Network Security

Introduction to networks

Radboud University, The Netherlands

Spring 2019

What is a (computer) network

Definition

A *computer network* is two or more computers that are connected, so that information can be transmitted between them.

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)?

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes (quite obviously)

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes (quite obviously)
- Facebook?

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop?

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)
- The World Wide Web?

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)
- The World Wide Web? No
  (network of documents, not of computers)

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)
- The World Wide Web? No
  (network of documents, not of computers)
- The Internet?

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)
- The World Wide Web? No
  (network of documents, not of computers)
- The Internet? Yes
  (probably the most famous computer network)

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)
- The World Wide Web? No
  (network of documents, not of computers)
- The Internet? Yes
  (probably the most famous computer network)
- Your laptop?

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)
- The World Wide Web? No
  (network of documents, not of computers)
- The Internet? Yes
  (probably the most famous computer network)
- Your laptop? Yes
  (many connected independent components ("computers"))

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)
- The World Wide Web? No
  (network of documents, not of computers)
- The Internet? Yes
  (probably the most famous computer network)
- Your laptop? Yes
  (many connected independent components ("computers"))
- The phone network?

Is X a computer network?

- Your WiFi at home (router, notebook, etc.)? Yes
  (quite obviously)
- Facebook? No
  (a network of people, not of computers)
- A USB harddrive connected to your laptop? Yes
  (a USB harddrive is actually a small computer)
- The World Wide Web? No
  (network of documents, not of computers)
- The Internet? Yes
  (probably the most famous computer network)
- Your laptop? Yes
  (many connected independent components ("computers"))
- The phone network? Yes
  (phones and backbone infrastructure are (special) computers)

Networks and protocols

Definition

A *network protocol* is a set of rules and conventions of how computers communicate in a certain network.

Networks and protocols

Definition

A *network protocol* is a set of rules and conventions of how computers communicate in a certain network.

Typical aspects of network protocols

- Message format, in particular *header format*

Networks and protocols

Definition

A *network protocol* is a set of rules and conventions of how computers communicate in a certain network.

Typical aspects of network protocols

- Message format, in particular *header format*
- Data encoding

Networks and protocols

Definition

A *network protocol* is a set of rules and conventions of how computers communicate in a certain network.

Typical aspects of network protocols

- Message format, in particular *header format*
- Data encoding
- Allowed messages and expected answers

Networks and protocols

Definition

A *network protocol* is a set of rules and conventions of how computers communicate in a certain network.

Typical aspects of network protocols

- Message format, in particular *header format*
- Data encoding
- Allowed messages and expected answers
- Session initialization and termination

Networks and protocols

Definition

A *network protocol* is a set of rules and conventions of how computers communicate in a certain network.

Typical aspects of network protocols

- Message format, in particular *header format*
- Data encoding
- Allowed messages and expected answers
- Session initialization and termination
- Synchronization of communication

A simple example: `netcat`

- Command on `tyrion`:

  ```
  netcat -lp 51966
  ```

- Command on `arya`:

  ```
  echo "Hi tyrion" | netcat tyrion 51966
  ```

A simple example: `netcat`

- Command on `tyrion`:

  ```
  netcat -lp 51966
  ```

- Command on `arya`:

  ```
  echo "Hi tyrion" | netcat tyrion 51966
  ```

Behind the scenes

- How/why does `arya` know `tyrion`? (or, what does "`tyrion`" mean from `arya`'s perspective?)

A simple example: `netcat`

- Command on `tyrion`:

```
netcat -lp 51966
```

- Command on `arya`:

```
echo "Hi tyrion" | netcat tyrion 51966
```

Behind the scenes

- How/why does `arya` know `tyrion`? (or, what does "`tyrion`" mean from `arya`'s perspective?)
- What's the magic value 51966? Could we use another one? Does it have to be the same on `tyrion` and `arya`?

A simple example: `netcat`

- Command on `tyrion`:

```
netcat -lp 51966
```

- Command on `arya`:

```
echo "Hi tyrion" | netcat tyrion 51966
```

Behind the scenes

- How/why does `arya` know `tyrion`? (or, what does "`tyrion`" mean from `arya`'s perspective?)
- What's the magic value 51966? Could we use another one? Does it have to be the same on `tyrion` and `arya`?
- How does `arya` know that information should go through the cable? Does it actually go through the cable?

A simple example: `netcat`

- Command on `tyrion`:

```
netcat -lp 51966
```

- Command on `arya`:

```
echo "Hi tyrion" | netcat tyrion 51966
```

Behind the scenes

- How/why does `arya` know `tyrion`? (or, what does "`tyrion`" mean from `arya`'s perspective?)
- What's the magic value 51966? Could we use another one? Does it have to be the same on `tyrion` and `arya`?
- How does `arya` know that information should go through the cable? Does it actually go through the cable?
- What information (sequence of bits) goes through the cable?

