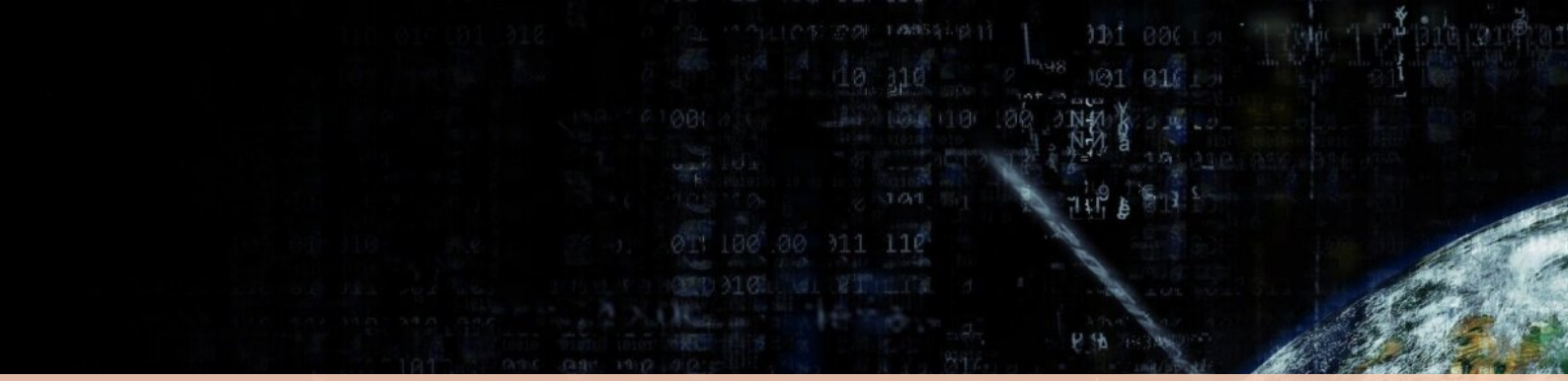


Computer Networks and Internet

104353 — Data Engineering — 1st year

Universitat Autònoma de Barcelona

Study Guide 2025

By Miguel Hernández-Cabronero
<miguel.hernandez@uab.cat>

UAB DEIC

License

The latest version of this document and the materials can be obtained from

add URL

.

Credits

Many of the visual contents were created by third-party artists and released under compatible licenses:

- Mathias Legrand and Vel: base LaTeX template
- TheDigitalArtist/pixabay: Front cover and Chapter 6 header.
- victorsteep/pixabay: Chapter 1 header.
- OpenClipart-Vectors/pixabay: Chapter 2 header.
- D-Kuru, Niridya, Joe Ravi, Shaddack / Wikimedia Commons (WC) : Ex. 2.6
- Timwether, Timewalk / WC : Ex. 2.7
- deepai : Iceberg Section 2.2
- Chetvorno : Antenna diagram Section 2.2
- Berserkerus : Modulation diagram Section 2.2
- Graham Rhind : Chapter 3 cover
- mrcolo : Chapter 4 header.
- Geek2003 Switch in Section 3.1.
- ignartonosbg : Chapter 5 header.
- Toffelginkgo : Chapter 7 header.
- Doodles43 : Chapter 8 header.

Contents

1. Intro: Computer Networks and Internet (XOI) 2025

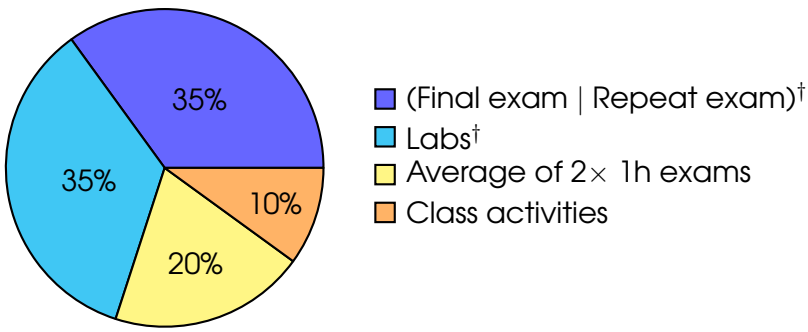
What you paid for

	Monday	Tuesday	Wednesday	Thursday	Friday
10:00					
10:30				**Problems** PAUL/**811**	**Labs** PLAB/**813**
11:00					
11:30				**Problems** PAUL/**812**	**Labs** PLAB/**814**
12:00					
12:30		**Lecture** Full class			
13:00					
13:30					
14:00					
14:30					
15:00					

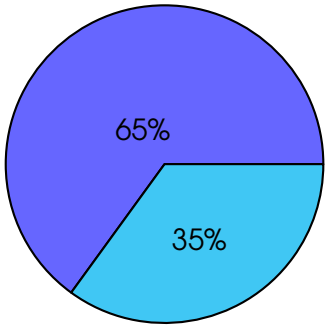
13× 2h **Lectures**
12× 1h **Problem sessions**
12× 1h **Lab sessions**
Unlimited **Office hour sessions** (*tutories*), individual or group: send email
90h **Autonomous work** (total, expected)

Evaluation

Option A
(regular)



Option B
(failed option A)



†: A passing grade (≥ 50%) is required in the part to pass the subject.

Dates:

- 1st 1h exam: during a Lecture session *early April*
- 2nd 1h exam: during a Lecture session *early May*
- Final 3h exam: Thursday June 5, 2025, 9h
- Repeat exam: Thursday June 26, 2025, 9h

This Guide

This guide is designed to help you progress through all parts of the "Computer Networks and Internet" course. Here you can find written materials, exercises and other tools to make the most of your effort and help you acquire valuable skills. It is strongly recommended that you become familiar with this guide and use it all throughout the semester, starting the very first week of class:

- During the lectures, we will use it to present and refer to new and previously visited concepts. We will also use it to drive activities and problem sessions.
- At home, it can help you gather and organize new knowledge, as well as to find useful exercises to practice your newly acquired skills.
- After you complete the course, it can be useful as an index of contents and pointers to useful reference materials.

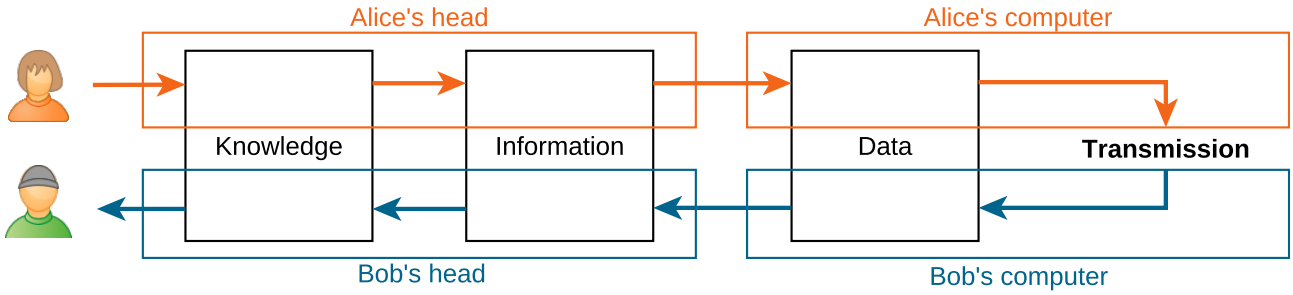
This guide is not a book, a reference material or even a complete set of class notes, and **it is not intended to substitute your own notes**. The guide's goal is rather about presenting you with new, interesting questions than about providing complete answers. This guide does not substitute continuous, active class attendance and autonomous work at home. Instead, you are encouraged to use the guide to help you understand the lectures, and vice-versa.

Computer Networks

How do computers exchange data? How can we locate and communicate with other machines located around the world? This part of the guide explores some of the main challenges we need to face if we want to interconnect devices at the planetary scale, and beyond.

2. Piercing the analog-digital veil

We send and receive messages constantly in our personal, professional and political spheres. The almost-instantaneous availability of virtually any desired information is a key characteristic of the current era. When we use a computer of any kind to share or retrieve knowledge, first it must be transformed into data that can be understood both by humans and computers. This can be imagined as a sequence of steps:



Computers work with and transmit digital data, *i.e.*, sequences of binary 0 and 1 symbols. However, data transmission happens in the real world, where 0 and 1 are ideas and not tangible objects.

This chapter studies how those digital symbols *pierce the veil* from the digital to the analog (physical) world on transmission, and back from analog to digital on reception.

2.1 Information ↔ Data ↔ Binary messages

Knowledge, information, and data are not the same thing, although they are often used used interchangeably. In technical contexts, they should be used and interpreted accurately.

> **Exercise 2.1** Suppose we are outdoors and we want to tell our friend about the weather. We have knowledge about everything related to the weather, because we are there. For our communication, we need to focus on part of that knowledge, and decide what information to send. For instance, we could want to communicate the temperature, and say something like "the temperature is about 22ºC". There are many ways in which we can store that information as data inside a computer. Most likely, we will represent the number 22 as part of those data. How would you define knowledge, information and data?

Even though there are many types of data, *i.e.* numerical, textual, visual, scientific, *etc.*, these are eventually represented by numbers. For instance, one could use ASCII encoding to represent the string "Bobby" as the following sequence: $66, 111, 98, 98, 121$.

Inside a computer, those numbers are stored in binary registers, *e.g.* in the CPU and in the RAM. However, there are multiple ways of representing numerical data in binary format. More specifically, we need to pay close attention to at least the following aspects whenever reading or writing data:

- integer or with decimals?
- bitdepth? (*e.g.*, 8 bits per number)
- signed or unsigned?
- For multibyte numbers: big endian or little endian?
- For numbers with decimals: floating-point or fixed-point?

Exercise 2.2 Explain **how to complete the second row given the first** (and vice-versa), and what assumptions you need to make in each case.

Number	7	2	−3	0.75	260
Binary	00111	00000010	1101	1100	0000 0100 0000 0001

How about à ↔ 1100 0011 1010 0000?

Sometimes we need to inspect the contents of a binary register (*e.g.*, for debugging or other analysis purposes) or to set those contents manually. For humans, it is inconvenient and *very* prone to error to handle binary strings longer than a few bits. When we need to inspect or modify the contents of a binary register (*e.g.*, for debugging purposes), we often use base 16, *i.e.*, hexadecimal notation.

In hexadecimal, there are exactly $16 = 2^4$ different digits (0 to 9, a to f) with decimal values from 0 to 15, each of which represents exactly 4 bits. When we writing hexadecimal values to a computer, *i.e.*, in source code, hexadecimal values are typically preceded by 0x, *e.g.*, 0x58a5b0. Similarly, binary expressions are preceded by 0b, *e.g.*, 0b0011.

- Spaces and even line breaks between digits do not carry any meaning, they are only used to facilitate visual inspection.
- In some texts, when multiple bases are applicable in a context, they are shown as subscripts as in $58a5b0_{16}$ and 0011_2.

Exercise 2.3 Consider the following message, which is composed of a concatenation of 3-bit unsigned integers: a4 3f 20.

- How many bits and bytes long is it?
- **What are the first five integers** contained in the message?

Most often, we will let our code handle data manipulation. To do that, we need bitwise and boolean logic operations such as the following

Operation	bitwise AND	logic AND	bitwise OR	logic OR	left shift	right shift
Python	`&`	`and`	`|`	`or`	`<<`	`>>`

Other useful python tools are the `bin()` and `hex()` functions, that convert an integer into its binary and hexadecimal representation, respectively, and `int()`, which can parse a string describing a number and return that number. We can also control the format in which we show numbers, *e.g.*, `print(f'{n:08b}')` would print the value of variable `n` in binary form, using at least 8 positions and filling the empty leading positions with zeros (more in the python docs).

