rekhter, T. Li, and S. Hares. A border gateway protocol 4 (bgp-4). RFC 4271, January 2006.

[28] Zheng Wang. *Internet QoS – Architectures and Mechanisms for Quality of Service*. Morgan Kaufmann Publishers, San Francisco, 2001.

[29] Paul P. White and Jon Crowcroft. The integrated services in the internet: State of the art. *Proceedings of the IEEE*, 85(12):1934–1946, December 1997.