The lowest level: Ethernet

- Simple case: place “bits” on a cable, all computers connected to this cable can see the bits
- Ethernet specifies what “cables” and “bits” look like

The lowest level: Ethernet

- Simple case: place "bits" on a cable, all computers connected to this cable can see the bits
- Ethernet specifies what "cables" and "bits" look like
- More interesting for us: Ethernet has unique identifiers for network interfaces
- *MAC address*: 48 bit "physical address"
- MAC stands for "media access control"
- Specified in IEEE 802, not only used by Ethernet

The lowest level: Ethernet

- Simple case: place "bits" on a cable, all computers connected to this cable can see the bits
- Ethernet specifies what "cables" and "bits" look like
- More interesting for us: Ethernet has unique identifiers for network interfaces
- *MAC address*: 48 bit "physical address"
- MAC stands for "media access control"
- Specified in IEEE 802, not only used by Ethernet
- Ethernet ensures that bits are correctly transmitted
  - Transmit data in *frames*
  - Detect and recover from collisions
  - Ethernet uses a 32-bit checksum

The Ethernet frame

Preamble	Start of Delimiter	Destination MAC address	Source MAC address	802.1Q tag (optional)	Ethertype or Length	Payload	Frame check sequence (32-bit CRC)	Interpacket gap
7 Bytes	1 Byte	6 Bytes	6 Bytes	(4 Bytes)	2 Bytes	46–1500 Bytes (42–1500 Bytes)	4 Bytes	12 Bytes

- Most interesting for us: MAC addresses (and payload)
- Minimal payload size is 46 bytes (without 802.1Q tag) or 42 bytes (with 802.1Q tag)
- Gigabit Ethernet defines *Jumbo Frames* with payload >1500 bytes

From physical to logical addresses: IP

- MAC addresses are (more or less) fixed
- Not very flexible for logical configuration of networks:

From physical to logical addresses: IP

- MAC addresses are (more or less) fixed
- Not very flexible for logical configuration of networks:
  - Imagine replacing a computer
  - Imagine replacing just a network card
  - Want some hierarchy in addresses for large networks

From physical to logical addresses: IP

- MAC addresses are (more or less) fixed
- Not very flexible for logical configuration of networks:
  - Imagine replacing a computer
  - Imagine replacing just a network card
  - Want some hierarchy in addresses for large networks
- Higher-level, logical addresses provided by the *Internet Protocol* (IP)
- Multiple versions of IP, most important: IPv4 and IPv6

From physical to logical addresses: IP

- MAC addresses are (more or less) fixed
- Not very flexible for logical configuration of networks:
  - Imagine replacing a computer
  - Imagine replacing just a network card
  - Want some hierarchy in addresses for large networks
- Higher-level, logical addresses provided by the *Internet Protocol* (IP)
- Multiple versions of IP, most important: IPv4 and IPv6
- For the moment, only consider IPv4:
  - 32-bit addresses (typically written in *dotted decimal*)
  - *Hostnames*, such as `tyrion` or `arya` are aliases for such an IP address
  - Multiple mechanisms to map hostnames to IP addresses, easiest one: `/etc/hosts`

From physical to logical addresses: IP

- MAC addresses are (more or less) fixed
- Not very flexible for logical configuration of networks:
  - Imagine replacing a computer
  - Imagine replacing just a network card
  - Want some hierarchy in addresses for large networks
- Higher-level, logical addresses provided by the *Internet Protocol* (IP)
- Multiple versions of IP, most important: IPv4 and IPv6
- For the moment, only consider IPv4:
  - 32-bit addresses (typically written in *dotted decimal*)
  - *Hostnames*, such as `tyrion` or `arya` are aliases for such an IP address
  - Multiple mechanisms to map hostnames to IP addresses, easiest one: `/etc/hosts`
- Entry in `/etc/hosts` on `tyrion`:

```
192.168.42.2      arya
```

From physical to logical addresses: IP

- MAC addresses are (more or less) fixed
- Not very flexible for logical configuration of networks:
  - Imagine replacing a computer
  - Imagine replacing just a network card
  - Want some hierarchy in addresses for large networks
- Higher-level, logical addresses provided by the *Internet Protocol* (IP)
- Multiple versions of IP, most important: IPv4 and IPv6
- For the moment, only consider IPv4:
  - 32-bit addresses (typically written in *dotted decimal*)
  - *Hostnames*, such as `tyrion` or `arya` are aliases for such an IP address
  - Multiple mechanisms to map hostnames to IP addresses, easiest one: `/etc/hosts`
- Entry in `/etc/hosts` on `tyrion`:

```
192.168.42.2      arya
```

- Entry in `/etc/hosts` on `arya`:

```
192.168.42.1      tyrion
```

IP networks and addresses

- IP addresses have a *network part* and a *host part*
- Hosts with the same network part are "directly reachable"
- Access to hosts with a different network part needs to go through gateway (later in this lecture)