Exercise 2.4 Consider the code shown next. The numbers printed when run are of special importance in networking and computer science. What do they have in common? (Hint: You may want to modify the code to show their binary representations).

.py .out snippets/bitwisemanipulation.py

```
1 a = 0xff
2 print(a)
3 for _ in range(8):
4     a = (a << 1) & 0xff
5     print(a)
```

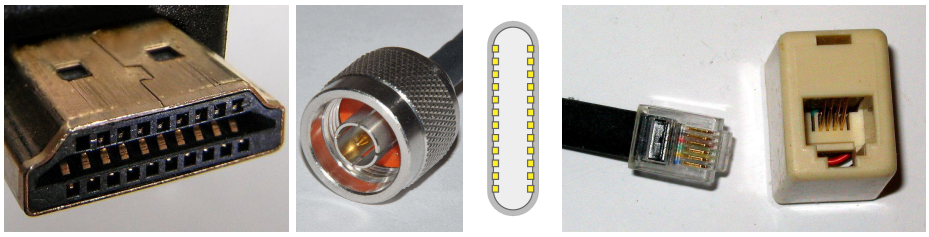
For code listings like the one above, you can download the source code and its output.

2.2 Analog ↔ Digital

Once we decide what bits to transmit, we need to find a way of making those bits available to another machine at a distance. First, we need to decide what medium to use and what physical properties we can control and monitor. Popular choices include using voltage on copper wires, sending light over optical fiber and manipulating the electromagnetic field using antennas.

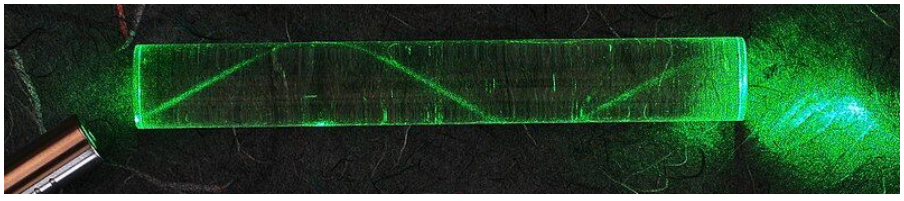
> **Exercise 2.5** **Ponder**:
>
> - Can we make a copper cable as long as we desire?
> - What medium do wifi connexions use?
> - Do we need optical fibers to perform light-based communication?

If we opt for copper wires, we can put several in parallel and send more data at the same clock speed. This, however, comes at the price of additional complexity and cost than serial designs. Voltage interferences in the data lines may happen for a number of reasons, including physical phenomena such as electromagnetic induction in the wire. Notwithstanding, cables compliant with modern data transmission standards (*e.g.*, IEEE 802.3) often employ twisted pairs of cables and shielding among other strategies to guarantee bit errors below 1 in 10^{12} bits.

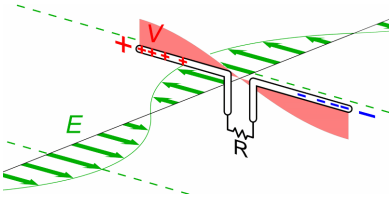
> **Exercise 2.6** **Consider and identify** the following wire connectors:
>
> 

Have you ever been diving in a swimming pool, looked up and the water was mirror-like? Optical fiber cables work of this phenomenon, called refraction. Light that enters the fiber through one end stays inside until it exits through the other end. The coating of the fiber is important to support its integrity, but that coating does not participate in the "bouncing" of light when advancing through the fiber.

In a single-mode optical fiber, the sensor at the receiving end detects the presence or absence of a single wavelength range. These cables are relatively thin and cheap, but carry a single signal. Another option is to transmit multiple wavelengths (*e.g.*, using several laser sources) at the same time, through the same fiber. At the receiving end, a prism is used to divide the beam of light back into its monochrome components, which are sensed separately. In this way, multi-modal optical fiber optical fiber allows multiplexing several signals, greatly increasing the effective transmission rate.

> **Exercise 2.7** When a beam of light enters an optical fiber, it does so with a certain angle of attack. This angle affects the time it takes to reach the other end. If we project a cone of light, light enters the fiber at multiple angles of attack. What happens at the other end if we **turn this cone of light on and off** very quickly? Is it relevant whether we use mono-modal or multi-modal beams?
>
> 
>
> 

Even though there are many classes of antennas, their main purpose is to detect electromagnetic fields and/or to create them. The Maxwell-Faraday equation tells us that *changes* in the electric field $\vec{E}$ create a perpendicular magnetic field $\vec{B}$, and the other way around.

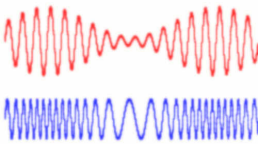


A dipole antenna like the one to the right uses this concept by applying an alternating voltage V, which generates electromagnetic waves that propagate outwards the dipole axis.

When we emit some energy E from a point in all directions, that energy is distributed around a growing sphere. Since the area of a sphere of radius r is $4\pi r^2$, the amount of energy that reaches a point at distance r will be in the ballpark of E/r^2.

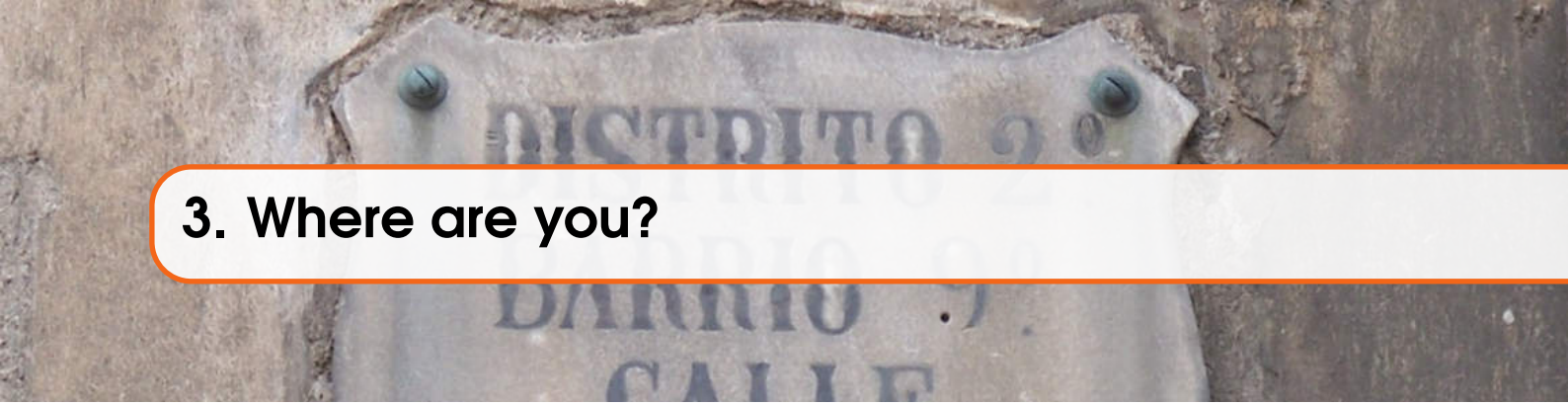
This is sometimes referred to as the inverse square law, and is the reason why currents inducted in the receiver's antenna are usually between nanovols (10^{-9} V) and picovolts (10^{-12} V). Surprisingly, this is enough for modern detectors to read the data signal.

Regardless of the method chosen to let the data travel, the veil between analog signals and digital data must still be pierced. Once way of doing this is modulation: a base sinusoidal signal called carrier is produced, and the data are encoded by modifying this carrier. For instance, once can change the amplitude of the carrier (ASK), its frequency (FSK), or even the amplitude and the phase at the same time (QAM).



Exercise 2.8 The figure above displays an example of amplitude modulation and of frequency modulation. Which is which? For each case: is that **modulation transmitting an analog or a digital signal**?

Exercise 2.9 What's the **difference between bandwidth and transmission speed**? Why do you think they are interchangeably used in non-technical contexts?



3. Where are you?

Connecting two computers makes them much more useful than two isolated machines. However, their true potential is only unlocked when they can be connected to *lots* of other computers.

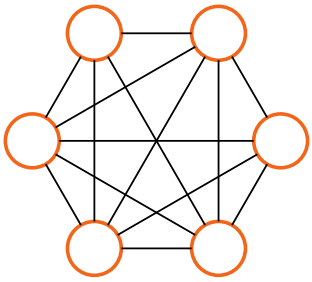
Practical limitations including cost and geographic location make it impossible to directly connect every pair of computers. Instead, devices are distributed across a graph of interconnected networks, which may extend even beyond the scale of a planet.

Whether locally within physical reach or in another continent, an addressing system is needed to identify, locate and route messages so that they reach their intended peers, just like we do with physical locations.

3.1 Please contact me

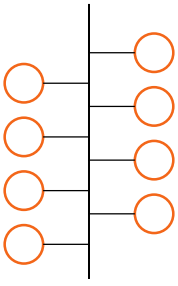
When only 2 devices are involved, one may use a point-to-point connection. In this type of connection, there is only "this side" and "the other side" of the cable. Moreover, when a device wants to communicate with the other end, there is only one link (*e.g.*, one cable) to choose from, so there is no possible confusion. Since we can operate this connection with just 2 network cards and 1 cable, point-to-point connections are viable for $N = 2$ devices.

If we want to connect $N > 2$ devices, a naïve option is to connect every device with each other in a *complete graph*. Now every device needs to handle $N - 1$ cables to the other devices, but communicating with any other device is as easy as choosing the correct cable and establishing a point-to-point connection.



The main issue with this approach is that the total number of connections is $N \cdot (N-1)/2 = O(N^2)$ (more on that in your trusted *Discrete Math* course). This means that, in order to have $N = 10$ devices connected in this way, you would need for instance 40 cables *and* 90 network cards. For $N = 1000$ devices, there would be half a million point-to-point connections, which is already quite unmanageable.

> $O(N^2)$ is notation for a quadratic complexity bound. This is *not* the same as exponential and should never be confused. (read more...)



There is one trick up our sleeves: data buses, where one or more data lines that are simultaneously connected to multiple devices. This retains the simplicity of point-to-point connections, because each device needs to handle only 1 cable and can use it to communicate with all other devices. The cost-effectiveness is also retained, since only N network cards for N devices ($O(N)$).

The main advantage of data buses is also their main weakness: all devices send and receive data using the same data line, but they cannot do it at the same time because there is only one line. If two or more do, there is a collision that makes data transmission impossible while it lasts.

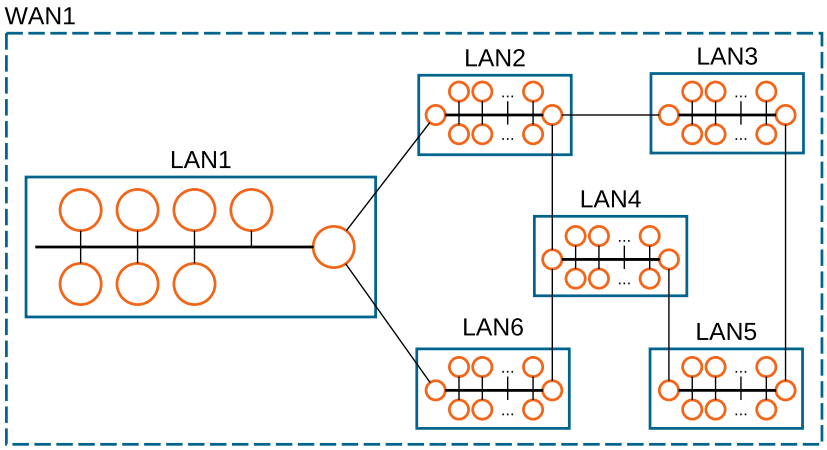
> As computer networking was developed, people experimented with many alternatives to the bus topology. One curious example is Token Ring, where devices are

connected around a circle, and messages are passed around in a single direction.