IP networks and addresses

- IP addresses have a *network part* and a *host part*
- Hosts with the same network part are "directly reachable"
- Access to hosts with a different network part needs to go through gateway (later in this lecture)
- Beginning of IP: First byte for network, last three bytes for host

IP networks and addresses

- IP addresses have a *network part* and a *host part*
- Hosts with the same network part are "directly reachable"
- Access to hosts with a different network part needs to go through gateway (later in this lecture)
- Beginning of IP: First byte for network, last three bytes for host
- More flexible: Classes A,B,C for different splitting of network and host part

IP networks and addresses

- IP addresses have a *network part* and a *host part*
- Hosts with the same network part are "directly reachable"
- Access to hosts with a different network part needs to go through gateway (later in this lecture)
- Beginning of IP: First byte for network, last three bytes for host
- More flexible: Classes A,B,C for different splitting of network and host part
- Today: variable-length subnet masks (VLSM):
  - Specify how far the network part reaches through a bitmask
  - Example: Netmask 255.255.255.0 means that the first 3 bytes are network part

IP networks and addresses

- IP addresses have a *network part* and a *host part*
- Hosts with the same network part are "directly reachable"
- Access to hosts with a different network part needs to go through gateway (later in this lecture)
- Beginning of IP: First byte for network, last three bytes for host
- More flexible: Classes A,B,C for different splitting of network and host part
- Today: variable-length subnet masks (VLSM):
  - Specify how far the network part reaches through a bitmask
  - Example: Netmask 255.255.255.0 means that the first 3 bytes are network part
  - Example 2: Netmask 255.224.0.0 means that the first 11 bits are network part

IP networks and addresses

- IP addresses have a *network part* and a *host part*
- Hosts with the same network part are "directly reachable"
- Access to hosts with a different network part needs to go through gateway (later in this lecture)
- Beginning of IP: First byte for network, last three bytes for host
- More flexible: Classes A,B,C for different splitting of network and host part
- Today: variable-length subnet masks (VLSM):
  - Specify how far the network part reaches through a bitmask
  - Example: Netmask 255.255.255.0 means that the first 3 bytes are network part
  - Example 2: Netmask 255.224.0.0 means that the first 11 bits are network part
  - Specify network together with mask, e.g:
  - Example: 192.168.42.0/24

Assigning an IP address to a network interface

- Network interfaces in Linux have logical names
- First ethernet interface (typically): `eth0`
- First WiFi interface (typically): `wlan0`

Assigning an IP address to a network interface

- Network interfaces in Linux have logical names
- First ethernet interface (typically): `eth0`
- First WiFi interface (typically): `wlan0`
- Configuration of IP address on `tyrion`:

```
root@tyrion# ip addr add 192.168.42.1/24 dev eth0
```

Assigning an IP address to a network interface

- Network interfaces in Linux have logical names
- First ethernet interface (typically): `eth0`
- First WiFi interface (typically): `wlan0`
- Configuration of IP address on `tyrion`:

```
root@tyrion# ip addr add 192.168.42.1/24 dev eth0
```

- Configuration of IP address on `arya`:

```
root@arya# ip addr add 192.168.42.2/24 dev eth0
```

Some special IP address ranges

- 10.0.0.0/8: Private network (not reachable from the Internet)
- 172.16.0.0/12: Private network
- 192.168.0.0/16: Private network