In practice, it is not trivial to implement a bus technology where new devices can be added over time. Historically, this was sometimes done by physically piercing the main bus cable, and adding a new cable towards the new computer being connected. Later on, this was simplified with devices called hubs. They worked similarly, but they offered pre-built ports and removable connectors. Nowadays, hubs have been replaced by switches. Switches offer the functionality of a bus, connecting every device with each other, and also prevent collisions. For obvious reasons, structuring connections around a switch is called a star topology.

Exercise 3.1 What is **needed in a switch** that is not needed in a hub?

Bus topologies work great at local scales, and are extensively employed in local area networks (LANs). However, it is not cost-effective, (sometimes not even physically possible) when devices are far apart. Instead, devices can be organized in separate LANs, each one conceptually a bus, and then these separate LANs can be connected using cheaper, feasible point-to-point connections. Considered together, these devices would then form a wide-area network (WAN).



3.2 I need to find you

Once again, the main advantage of data buses is also their main weakness: all devices send and receive data using the same data line. Even if collisions don't happen, every message sent into the bus is received by all devices. However, what we actually want is for our message to reach its destination, and only its destination. Let's call this the *find-my-device problem*.

The find-my-device problem is not unique of buses or LANs. On the contrary, it becomes very important when we consider WANs, where not all devices are directly interconnected. In this case, we need to make sure we can route our messages between different LANs, and that they reach their destination.

Surprisingly, not even point-to-point connections are safe from the find-my-device problem. Assuming devices A and B are connected that way, two different programs may want to exchange information at the same time, *e.g.*, some alarm control software and a music streaming app. In this scenario, you want the client and server of the alarm monitoring service to communicate with each other but not with the music streaming service, and vice-versa.

The most common solution to this problem is to use identifiers that let us differentiate between devices or even between running processes inside an operating system (OS). Some of these identifiers are referred to as addresses, precisely because they are used to find and reach a

remote device.

Addresses are typically *numerical* identifiers, that is, just a number. As such, they can be expressed in different bases, including decimal and hexadecimal. These numbers are drawn from a predefined set, and the choice of that set determines the maximum number of elements we can uniquely identify.

> Addresses expressed as `d8:43:ae:61:ed:f1` or `192.168.1.1` (as it will be seen later in the course) are also numbers we could have expressed as 237785200061937 and 3232235777, respectively (which is not as convenient).

Exercise 3.2 **How many elements can be uniquely identify** (at most) with an address we express using 5 bits? How about 10? How about 20? Express the general solution mathematically.

Exercise 3.3 Suppose there are 1000 machines in a LAN. **How many address bits are needed**? How about 800 machines? And 1100? Express the general solution mathematically.

> In 2019, we run out of Internet (v4) addresses. Whoops!

3.3 How do I get there?

In many scenarios, multiple addresses are needed to perform the desired end-to-end communication. For instance, we may need to identify a single device within a LAN, and that LAN within a WAN. Sometimes, we will need to identify a process (a running program) within that device. This is not unlike someone paying you a visit in person:

Joseph Edgar Foreman
123 South Fake St.
Adams County
Ohio

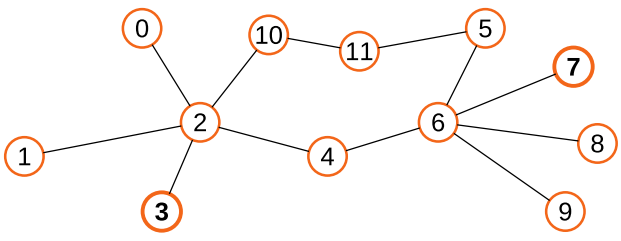
Process #1717403
Device #51688798
LAN 3141 (Data Engineering Labs)
WAN 442 (UAB)

Whatever addressing system we use, it must contain enough information to locate and reach the destination, *i.e.*, to route our messages through the graph of connections. The process of routing involves multiple steps or "jumps" across networks until the destination LAN is reached.

The name router refers to a type of network device whose job is to forward these messages across networks. These devices must contain enough information so that, when handed a message with a destination address, they can decide what network interface to use. In turn, non-router devices only need to be configured so that they can reach the next router.

> At home, your router typically also plays the role of a switch, but those are different concepts.

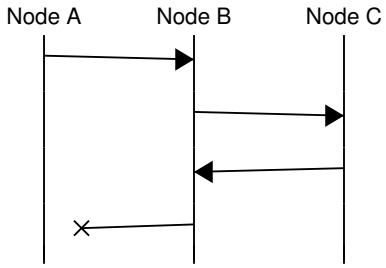
Exercise 3.4 This figure represents a handful of devices and their physical connections. These devices can communicate only by sending messages through those connections. Notwithstanding, (3) wants to send a message to (7).



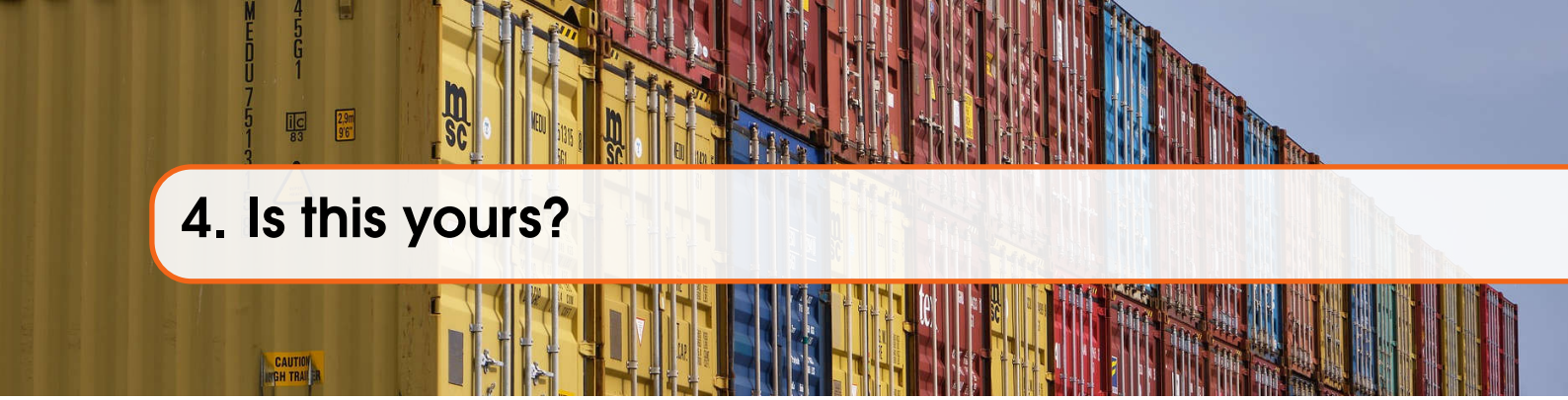
- Can you identify any devices likely to be acting as a switch? And as a router?

- What's the **minimum number of messages** sent so that ③ can contact ⑦, and ⑦ can reply to ③? List them in the order they are produced.
- What information needs to be included in those messages so that the communication ③ ↔ ⑦ may take place?

Swimlane plots like the one below are used to represent the interactions (*e.g.*, message exchange) between actors (*e.g.*, network devices). In them, the vertical axis represents time, which is invaluable to identify cause-effect relations.



Exercise 3.5 Based on your answer to Ex.3.4, **produce a swimlane diagram** that represents the message exchanges. For each message (*e.g.*, above the arrows), include the addressing information present in it.



4. Is this yours?

Information travels through this graph of networks in the form of packets, which contain a small amount of information and meta-information.

Packet format must be agreed upon by both ends of the communication, so that those packets can be efficiently and automatically produced and interpreted.

When packets travel through one or more networks, they can get lost or reordered. If a continuous stream of data must flow between two computers, packets must contain mechanisms to detect losses and restore the proper order.

4.1 I have a delivery for you

It is not feasible to establish physical connections between each pair of computers that want to communicate. Instead, relatively few connections are shared to carry messages between many pairs of devices.

If a single device transmits data continuously for a long time, that connection is blocked and may become a bottleneck that prevents other devices to communicate. To avoid this problem, connections limit the maximum amount of data that can be sent without interruption. As a result, devices are forced to send data in discrete bursts called packets. The exchange of these type of messages across one or more networks is called packet switching.

Leonard Kleinrock, one of the pioneers of internetworking, disputes the authorship of packet switching, which is often attributed to Paul Baran and Donald Davies.

Packets need to be small so that connections are not blocked for too long. At the same time, we want packets to contain as much data as possible, because each packets contain overhead data (such as the addresses discussed in Section 3) that need to be sent in addition to the user's payload data. Networks set the maximum packet length using a parameter called Maximum Transmission Unit (MTU), *e.g.*, 1500 bytes in Ethernet.

Packet transmission across a data link must be done carefully, particularly when that link is a bus shared by multiple devices. Each packet must be individually distinguishable from the rest, which can be done via at least three strategies:

1. *Fixed length*: the length of all packets is the same. The sender and the receiver must have agreeded to this value in advance.
2. *Explicit length*: the length of the packet is included in the packet, *e.g.*, using the first byte of the packet. The sender and the receiver must agree on how this information is included and how to interpret it.
3. *Escape sequences*: Certain binary escape sequences within the data have a special meaning. For instance, one could define the byte `11110000` (`0xf0`) to mean "end of packet": these bits must appear after each packet, and only then. Then the sender could start transmitting the contents of a packet, followed by `0xf0`. Both parties must agree on what escape sequences to use, what they mean, and how to send data equal to those sequences (*e.g.*, it should be possible to send `0xf0f0f0` without triggering an "end of packet" until all bytes are transmitted).

It is also possible to to signal the *beginning* of a packet. In that case, a preceding sequence called preamble is used.

Exercise 4.1 All strategies described above are used nowadays in real scenarios, depending on the features, costs and trade-offs of each one.

Discuss which of these strategies would be most appropriate for each of these scenarios:

- A single manufacturer designs the hardware and software that communicates two endpoints. A single data line must be multiplexed for multiple control commands as well as multiple data streams. The priority is efficient power and buffer usage.
- Multiple devices share a bus. They occasionally send messages of different lengths, but not a huge volume of data. The priority is cost.
- Multiple devices share a bus. They continuously send messages of different lengths, trying to maximize the effective transmission rate through the channel. The priority is throughput.

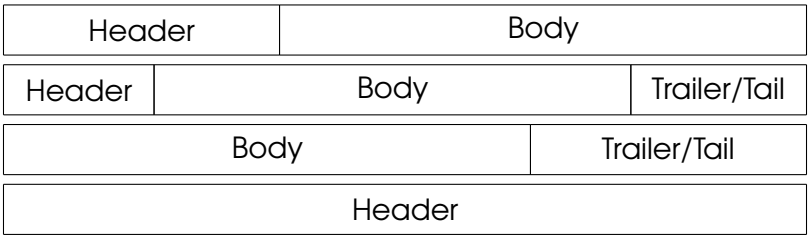
4.2 Please fill in the form

Regardless of the strategy employed, data packets typically comprise two parts:

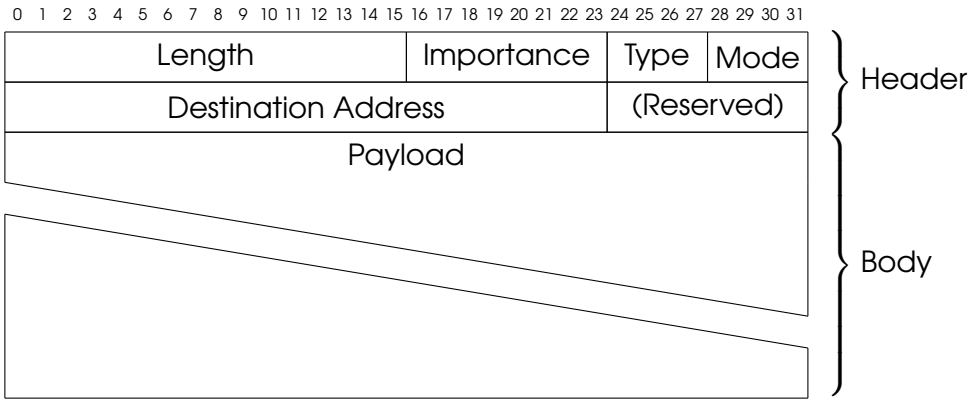
- Data payload: the information that needs to be transmitted, and
- Metadata: the meta-information needed to deliver the payload.

Length might be one of the metadata fields included in the packet, along with others that help identify their type and function. The location of payload and metadata, as well as the exact metadata fields included, their length and their meaning must be known by both ends of the communication. All these decisions constitute the packet format or packet structure.

One of the most common ways of arranging payload and metadata is with a header-body format. In this case, all meta-information is included first, followed by the data. Other formats may include a trailer (also known as tail, most often used for error detection and correction), in combination with the header, or replacing it. All of the following configurations are possible.



The format of the payload in a packet is normally user-defined. The format of the header, however, is strictly defined so that it can be easily produced and parsed. This format is often presented using a diagram that helps identify the individual bit and byte positions, like in the following (not-real) example:



Fields may also have fixed or variable lengths, which do not necessarily align with byte boundaries. Fields may also have length equal to 1 bit, in which case its is normally called a flag or flag bit.d

In diagrams like the one above, multiple lines (rows) are often used so that they can be easily printed and read. In that case, the meaning is the same as if all fields had been presented along the same line (row).

Exercise 4.2 The following packet contents were sniffed out of a network, expressed in hexadecimal. Assuming the packet has the format of the example above:

- Indicate the value of each field (length, importance, type, mode, destination address, and payload.
- Can you guess the meaning of the payload?

```
00 0c 00 f3 bb bb bb 00 68 65 6c 70
```

Exercise 4.3 The following code accepts two arguments: (1, `sys.argv[1]`) the path of an input file, and (2, `sys.argv[2]`) the path of an output file. The first 255 bytes of the contents of the input file are read, they are formatted as a packet, and the bytes of that packet are output to the output file.

.py .out snippets/simplepacketoutput.py

```
1 with (open(__file__, "rb") as input_file,
2       open("/dev/stdout", "wb") as output_file):
3     # Read at most 255 bytes from the file
4     payload: bytes = input_file.read(255)
5     # The + operation concatenates bytes
6     output_file.write(bytes(len(payload)) + payload)
```

- Describe the packet format used in the code, and its limitations.
- Extend the packet format so that
  - Packets can be longer than 255 bytes
  - The address of the destination device can be encoded
- Provide a byte diagram of the new format, as well as an explanation of the addressing system you are using.
- Modify the code so that it outputs packets of your proposed format.

Languages like Python can make a distinction between files containing text and files containing binary data. In the code above, files are open in binary mode using modes `'rb'` and `'wb'`.

When open in binary mode, file bytes are directly represented by a `bytes` object, an array of integers between 0 and 255. In text mode, those bytes are processed (decoded) further by the `open` method, and returns a string that might contain special characters like accents or non-latin glyphs. (read more...)

4.3 They keep coming

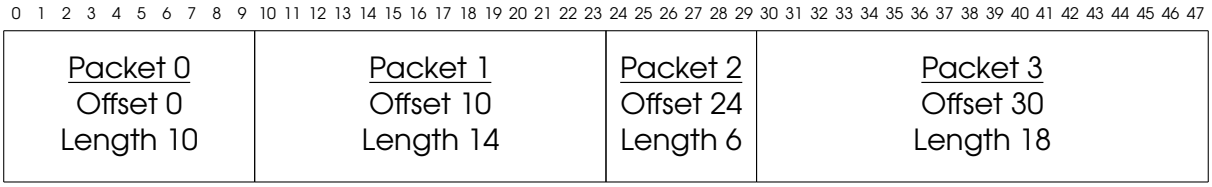
When developing applications that use networking capabilities, it is often useful to imagine communication as a stream, *i.e.*, as a conceptual pipe where we put our data (any amount of data) in one end, and it comes out at the other end. If we have this capability, then it becomes much easier to send files of any size, and to transmit a never-ending amount of audio or video. Unfortunately, all we have to simulate those streams are packets.

Packets often need to be forwarded through multiple networks. Routers are not perfect and are sometimes inoperative or improperly configured, so not all packets arrive to their destination. Moreover, different packets may be delivered through different routes that may be of different length, so packets may arrive out of order.

If we want to simulate a stream of data, packets must contain enough meta-information in them so that losses can be detected and the correct order can be restored. This introduces an overhead that is not suitable in all scenarios –*e.g.*, very low latency–, so some applications base their network communication in messages instead of streams.

When streams are required, the concept of offset is used to reassemble multiple messages into a single stream. Each packet contains some bytes of the data stream, and the offset indicates the exact location in that stream.

Exercise 4.4 The following diagram describes a stream of 48 bytes produced by a source node, as well as the packets it was split into before transmission:



- Describe the minimum header that can be used to transmit this stream.
- Assuming that the recipient receives Packet 2 first, followed by Packet 0 and other packets are lost: what bytes of the stream are known by the destination node, and what bytes are unknown?
- How can the source node know that some packets were not delivered?
- What would happen if a bogus message is received by the destination, with offset 7 bytes and length 4 bytes?

5. A Stack of Layers

Modern computer communication is achieved using a stack of network layers. Each stack focuses on some problems and is responsible for providing certain features, *e.g.*, addressing and sequencing (introduced in the previous chapters).

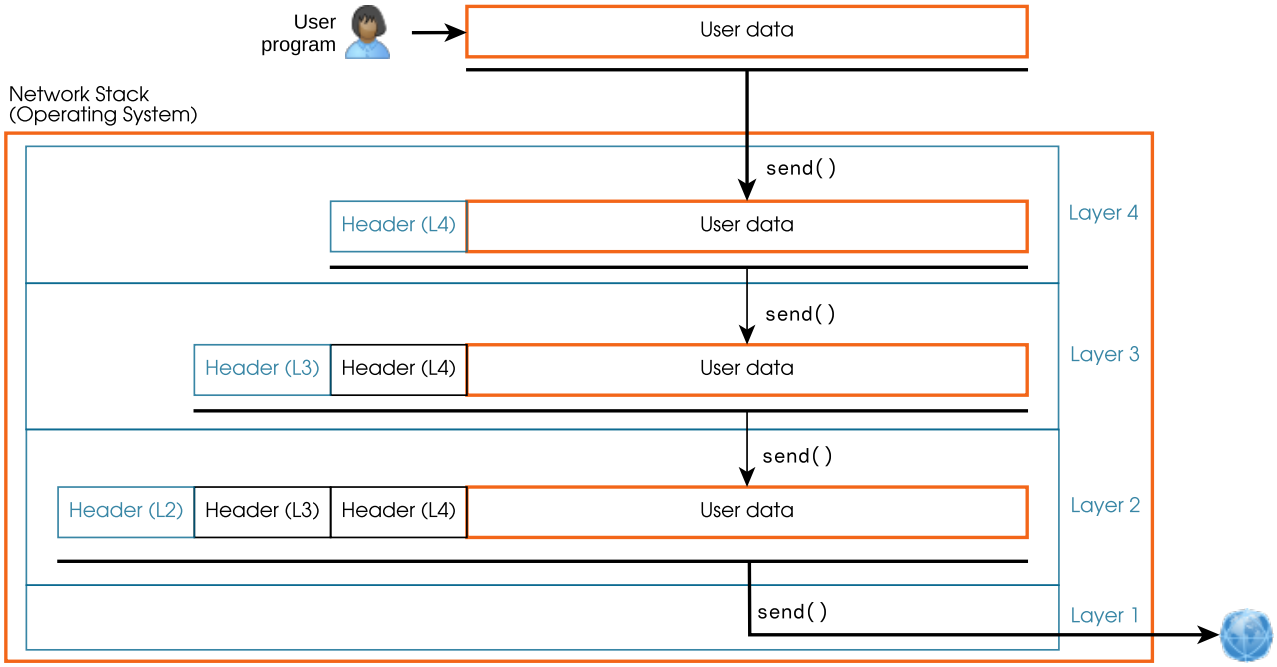
Each layer defines a `send`/`receive` function *pair*, which is responsible for some of these features. Each layer uses the `send` and `receive` of the layer below, *i.e.*, the features of one layer rely on the features of all layers below it.

To coordinate layers and enable virtual communication between them, each layer defines a different packet format, which encapsulates a packet of the layer above it.

5.1 Sending data

To implement these capabilities, each layer transmits not only the `data` provided by the layer above, but also a header with all the meta-information required for that capability. Thus, each layer needs to define the packet format, adding some overhead (typically a header) to the payload, *e.g.*:

When a user program wants to send some data, it uses the `send` method of the *top* layer of the stack, which eventually results in some data being transmitted by the physical layer, *e.g.*, through a cable or antenna. The following figure depicts the structure of a 4-layer stack:



The packets produced by each layer are generically referred to as PDUs (Protocol Data Unit). The PDUs of each layer have specific names such as frame, datagram or segment. These names must be used with precision in technical contexts.

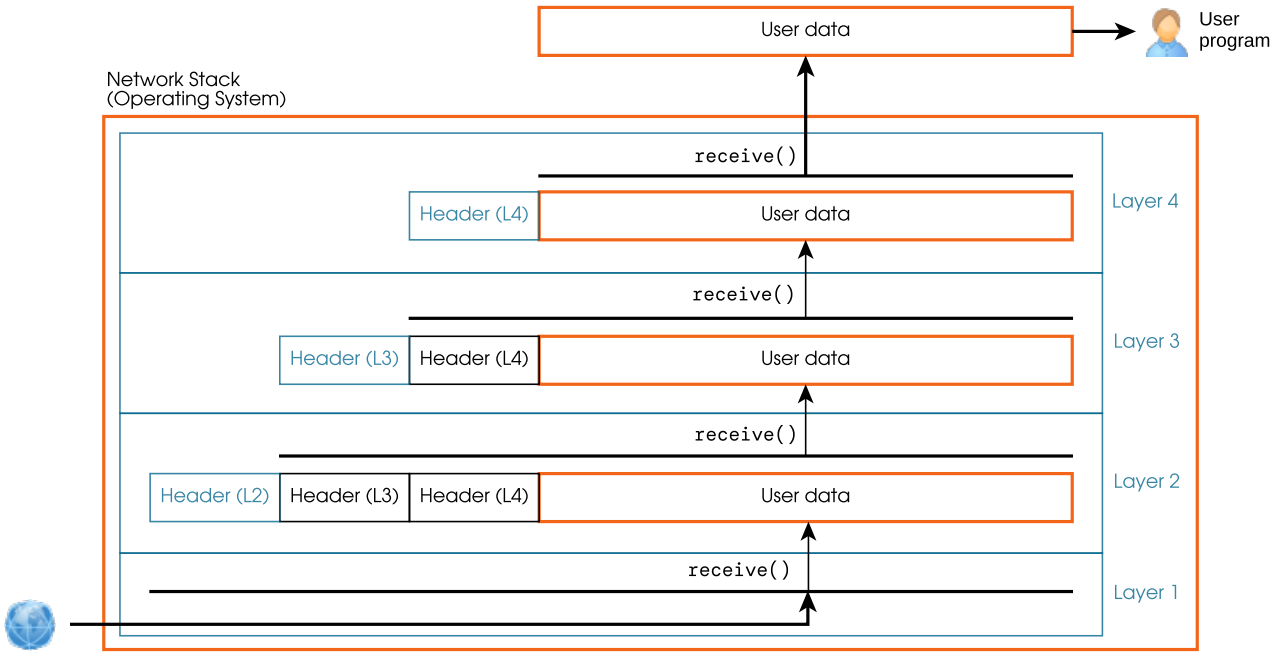
Each layer includes the PDU of the layer above inside its PDU's packet format definition. This process is referred to as encapsulation.