Some special IP address ranges

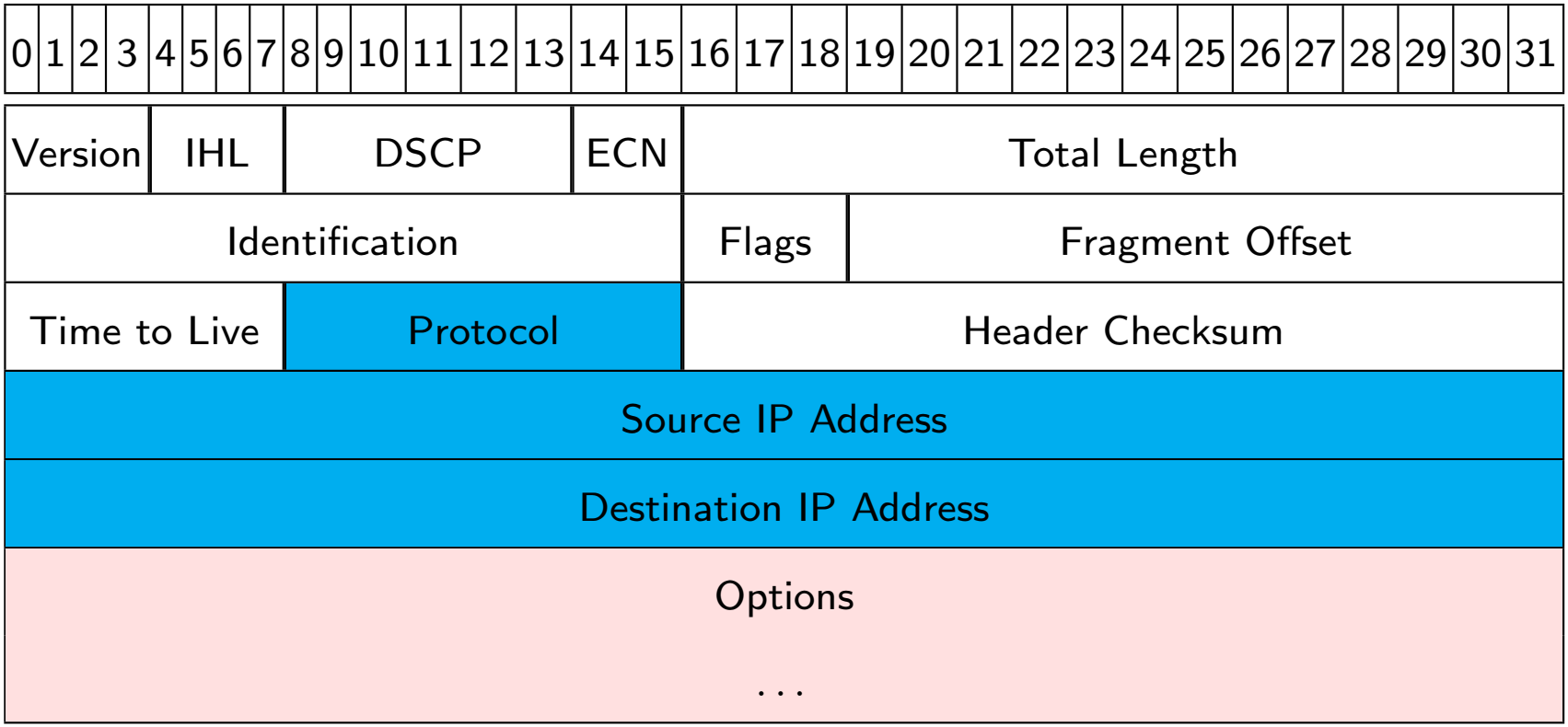
- 10.0.0.0/8: Private network (not reachable from the Internet)
- 172.16.0.0/12: Private network
- 192.168.0.0/16: Private network
- 169.254.0.0/16: Link local (direct physical connection)

Some special IP address ranges

- 10.0.0.0/8: Private network (not reachable from the Internet)
- 172.16.0.0/12: Private network
- 192.168.0.0/16: Private network
- 169.254.0.0/16: Link local (direct physical connection)
- 127.0.0.0/8: Loopback, important host: 127.0.0.1 (localhost)

Picture source: http://ars.userfriendly.org/cartoons/?id=20010523

The IP header



Address mapping from IP to MAC: ARP

- `arya` knows the IP address of `tyrion` (192.168.42.1)
- `arya` knows that she's on the same network as `tyrion`
- Needs to find out the MAC address of `tyrion`

Address mapping from IP to MAC: ARP

- `arya` knows the IP address of `tyrion` (192.168.42.1)
- `arya` knows that she's on the same network as `tyrion`
- Needs to find out the MAC address of `tyrion`
- Solution: Address resolution protocol (ARP)
- `arya` sends *ARP who has 192.168.42.1* to MAC broadcast address ff:ff:ff:ff:ff:ff

Address mapping from IP to MAC: ARP

- `arya` knows the IP address of `tyrion` (192.168.42.1)
- `arya` knows that she's on the same network as `tyrion`
- Needs to find out the MAC address of `tyrion`
- Solution: Address resolution protocol (ARP)
- `arya` sends *ARP who has 192.168.42.1* to MAC broadcast address ff:ff:ff:ff:ff:ff
- All other computers in the network check whether they have this IP address
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29

Address mapping from IP to MAC: ARP

- `arya` knows the IP address of `tyrion` (192.168.42.1)
- `arya` knows that she's on the same network as `tyrion`
- Needs to find out the MAC address of `tyrion`
- Solution: Address resolution protocol (ARP)
- `arya` sends *ARP who has 192.168.42.1* to MAC broadcast address ff:ff:ff:ff:ff:ff
- All other computers in the network check whether they have this IP address
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29
- `arya` remembers this information in the *ARP cache*

Getting to the right process: TCP

- Need to be able to distinguish bits that shall go to `netcat` and bits that go to, e.g., my browser
- Solution: Transport Control Protocol (TCP)
- TCP introduces *port* numbers
- An end-to-end connection is characterized by
  - Source IP address, destination IP address
  - Source port, destination port