Exercise 5.1 The user program wants to send 420 bytes of data using the stack of the previous figure. Assume a header size for layers 2, 3 and 4 of 14 bytes, 20 bytes and 20 bytes, respectively.

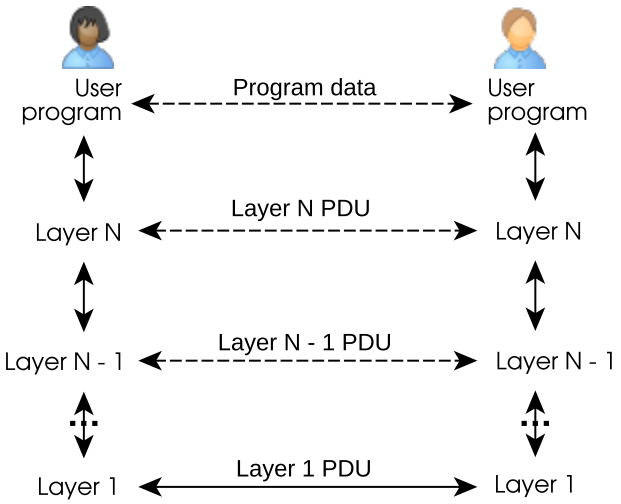
- What is the payload size and the overhead size for each layer's PDU?
- What is the payload size and the overhead size of the whole stack?

5.2 Receiving data

When a packet reaches the network stack of the recipient, it is processed in reverse order, *i.e.*, layers are traversed from bottom to top. Each layer inspects *only* the header corresponding to that layer. If everything is correct, the layer removes the header and passes the remainder of the packet to the layer above. In turn, the layer on top passes the remaining data (the payload) to the recipient. Continuing the example of the 4-layer stack:

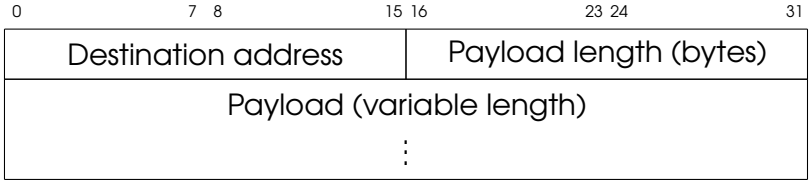


In a network stack, only the bottom layer interacts directly with the exterior, *e.g.*, only the physical network card of a computer is connected to a wire that carries data to other computers. At the same time, data sent by the bottom layer includes the PUDs of all layers by means of encapsulation. Thanks to this, there exists a virtual communication between each layer in one end and its counterpart in the other end of communication. The following figure represents the real (solid line) and virtual (dashed line) communication in our example:

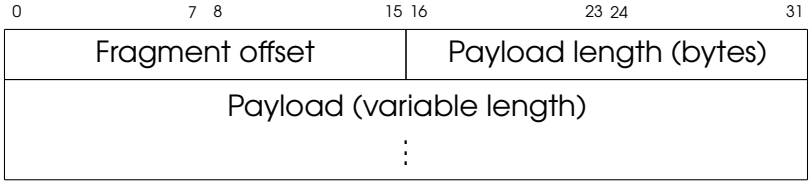


Exercise 5.2 Another network stack example has 3 layers: L1, L2 and L3 from bottom to top.

The PDU of layer L2 is as follows:



The PDU of layer L3 is as follows:



Layer L1 receives a packet of data and its `receive` function returns the following data (assume unsigned integers and big endian order when needed:

```
01 23 00 0a 01 00 00 06 00 aa 00 bb 43 20
```

- What data payload was sent to the recipient program in this packet?
- If this is the last packet of the communication, how much (payload) data was sent to the recipient user program?
- At most, how many different computers can connect using this stack?
- We send a file as the payload of this stack. What's its maximum possible size?

.py .out snippets/encapsulation.py

```python
import typing
import struct

def layer2_encapsulate(address: int, data: bytes) -> bytes:
    """Encapsulate a PDU of layer 3 into a PDU of layer 2."""
    assert 0x0000 <= address <= 0xffff, "Invalid layer 2 address"
    assert len(data) <= 0xffff, "Too much data for a layer 2 PDU"
    return bytes(struct.pack(">H", address)
                 + struct.pack(">H", len(data))
                 + data)

def layer3_encapsulate(offset: int, data: bytes) -> bytes:
    """Encapsulate a chunk of user data into a PDU of layer 3."""
    assert 0x0000 <= offset <= 0xffff, "Invalid layer 3 offset"
    assert len(data) <= 0xffff, "Too much data for a layer 3 PDU"
    return bytes(struct.pack(">H", offset)
                 + struct.pack(">H", len(data))
                 + data)

def stack_send(data: bytes, output_file: typing.BinaryIO):
    layer3_pdu = layer3_encapsulate(offset=0, data=data)
    layer2_pdu = layer2_encapsulate(address=0xabcd, data=layer3_pdu)
    output_file.write(layer2_pdu)

if __name__ == '__main__':
    with (open(__file__, "rb") as input_file,
          open("/dev/stdout", "wb") as output_file):
        payload = input_file.read()
        stack_send(data=payload, output_file=output_file)
```

Exercise 5.3 The previous code takes some data and writes the bytes that would be sent to Layer 1 (the physical layer) of the network stack of Exercise 5.2. Implement the receiving end of the stack as follows:

- Implement a `layer2_decapsulate(data: bytes) -> (int, bytes)` function that takes a Layer 2 PDU and returns the tuple `(address, payload)`, where `address` is the destination address and `payload` is the encapsulated Layer 3 PDU.
- Implement a `layer3_decapsulate(data: bytes) -> (int, bytes)` function that takes a Layer 3 PDU and returns the tuple `(offset, payload)`, where `offset` is the payload offset, and `payload` is the chunk of data sent by the other end of communication.
- Implement a `stack_receive(data: bytes)` function that decodes a Layer 1 PDU (using the two previous functions) and prints a message line similar to
  `"Received a chunk of data. Length: 12 bytes, Address: 0xabcd, Offset: 0."`
- Complete the `stack_send` function so that the input data are split in chunks of at most 100 bytes, a valid Layer 1 PDU is produced for each of those chunks, and those Layer 1 PDUs are passed to `stack_receive`.
- Would your implementation work if the Layer 1 PDUs were randomly ordered before passing them to the receiving stack?

Python's `struct` library is useful, but not mandatory, to format packet bytes.

II Internet

The previous part of this guide explored some of the main challenges of communicating multiple computers. The Internet is one working solution to those challenges: the one humans have created over the last half century, now vehicular to many social, commercial and political activities. This part studies the mean features offered by the Internet, as well as the suit of protocols employed to achieve them.

6. The Internet

Modern communications are based on layered network stacks. Maybe you just downloaded this guide from the Internet. What layers are used exactly? What protocols are employed, and with what purpose? This chapter provides a primer to answer these questions.

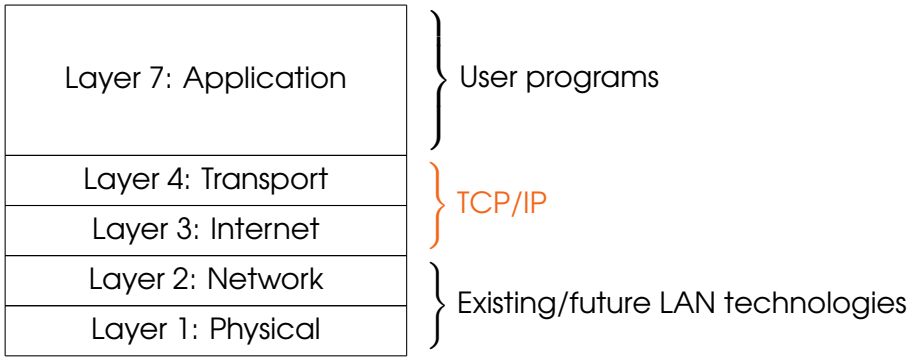
6.1 The TCP/IP stack

The Internet is a practical solution to the problem of global computer communication, whose development started in the late 1960's – early 1970's, and is still ongoing.

Packet switching was conceived during the early 1960s, and the first wide-area implementation, ARPANET, began in 1969. A key challenge at the time was to interconnect existing networks, owned by different organizations, and based on different technologies and protocols. To make things viable, all the following requirements had to be met:

- Existing networks should work without updates to the physical infrastructure or their protocols, just the way those protocols are used.
- Existing networks should remain independently managed at the local level (autonomous internal organization).
- Packets may not always be delivered, some are lost.
- Messages that arrive may not be in order, and may be repeated.

The solution adopted by ARPANET –and later by the whole world– is the TCP/IP protocol suite, first described in 1974. It is a layered approach, designed to be "plugged-in" on top of existing LANs, with the goal of offering networking capabilities to all user applications.



In a nutshell, the main services offered by these layers are as follows:

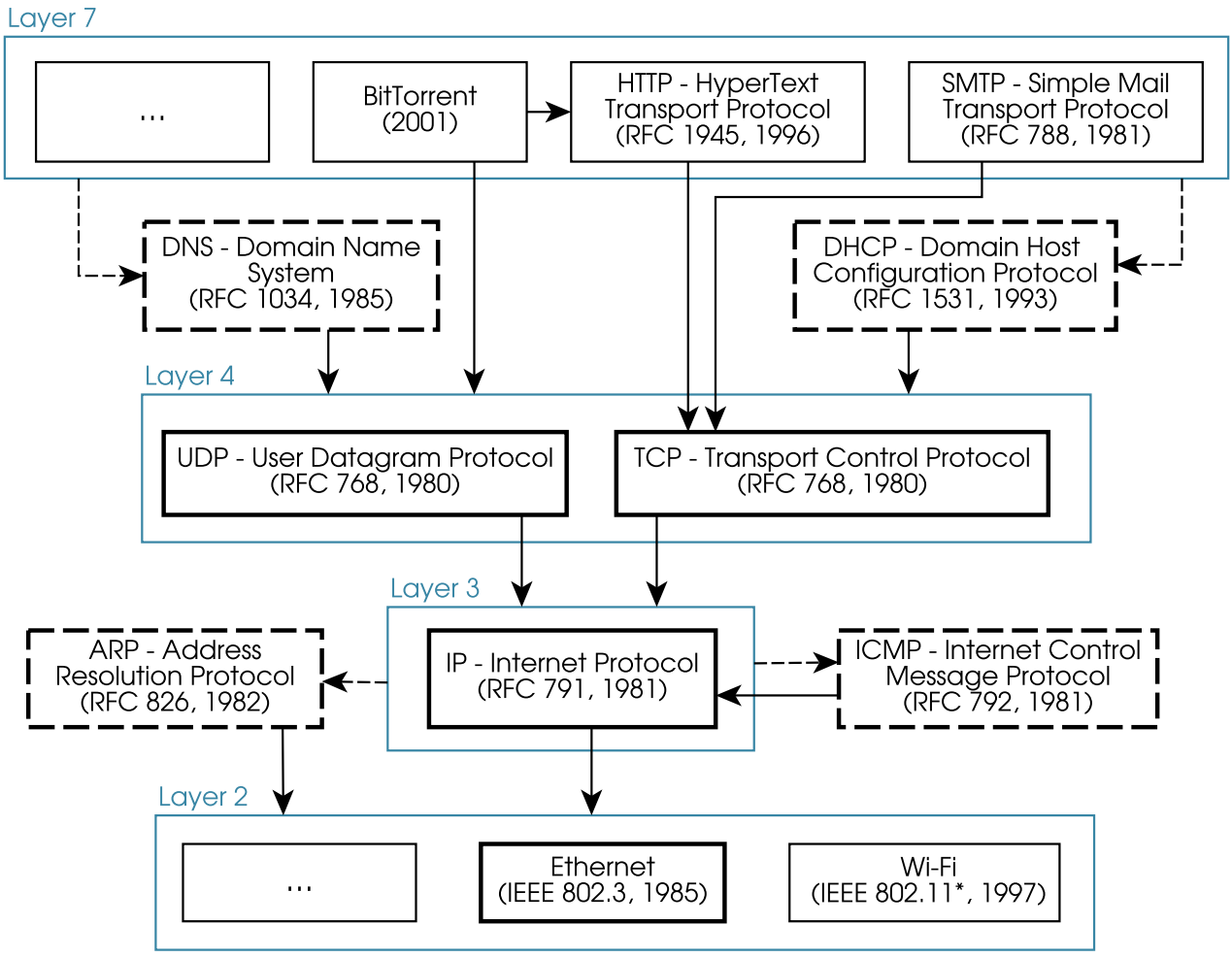
Layer 1 (Physical): Send digital data through analog media.
Layer 2 (Network): Identify and communicate devices within the same LAN.
Layer 3 (Internet): Identify and communicate devices across all LANs.
Layer 4 (Transport): Identify and communicate programs (services).
Also (choose *only* one):
a) Make sure a stream of data successfully reaches the other end, and in order – *but* an overhead is introduced and connections must be established (TCP).
b) Use simple, relatively efficient messages, and have a small overhead – *but* messages may be lost, reordered or duplicated (UDP).
Layer 7 (Application): Do anything the user might need.
Don't forget about security (*e.g.*, cryptography) or about efficiency (*e.g.*, compression): Layers 4 and below don't provide that.