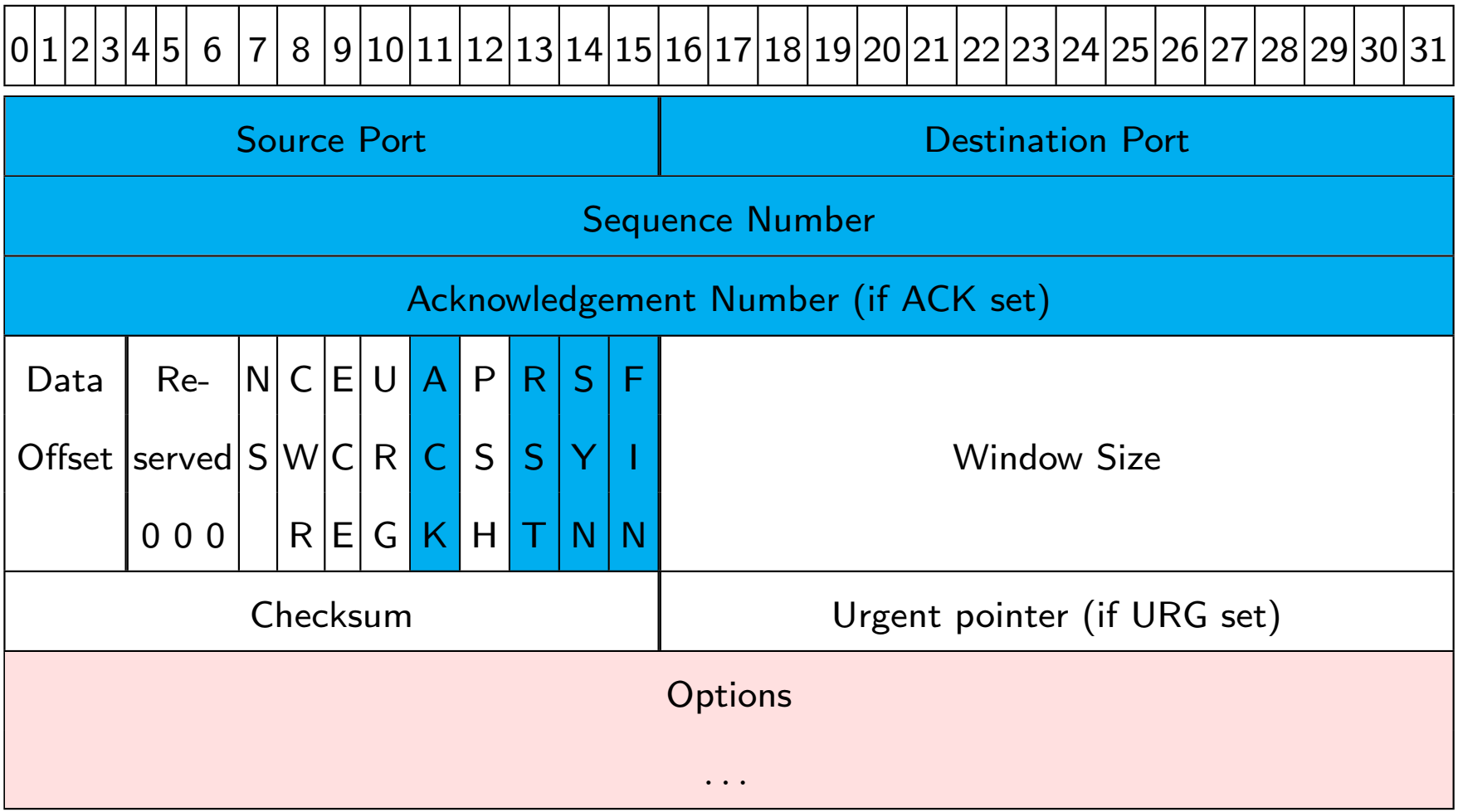
Getting to the right process: TCP

- Need to be able to distinguish bits that shall go to `netcat` and bits that go to, e.g., my browser
- Solution: Transport Control Protocol (TCP)
- TCP introduces *port* numbers
- An end-to-end connection is characterized by
  - Source IP address, destination IP address
  - Source port, destination port
- Low port numbers (<1024) are *well-known ports*
- Examples: 22 (SSH), 25 (SMTP), 80 (HTTP), 443 (HTTPS)
- Low ports can typically only be opened (used by an application) with root rights

Getting to the right process: TCP

- Need to be able to distinguish bits that shall go to `netcat` and bits that go to, e.g., my browser
- Solution: Transport Control Protocol (TCP)
- TCP introduces *port* numbers
- An end-to-end connection is characterized by
  - Source IP address, destination IP address
  - Source port, destination port
- Low port numbers (<1024) are *well-known ports*
- Examples: 22 (SSH), 25 (SMTP), 80 (HTTP), 443 (HTTPS)
- Low ports can typically only be opened (used by an application) with root rights
- TCP does *much* more than offering ports; e.g.:
  - Creates a *reliable* connection
  - Takes care of retransmissions
  - Congestion control

The TCP header



TCP sessions

- Before sending data, create TCP connection with three-way handshake:
  - Client sends SYN, SEQ=X
  - Server answers with SYN, ACK, SEQ=Y, ACK=$X + 1$
  - Client answers with ACK, SEQ=$X + 1$, ACK=$Y + 1$