6.2 TCP/IP protocol overview

Multiple protocols are defined to implement the functionality of each layer. These definitions include the required packet format, the rules to exchange those packets and their semantics. These protocols must be understood as a system, in which each part has very specific tasks.

Some of the most relevant protocols to communications using the Internet are shown in the next figure. In it:

- Solid arrows indicate a protocol being encapsulated in another. If the destination is a layer, one protocol within that layer must be used.
- Dashed boxes indicate auxiliary protocols defined to help other function. Dashed arrows indicate what layer uses what auxiliary protocol.
- Boxes with thicker lines indicate the protocols highlighted in this guide.



Exercise 6.1 We want to send the sentence "Sorry, I'll be 5 minutes late" by email. Our email client will send those data using the SMTP protocol, which in turn uses TCP (part of Layer 4). Assuming SMTP uses headers in its PDU, how many headers will the corresponding Layer-2 PDU contain?

Exercise 6.2 Why do you think Layer 3 has only one protocol (two if you count IPv4 and IPv6 separately)?

6.3 Who is the Internet boss?

The Internet Engineering Task Force (IETF) was created in 1986 to develop and publish standards to be used in the Internet. Their main output are "Request for Comment" (RFC) documents, which contain complete technical descriptions of those standards.

These RFCs are created by working groups of experts, for everyone to use freely and free of charge. Anyone can participate in these working groups, including researchers, governments

and industry stakeholders. Decisions are made when 80-90% of the participants agree, so no single entity controls the standardization process.

> "We reject kings, presidents, and voting. We believe in rough consensus and running code" — Dave Clark (IETF), 1992.
>
> See DOI 10.1109/MAHC.2006.42 for a short story of struggle for power.

RFCs standards are descriptive, not prescriptive. This means that nobody polices the correct use of these standards. Instead, manufacturers have the incentive to be compliant with the RFCs so that their products are compatible with others. The value of RFCs (and any other standard) is dependent on whether they are adopted by users. RFCs can be updated and retired for this reason.

TCP/IP protocols prevail because they are flexible and work reasonably well in most situations, even though they are not optimal in virtually any of them. Also, TCP/IP protocols are free, as opposed to privative protocols that forced the purchase of a specific vendor's solution.

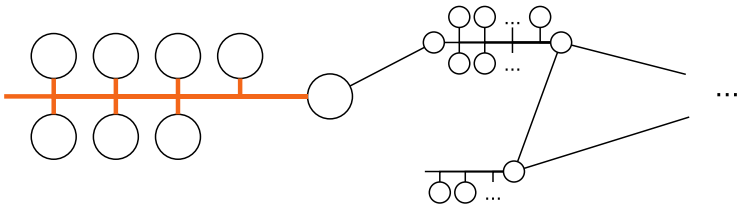
> Today it is hard to imagine a world *not* dominated by the TCP/IP architecture, but entirely different stacks were commercialized and supported until the late 90s. Examples include AppleTalk and IPX/SPX, by Apple and Novell, respectively.

Some aspects of TCP/IP like unique identifiers (including DNS, the Domain Name System) and reserved numbers need to be agreed upon by everyone for the Internet to work properly. The IANA (Internet Assigned Numbers Authority) works in coordination with the IETF to define the appropriate RFCs.

> Since 2016, IANA is managed by a nonprofit multistakeholder organization (ICANN - Internet Corporation for Assigned Names and Numbers). Since 2004, the responsibility for regional domains (*e.g.*, `.cat`, `.fr`, `.es`) is transferred to continent-level organizations called regional Internet Registries (RIRs).

7. Layer 2: Network (LAN) communication

Layer 7: Application
Layer 4: Transport
Layer 3: Internet
Layer 2: Network (LAN)
Layer 1: Physical



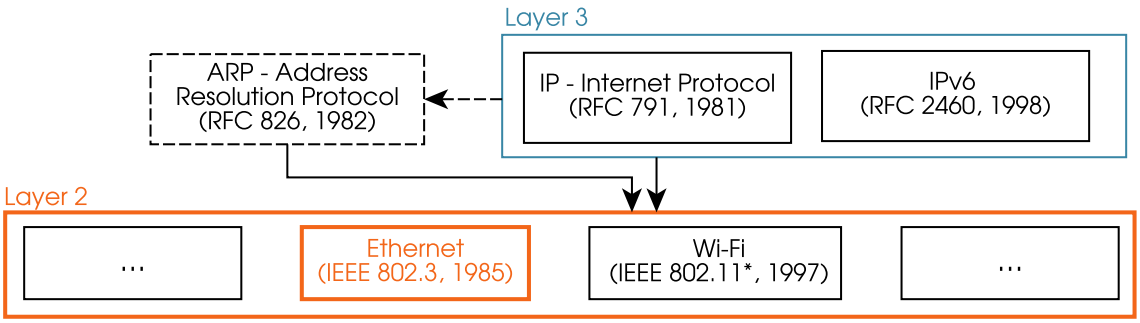
Capabilities

Layer 2 protocols allow multiple devices to exchange messages and identify each other within a local network (LAN). Devices outside the LAN cannot be contacted *directly* with Layer 2 protocols.

Layer 2 abstracts upper layers (including user applications) from the specifics of connected hardware. This makes it easier (read: cheaper) to write code once and run it everywhere.

Layer 2 messages are limited in length. The maximum amount of Payload data that can be inserted in a Layer 2 protocol is called the Maximum Transmission Unit (MTU), *e.g.*, 1500 bytes for Ethernet.

Some Layer 2 protocols may also provide user authentication and cryptographic protection; Wi-Fi is a notable example, although wired networks may also employ, *e.g.*, IEEE 802.1X.



Protocols

Many protocols have been and are being defined within Layer 2. Most importantly nowadays:

- Ethernet (IEEE 802.3), addressed in Section 7.2.
- Wi-Fi (*e.g.*, IEEE 802.11be aka Wi-Fi 7, of 2024).

Within TCP/IP, Layer 2 encapsulates the following protocol PDUs (both addressed in Chapter 8):

- IPv4 and IPv6 datagrams.
- ARP messages.

> The `ip link` command shows and can configure your interfaces, including the MAC address.
>
> Your Operating System typically assigns internal interface names automatically based on their type, *e.g.*, `eth0` or `wlan1`. Interfaces can usually be renamed, because interface names are not part of TCP/IP; they are used exclusively within that Operating system.

7.1 Layer 2 Addressing

In most LAN technologies, Layer 2 addresses are numeric IDs unique for each device within the LAN. The most frequent type are Ethernet MAC addresses (technically EUI-48) that span 6 bytes (*i.e.*, 48 bits). These are also used outside Ethernet, *e.g.*, by Wi-Fi and Bluetooth.

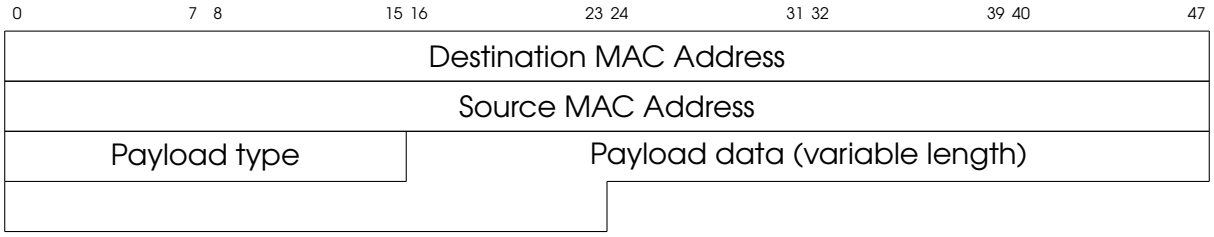
MAC addresses are most often presented to humans in the following format: `01:23:45:67:89:ab`. Some addresses have a special meaning and are not valid device identifiers. For instance, `ff:ff:ff:ff:ff:ff` is the broadcast address, meaning "everyone in the LAN". Also, `00:00:00:00:00:00` is used to represent an unknown MAC address (see RFC 9542 for a full description of reserved MAC addresses, including multicast).

Network hardware comes with a predefined, unique built-in MAC address ready to be used. Device MACs can be configured at the OS level, but remain constant while in use.

7.2 Ethernet Layer 2 Protocol

Packet format

Ethernet Layer 2 defines the following format, used in all messages. Packets of this type are called frames:



- **Destination MAC**: The MAC address of the destination device, or `ff:ff:ff:ff:ff:ff` for broadcast.
- **Source MAC**: The MAC of the device sending the frame.
- **Payload type**: Identifier for the protocol encapsulated in the payload, *e.g.*, `0x0800` → IPv4, `0x86DD` → IPv6, and `0x0806` → ARP. Sometimes called EtherType.
- **Payload**: Content requested to be sent, *e.g.*, by IPv4 or ARP. The maximum length of this field, the MTU, is 1500 bytes for ethernet.

Exercise 7.1

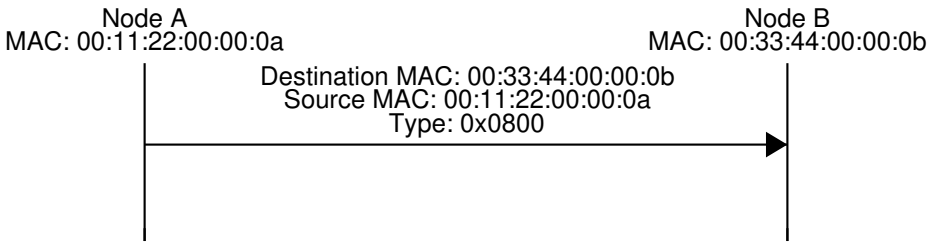
- How many different MAC addresses are there in Ethernet (including reserved blocks)?
- Are they enough so that every internet-capable device on Earth has a unique MAC?