TCP sessions

- Before sending data, create TCP connection with three-way handshake:
  - Client sends SYN, SEQ=X
  - Server answers with SYN, ACK, SEQ=Y, ACK=$X + 1$
  - Client answers with ACK, SEQ=$X + 1$, ACK=$Y + 1$
- Negative answer to a SYN is an RST

TCP sessions

- Before sending data, create TCP connection with three-way handshake:
  - Client sends SYN, SEQ=X
  - Server answers with SYN, ACK, SEQ=Y, ACK=$X+1$
  - Client answers with ACK, SEQ=$X+1$, ACK=$Y+1$
- Negative answer to a SYN is an RST
- During data transmission, each correctly received byte is acknowledged:
  - Sequence numbers increase with payload bytes sent
  - Acknowledgement numbers are the next expected sequence number

TCP sessions

- Before sending data, create TCP connection with three-way handshake:
  - Client sends SYN, SEQ=X
  - Server answers with SYN, ACK, SEQ=Y, ACK=$X+1$
  - Client answers with ACK, SEQ=$X+1$, ACK=$Y+1$
- Negative answer to a SYN is an RST
- During data transmission, each correctly received byte is acknowledged:
  - Sequence numbers increase with payload bytes sent
  - Acknowledgement numbers are the next expected sequence number
- Termination of a connection uses a 4-way handshake:
  - Each side terminates independently (through a FIN)
  - Each side acknowledges the FIN of the other side

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1
- `arya` sends an ARP request (ARP who has 192.168.42.1) to the broadcast MAC address ff:ff:ff:ff:ff:ff

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1
- `arya` sends an ARP request (ARP who has 192.168.42.1) to the broadcast MAC address ff:ff:ff:ff:ff:ff
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1
- `arya` sends an ARP request (ARP who has 192.168.42.1) to the broadcast MAC address ff:ff:ff:ff:ff:ff
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29
- `arya` establishes a TCP/IP connection to `tyrion` (TCP port 51966)

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1
- `arya` sends an ARP request (ARP who has 192.168.42.1) to the broadcast MAC address ff:ff:ff:ff:ff:ff
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29
- `arya` establishes a TCP/IP connection to `tyrion` (TCP port 51966)
- `arya` sends `"Hi tyrion"` through the established TCP connection
  - TCP segment with destination port 51966
  - IP packet with source IP addr: 192.168.42.2 and dest. IP addr: 192.168.42.1
  - Ethernet frame with source MAC addr: 00:1e:68:b8:c7:eb, and dest. MAC addr: 50:7b:9d:f4:db:29

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1
- `arya` sends an ARP request (ARP who has 192.168.42.1) to the broadcast MAC address ff:ff:ff:ff:ff:ff
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29
- `arya` establishes a TCP/IP connection to `tyrion` (TCP port 51966)
- `arya` sends `"Hi tyrion"` through the established TCP connection
  - TCP segment with destination port 51966
  - IP packet with source IP addr: 192.168.42.2 and dest. IP addr: 192.168.42.1
  - Ethernet frame with source MAC addr: 00:1e:68:b8:c7:eb, and dest. MAC addr: 50:7b:9d:f4:db:29
- `arya` sends all the bits of the Ethernet frame through the cable

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1
- `arya` sends an ARP request (ARP who has 192.168.42.1) to the broadcast MAC address ff:ff:ff:ff:ff:ff
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29
- `arya` establishes a TCP/IP connection to `tyrion` (TCP port 51966)
- `arya` sends `"Hi tyrion"` through the established TCP connection
  - TCP segment with destination port 51966
  - IP packet with source IP addr: 192.168.42.2 and dest. IP addr: 192.168.42.1
  - Ethernet frame with source MAC addr: 00:1e:68:b8:c7:eb, and dest. MAC addr: 50:7b:9d:f4:db:29
- `arya` sends all the bits of the Ethernet frame through the cable
- `tyrion` receives frame, opens it up, hands payload of TCP segment to the application listening on port 51966 (netcat)

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1
- `arya` sends an ARP request (ARP who has 192.168.42.1) to the broadcast MAC address ff:ff:ff:ff:ff:ff
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29
- `arya` establishes a TCP/IP connection to `tyrion` (TCP port 51966)
- `arya` sends `"Hi tyrion"` through the established TCP connection
  - TCP segment with destination port 51966
  - IP packet with source IP addr: 192.168.42.2 and dest. IP addr: 192.168.42.1
  - Ethernet frame with source MAC addr: 00:1e:68:b8:c7:eb, and dest. MAC addr: 50:7b:9d:f4:db:29
- `arya` sends all the bits of the Ethernet frame through the cable
- `tyrion` receives frame, opens it up, hands payload of TCP segment to the application listening on port 51966 (netcat)
- `tyrion` confirms receipt of the TCP segment (ACK)