Exercise 7.2 Is the following frame valid?

```
c2 21 90 15   bc b6 42 40   e5 f7 5e 32   08 42 77 57
03 ca 27 6b   1f 97 e9 02   24 7f 80 fa   8b 61 59 2a
81 b6 73 b0   b4 e1 75 4f   ef 40 dd 9f   34 3c 48 67
```

Operation

Messages within an Ethernet LAN are normally sent in unicast mode, *i.e.*, they are one-to-one messages. This allows switches to minimize power and bandwidth consumption. In unicast, the source device node must know the MAC of the destination.



Ethernet also supports broadcast to all nodes in the LAN. In this case, the destination MAC address must be `ff:ff:ff:ff:ff:ff`. When a device receives a frame, it discards it unless the destination MAC field in the frame is identical to its own MAC address, or broadcast.

A node will also discard a frame with an unsupported Payload Type (EtherType) field. If the payload type is supported, it is used to decide what part of the OS receives the message, *e.g.*, the IPv4 stack or the ARP subsystem.

Exercise 7.3 Continuing the example of the figure above, the network card of Node B receives a frame that begins with the following bytes. Should it be accepted by B?

```
00 11 22 00   00 0a ff ff   ff ff ff ff   08 DD ...
```

Exercise 7.4 The following scripts exemplify how to send and receive raw Ethernet frames. The client sends 5 identical frames of EtherType 0x1234 and then exits. The server waits until it receives 5 Ethernet frames of that type (checked in lines 12-13) and then exits.

- Change the client's MAC address (see page **??**) and interface name with yours.
- Make the server reject frames not addressed to it.
- Extend these scripts to implement a simple application with LAN capabilities. Suggested examples:
  - A Layer-2 echo service that distinguishes between petitions and responses.
  - A Layer-2 peer-to-peer chat that includes the origin's nickname in all messages.
  - A Layer-2 calculator service that supports basic arithmetic operations (`+`, `-`,`*`, integer division `//`, optionally division `/`).
- What is missing so that your application can connect to the rest of the Internet beyond your LAN?

You will need to run these scripts with privileges, *e.g.*, with `sudo`. Alternatively, you can permanently add the `CAP_NET_RAW` capabilities to your python binary with

```
sudo setcap cap_net_raw+ep ./venv/bin/python.
```

After that, you can run `./venv/bin/python script.py` directly without `sudo`.

snippets/ethernetclient.py

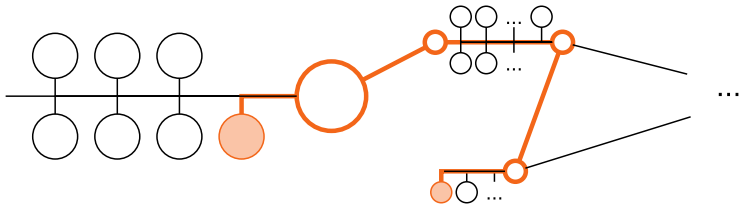
```python
import socket

interface_name = "wlp5s0"  # Needs manual configuration
source_mac = bytes([0x00, 0x11, 0x22, 0x33, 0x44, 0x55])
destination_mac = bytes([0xff, 0xff, 0xff, 0xff, 0xff, 0xff])
packet_type = bytes([0x12, 0x34])
payload_data = bytes([0x78, 0x6f, 0x69, 0x20, 0x75, 0x61, 0x62])

# This socket sends Layer 2 packets directly.
s = socket.socket(
    family=socket.AF_PACKET, type=socket.SOCK_RAW, proto=socket.ntohs(socket.ETH_P_ALL))
s.bind((interface_name, 0))

for i in range(5):
    frame = destination_mac + source_mac + packet_type + payload_data
    s.send(frame)
    print(f"Sent frame {i + 1}/5: {len(frame)=}, {payload_data=}.")
```

.py .out snippets/ethernetserver.py

```python
import socket

interface_name = "wlp5s0"  # Needs manual configuration

# This socket receives Layer 2 packets directly.
s = socket.socket(
    family=socket.AF_PACKET, type=socket.SOCK_RAW, proto=socket.ntohs(socket.ETH_P_ALL))
s.bind((interface_name, 0))

i = 0
while i < 5:
    frame = s.recv(1514)
    if frame[12:14] != bytes([0x12, 0x34]):
        continue
    i += 1
    print(f"Received frame {i + 1}/5, {len(frame)} bytes. Payload: {frame[14:]!r}")
```

8. Layer 3: Internet(work) communication

Layer 7: Application
Layer 4: Transport
Layer 3: Internet
Layer 2: Network (LAN)
Layer 1: Physical



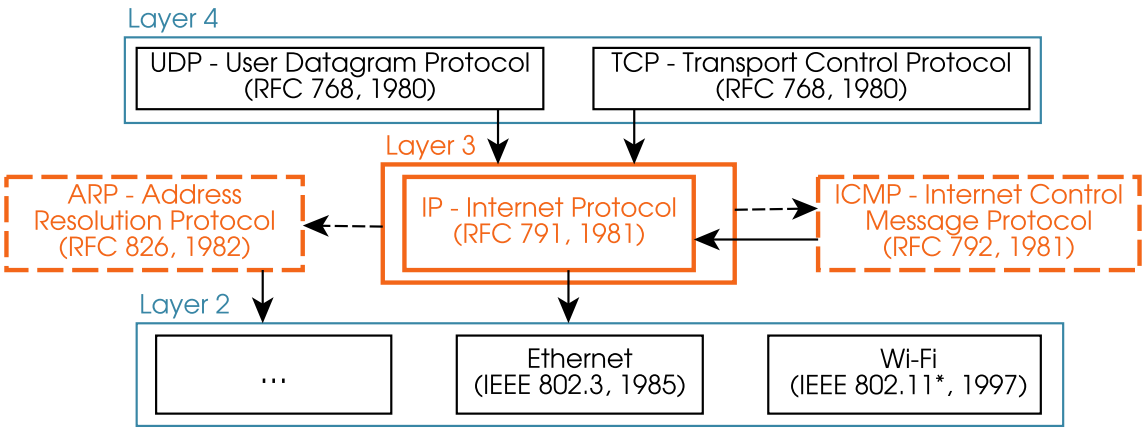
Capabilities

The Internet Protocol (IP) allows exchanging datagrams (Layer 3 PDUs) between devices from different LANs. Its addressing system identifies devices globally and makes it possible to route datagrams anywhere we need.

Devices communicating via IP may belong to LANs based on different technologies and with different types of MAC address. This has two implications:

- A datagram that "fits" in the MTU of the source LAN may be too large for the destination's (or any intermediate's) LAN MTU. IP defines a fragmentation system to split datagrams, which are reconstructed only at the final destination.
- When we send a datagram, we know the target device's IP address, but not necessarily its MAC address. Only if we are part of the destination LAN we will need that MAC. IP uses ARP to translate known IP addresses to MAC addresses of the destination LAN's type.

Datagrams may travel through networks not controlled by neither the source nor the destination, so datagrams may be lost, reordered and duplicated. Layer 3 does not provide mechanisms to deal with this; upper layers must implement protection mechanisms when required (*e.g.*, to implement TCP's data streams).



Protocols

Version 4 of IP is *the* protocol used in Layer 3, *i.e.*, it is common to virtually all communications in the Internet as we know it today. For many years, the Internet has been struggling to transition towards IPv6, but the process is not complete and only IPv4 provides global coverage.

IPv4 delegates in the ARP protocol the translation of known IP addresses into MAC addresses within each LAN. ARP does *not* use IP datagrams.

The following protocols encapsulate their PDUs in the payload of IP datagrams:

- Transport Control Protocol (TCP)
- User Datagram Protocol (UDP)
- Internet Control Message Protocol (ICMP)

The `ip address`, `ip route` and `ip neighbor` commands (among others) let you query several aspects of your IP configuration.

8.1 Layer 3 Addressing

IPv4 addresses are 32 bit long. They are most often presented to humans in the quad decimal format, *e.g.*, `142.250.201.67`.

Public and reserved addresses

Most IP addresses are public and unique across the Internet, *i.e.*, the previous IP has the same meaning worldwide: it identifies a single connected device. Many addresses are reserved/private and can only be used within a particular scope (*e.g.*, a LAN) or situation. Some of the most common are the following (there are some more):

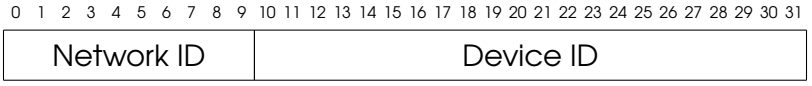
IP address block	Address scope
`127.*.*.*`	This computer (localhost)
`10.*.*.*`	This LAN
`172.16.*.*`	This LAN
`192.168.*.*`	This LAN
`0.*.*.*`	Special usages
`169.254.*.*`	Special usages
`255.255.255.255`	Special usages

Exercise 8.1

- How many different IPv4 addresses are there (including reserved and private)?
- Are they sufficient now and in the future?
- Is it problematic that an address like `192.168.0.1` is simultaneously used by thousands (likely millions) of computers in the Internet right now?

Netmasks

The 32 bits of an IP address can be divided in two parts using a netmask:



The first part identifies the LAN or group of LANs to which a device belongs. The second part identifies a single device in the LAN. The netmask determines how many bits are assigned to each part.

In the previous figure, the netmask assigns 10 bits to the network ID and 22 to the device ID. This mask can be anywhere from 0 (all bits are device ID) to 32 (all bits are network ID).

Netmasks are presented to humans in two main ways:

- `/N`, where N is the number of bits assigned to the Network ID. This is used in the CIDR format.
- The quad decimal (`A.B.C.D`) format of N bits set to `1`, followed by $(32 - N)$ bits set to `0`.

Exercise 8.2 The netmask of the previous example can be expressed in binary (although it is not very convenient):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10 bits										$32 - N = 22$ bits																					

- Express the netmask of the previous example in the CIDR and quad decimal formats.
- What is your IP and netmask? Express it in the same two main ways.

Blocks of IP addresses

Netmasks are used in combination with IP addresses to define blocks of IP addresses that begin with the same N bits (where N is the netmask). For instance, 192.168.3.0/24 (in CIDR format) and 192.168.3.0/255.255.255.0 refer to the block of addresses between 192.168.3.0 and 192.168.3.255, both included (192.168.3.* in the informal notation of the previous table).

Exercise 8.3 IP blocks can be smaller than, equal to or larger than an IP network. Provide examples where an IP block refers to:

- A single IP.
- Half the IP addresses in the 192.168.3.0/24 network.
- All addresses of the 192.168.3.0/24 network, plus 256 more addresses.
- All possible IPs.

Exercise 8.4

- Can every netmask be expressed as an IP address?
- List all possible netmasks.

When referring to a block of addresses (like a LAN), the IP address in this IP + netmask combination must have all device ID bits set to 0. This way, 192.168.3.0/24 correctly refers to a network, but 192.168.3.1/24 refers instead to a device (the one with IP 192.168.3.1) in that network (192.168.3.0/24).

Exercise 8.5 $192.168.0.0/24 \neq 192.168.0.0/25$. Why?

LANs and netmasks

Netmasks need not be multiples of 8. Consider the following example in which 192.168.76.1/19 (a device's IP + netmask) is used to obtain its network address (192.168.64.0/19):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	
192								168								76								1								Device's IP
1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	**1**	**1**	0	0	0	0	0	0	0	0	0	**1**	
&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	&	
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	Netmask
=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	=	
1	1	0	0	0	0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	**0**	**0**	0	0	0	0	0	0	0	0	0	**0**	Network IP
192								168								64								0								

Exercise 8.6

- In the previous example, we only really needed to transform one decimal value to binary, and one decimal value to binary. Which one?
- Is this always true when converting IP+netmask combinations to network addresses?
- Express the netmask in quad decimal format.