Short recap: putting it together

- `tyrion`'s netcat "listens" on TCP port 51966
- `arya` looks up the IP address of `tyrion`: 192.168.42.1
- `arya` sends an ARP request (ARP who has 192.168.42.1) to the broadcast MAC address ff:ff:ff:ff:ff:ff
- `tyrion` answers with ARP reply 192.168.42.1 is at 50:7b:9d:f4:db:29
- `arya` establishes a TCP/IP connection to `tyrion` (TCP port 51966)
- `arya` sends `"Hi tyrion"` through the established TCP connection
  - TCP segment with destination port 51966
  - IP packet with source IP addr: 192.168.42.2 and dest. IP addr: 192.168.42.1
  - Ethernet frame with source MAC addr: 00:1e:68:b8:c7:eb, and dest. MAC addr: 50:7b:9d:f4:db:29
- `arya` sends all the bits of the Ethernet frame through the cable
- `tyrion` receives frame, opens it up, hands payload of TCP segment to the application listening on port 51966 (netcat)
- `tyrion` confirms receipt of the TCP segment (ACK)
- `arya` closes the session, `tyrion` closes the session

Network sockets

- Berkeley sockets are a network and inter-process communication API
- Most important functionalities of Berkeley sockets:
  - **socket():** creates a new socket

Network sockets

- Berkeley sockets are a network and inter-process communication API
- Most important functionalities of Berkeley sockets:
  - **socket():** creates a new socket
  - **bind():** binds a socket to an address and port (typically on the server)

Network sockets

- Berkeley sockets are a network and inter-process communication API
- Most important functionalities of Berkeley sockets:
  - **socket():** creates a new socket
  - **bind():** binds a socket to an address and port (typically on the server)
  - **listen():** go into "listen" state, announce size of input buffer (server side)

Network sockets

- Berkeley sockets are a network and inter-process communication API
- Most important functionalities of Berkeley sockets:
  - **socket():** creates a new socket
  - **bind():** binds a socket to an address and port (typically on the server)
  - **listen():** go into "listen" state, announce size of input buffer (server side)
  - **accept():** block until connection then create new communication socket (server side)

Network sockets

- Berkeley sockets are a network and inter-process communication API
- Most important functionalities of Berkeley sockets:
  - **socket():** creates a new socket
  - **bind():** binds a socket to an address and port (typically on the server)
  - **listen():** go into "listen" state, announce size of input buffer (server side)
  - **accept():** block until connection then create new communication socket (server side)
  - **connect():** Connect to remote "listening" socket

Network sockets

- Berkeley sockets are a network and inter-process communication API
- Most important functionalities of Berkeley sockets:
  - **socket():** creates a new socket
  - **bind():** binds a socket to an address and port (typically on the server)
  - **listen():** go into "listen" state, announce size of input buffer (server side)
  - **accept():** block until connection then create new communication socket (server side)
  - **connect():** Connect to remote "listening" socket
  - **send():** Send data through connected socket

Network sockets

- Berkeley sockets are a network and inter-process communication API
- Most important functionalities of Berkeley sockets:
  - **socket():** creates a new socket
  - **bind():** binds a socket to an address and port (typically on the server)
  - **listen():** go into "listen" state, announce size of input buffer (server side)
  - **accept():** block until connection then create new communication socket (server side)
  - **connect():** Connect to remote "listening" socket
  - **send():** Send data through connected socket
  - **recv():** Receive data from connected socket

Network sockets

- Berkeley sockets are a network and inter-process communication API
- Most important functionalities of Berkeley sockets:
  - **socket():** creates a new socket
  - **bind():** binds a socket to an address and port (typically on the server)
  - **listen():** go into "listen" state, announce size of input buffer (server side)
  - **accept():** block until connection then create new communication socket (server side)
  - **connect():** Connect to remote "listening" socket
  - **send():** Send data through connected socket
  - **recv():** Receive data from connected socket
  - **close():** Close the connection, release resources allocated to socket

“netcat” client in Python

```
#!/usr/bin/env python

import socket

host = 'tyrion'
port = 51966
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host,port))
s.send('Hi tyrion\n')
s.close()
```

“netcat -l” in Python

```
#!/usr/bin/env python

import socket

host = ''
port = 51966
backlog = 5
bufsize = 1024

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((host,port))
s.listen(backlog)

client, address = s.accept()
data = client.recv(bufsize)
if data:
    print data
    client.close()
```

The Internet protocol suite

- The "Internet protocol suite" knows four layers:
- The *link layer*:
  - Communication of directly connected nodes
  - Physical channel, media access control
  - Examples: Ethernet (IEEE 802.3), WiFi (IEEE 802.11)

The Internet protocol suite

- The "Internet protocol suite" knows four layers:
- The *link layer*:
  - Communication of directly connected nodes
  - Physical channel, media access control
  - Examples: Ethernet (IEEE 802.3), WiFi (IEEE 802.11)
- The internet layer:
  - Connect hosts across networks
  - Host identification and routing
  - Most importantly: IP, ICMP, ARP, DHCP

The Internet protocol suite

- The "Internet protocol suite" knows four layers:
- The *link layer*:
  - Communication of directly connected nodes
  - Physical channel, media access control
  - Examples: Ethernet (IEEE 802.3), WiFi (IEEE 802.11)
- The internet layer:
  - Connect hosts across networks
  - Host identification and routing
  - Most importantly: IP, ICMP, ARP, DHCP
- The transport layer:
  - Provides end-to-end communication
  - Can provide a reliable connection (retransmission etc.)
  - Most importantly: TCP, UDP

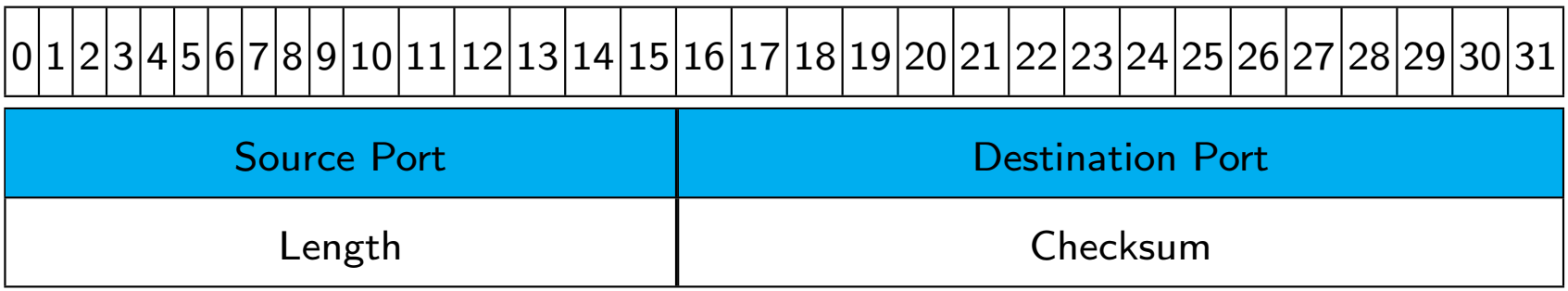
The Internet protocol suite

- The "Internet protocol suite" knows four layers:
- The *link layer*:
  - Communication of directly connected nodes
  - Physical channel, media access control
  - Examples: Ethernet (IEEE 802.3), WiFi (IEEE 802.11)
- The internet layer:
  - Connect hosts across networks
  - Host identification and routing
  - Most importantly: IP, ICMP, ARP, DHCP
- The transport layer:
  - Provides end-to-end communication
  - Can provide a reliable connection (retransmission etc.)
  - Most importantly: TCP, UDP
- The application layer:
  - Process-to-Process communication
  - Examples: HTTP, SSH, SMTP

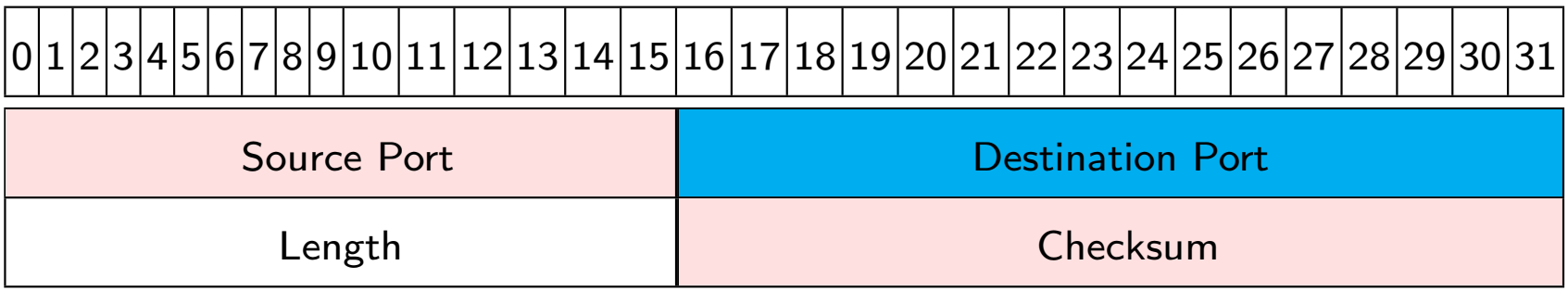
Lightweight communication: UDP

- For some messages you do not have to ensure that they arrive
- A TCP session for sending "Hi tyrion" is like cracking nuts with a sledgehammer
- Solution: User Datagram Protocol (UDP):
  - No session initialization
  - No session termination
  - No acknowledgements
  - No guaranteed transmission
- "Send your data and hope for the best"

The UDP header



The UDP header



- The Source Port and the Checksum are even optional

Control messages: ICMP

- ICMP stands for Internet Control Message Protocol
- Provides diagnostics and control on the internet layer
- ICMP messages are in the IP payload (protocol number 1)
- Most important ICMP messages:
  - Echo request and Echo reply ("ping")
  - Destination unreachable
  - Redirect message
  - Source quench