Given a IP + netmask address block and a device's IP address, it is direct to determine

whether that device's IP belongs to the block (if and only if the first N bits match, where /N is the CIDR netmask).

Exercise 8.7 Determine whether each of the following addresses belong to the same network as the following device 123.45.67.89/14:

- 23.45.67.89
- 123.45.203.25
- 123.43.255.255
- 0.45.67.1

An IP network (a LAN) is a block of addresses, but not all those addresses can be assigned to devices. Two addresses are always reserved:

- The address with all node bits set to 0: it identifies the LAN itself.
- The address with all node bits set to 1: it identifies broadcast within the LAN.

Exercise 8.8

- List all addresses that can be assigned to devices in 192.168.54.0/29.
- List the broadcast address of that network.

Exercise 8.9 The following code snippet takes an IP address and shows its 4 bytes expressed in quad decimal and in binary. Extend (or replace) the code so that you can:

- Transform 32 bits into a quad decimal string
- Display a netmask in IP address and CIDR formats
- Given a device's IP and netmask, determine whether other IP belongs to the same network.

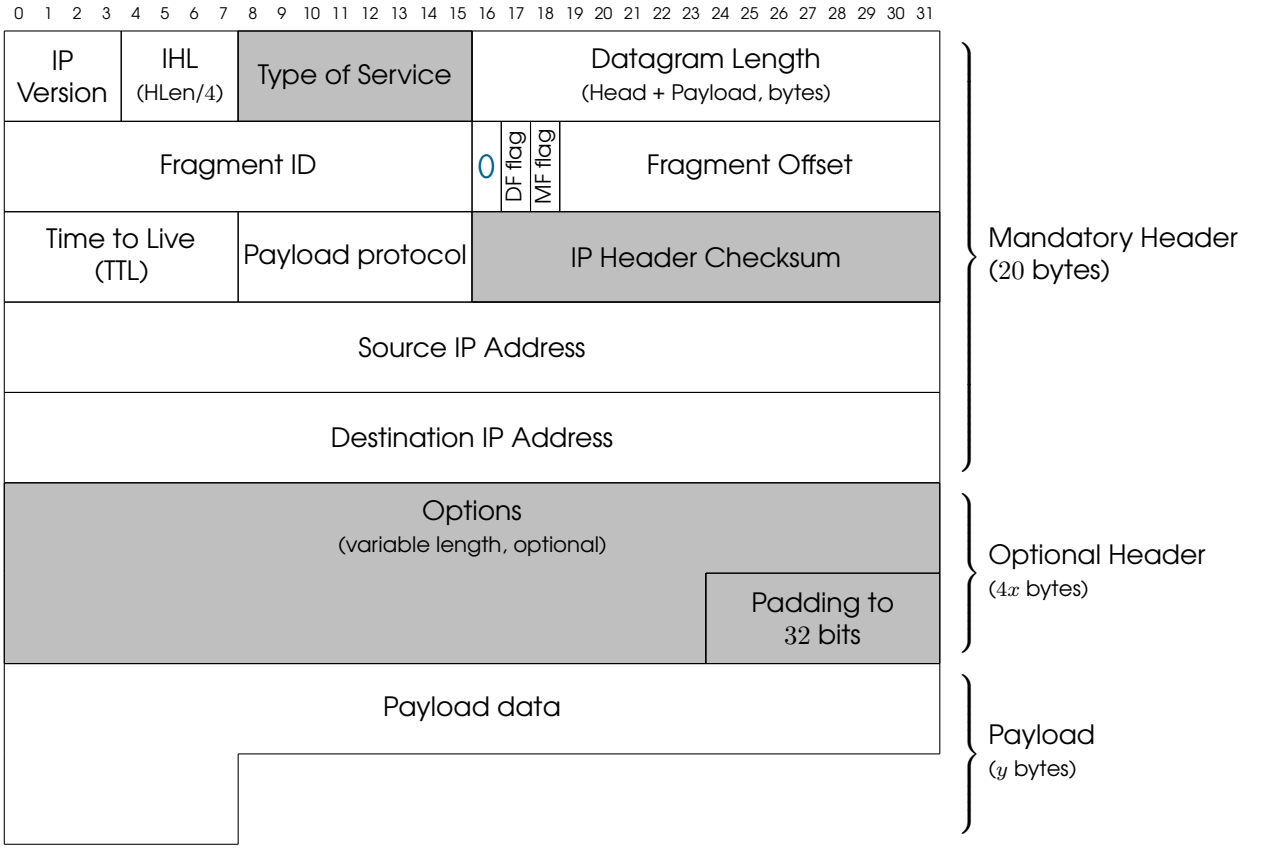
.py .out snippets/ipcalculator.py

```
def print_ip_binary(ip: str):
    parts = [int(s) for s in ip.split(".")]
    print(" ".join(f"{part:8d}" for part in parts))
    print(" ".join(f"{part:08b}" for part in parts))

print_ip_binary(ip = "192.168.0.1")
```

8.2 IP – Internet Protocol

Packet format

IP defines a packet format with a header that is *at least* 20 bytes long. Optional parts may be included, as long as the total header size is a multiple of 32 bits (4 bytes). The payload can be empty, so the minimum datagram size is 20 bytes.



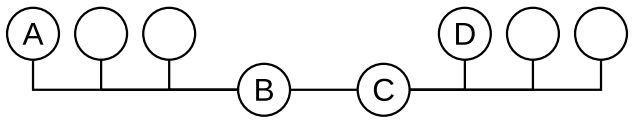
Exercise 8.10

- How can the payload length be calculated from the header?
- What is the maximum payload length?
- What are the possible values of the flags in the IP header?
- What's longer, an IP address or an Ethernet MAC address?
- Is it possible to send exactly 17 bits of payload data in an IP datagram?
- How can we know whether a datagram is encapsulating TCP or UDP data?
- Why must the optional header part by a multiple of 32 bits?

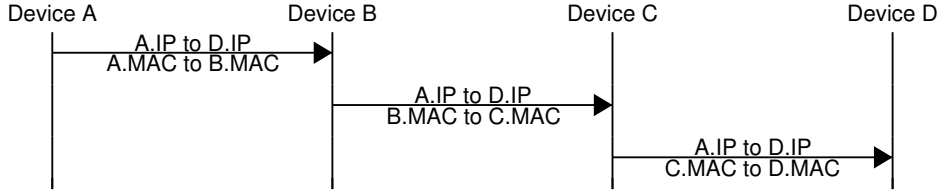
Operation

When a device A wants to send an IP datagram to a device B, the source and destination IPs in the header are fixed and don't change for this datagram.

If devices A and B belong to the same LAN, the datagram is sent directly (*e.g.*, encapsulated in an Ethernet frame). However, IP can go much farther. Consider the following scenario, where A wants to communicate with D.



Here, the source and destination devices belong to different LANs. The datagram is passed first to the source's gateway router. Then, each intermediate router passes it to the next one until the datagram reaches the last router, which is connected to the destination LAN. Finally, that last router passes the datagram to the destination.



The source and destination IP addresses don't change throughout the journey. In contrast,

MAC address change in each LAN hop.

> In this example, the gateway router (B) is directly connected to the destination LAN's router (C). Normally, there would be several routers chained between B and C.

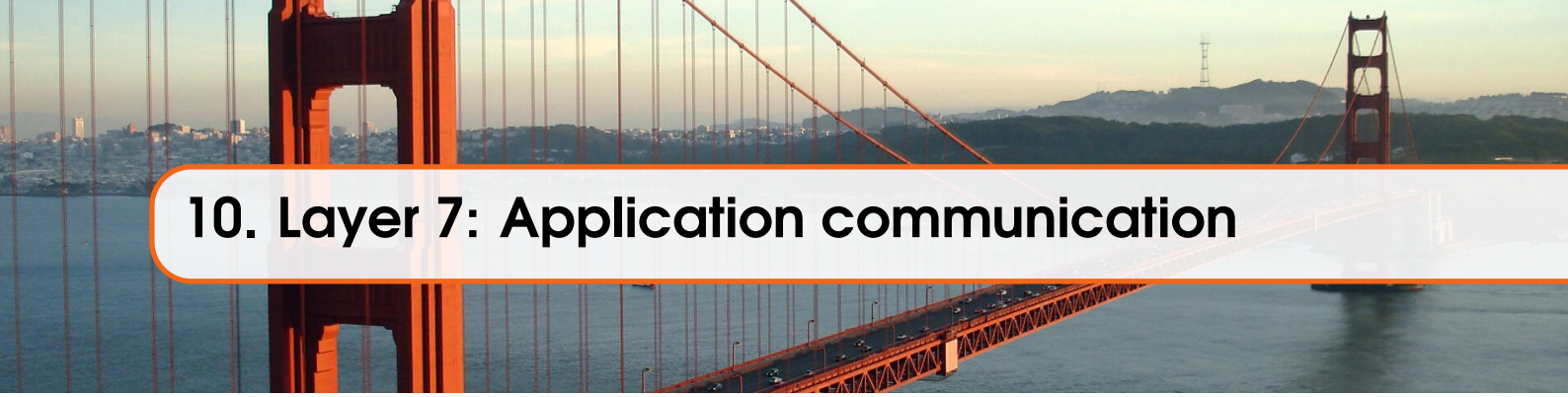
8.3 ARP – Address Resolution Protocol

> The ARP protocol does *not* use IP headers.

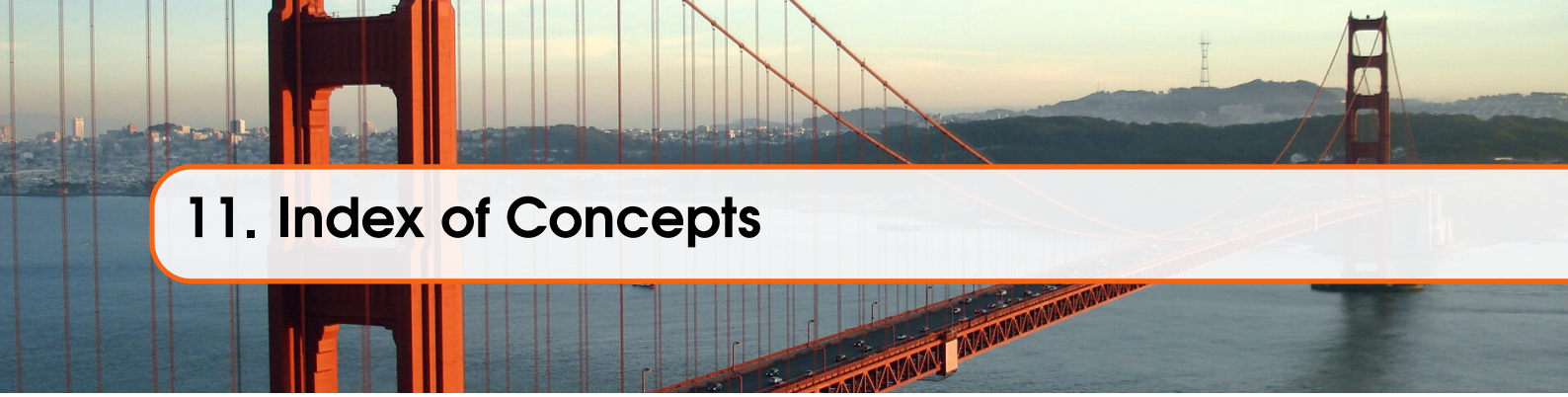
8.4 ICMP – Internet Control Message Protocol

9. Layer 4: Messages and Streams

la ventana controla el flujo porque, si envio y recivo rápido, - envío el siguiente paquete antes, y sé que puedo porque la otra parte confirma al mismo ritmo.



10. Layer 7: Application communication



11. Index of Concepts