Network Security - part I

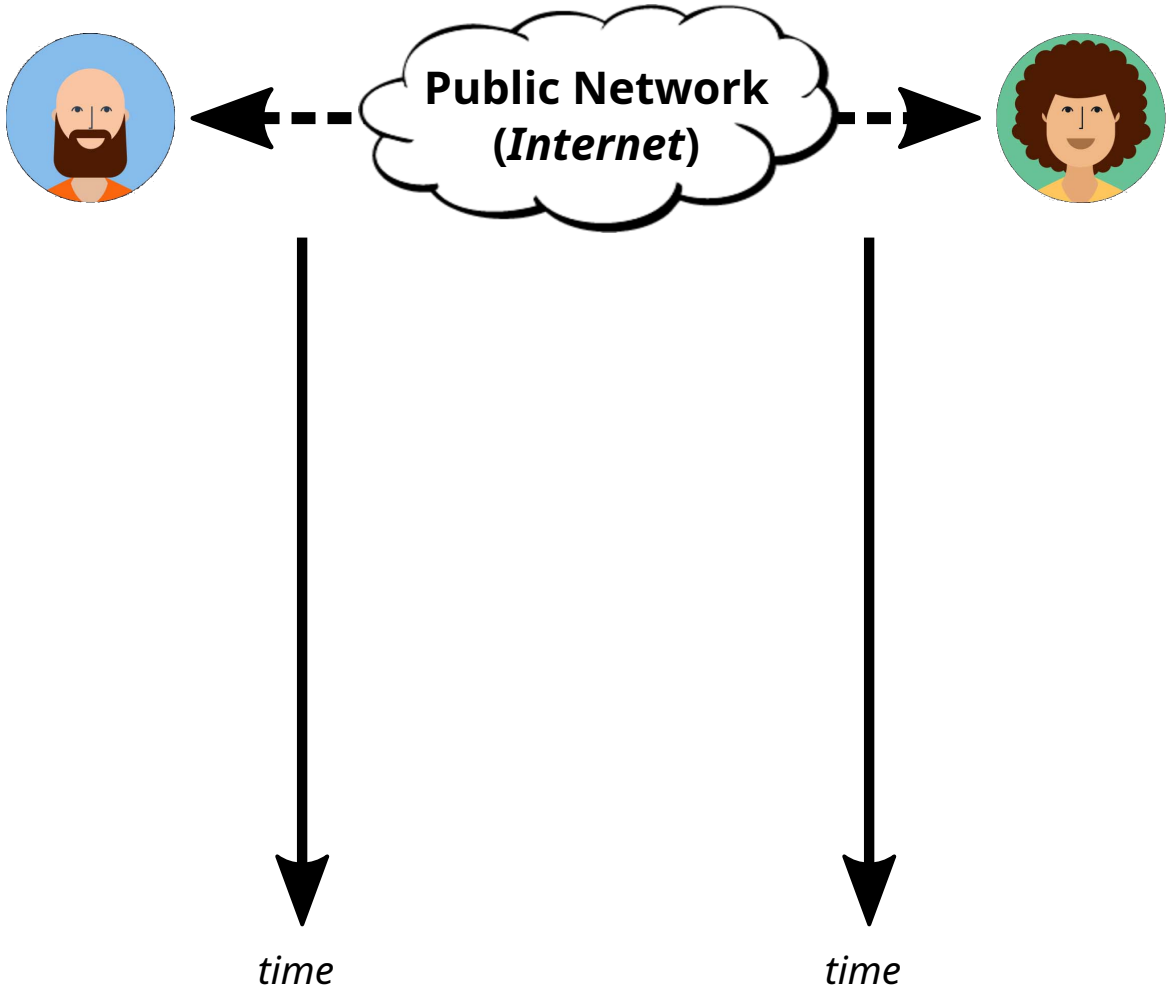
Srdjan Matic

16th September 2024

based on the material prepared by:
Emiliano de Cristofaro and Juan Caballero

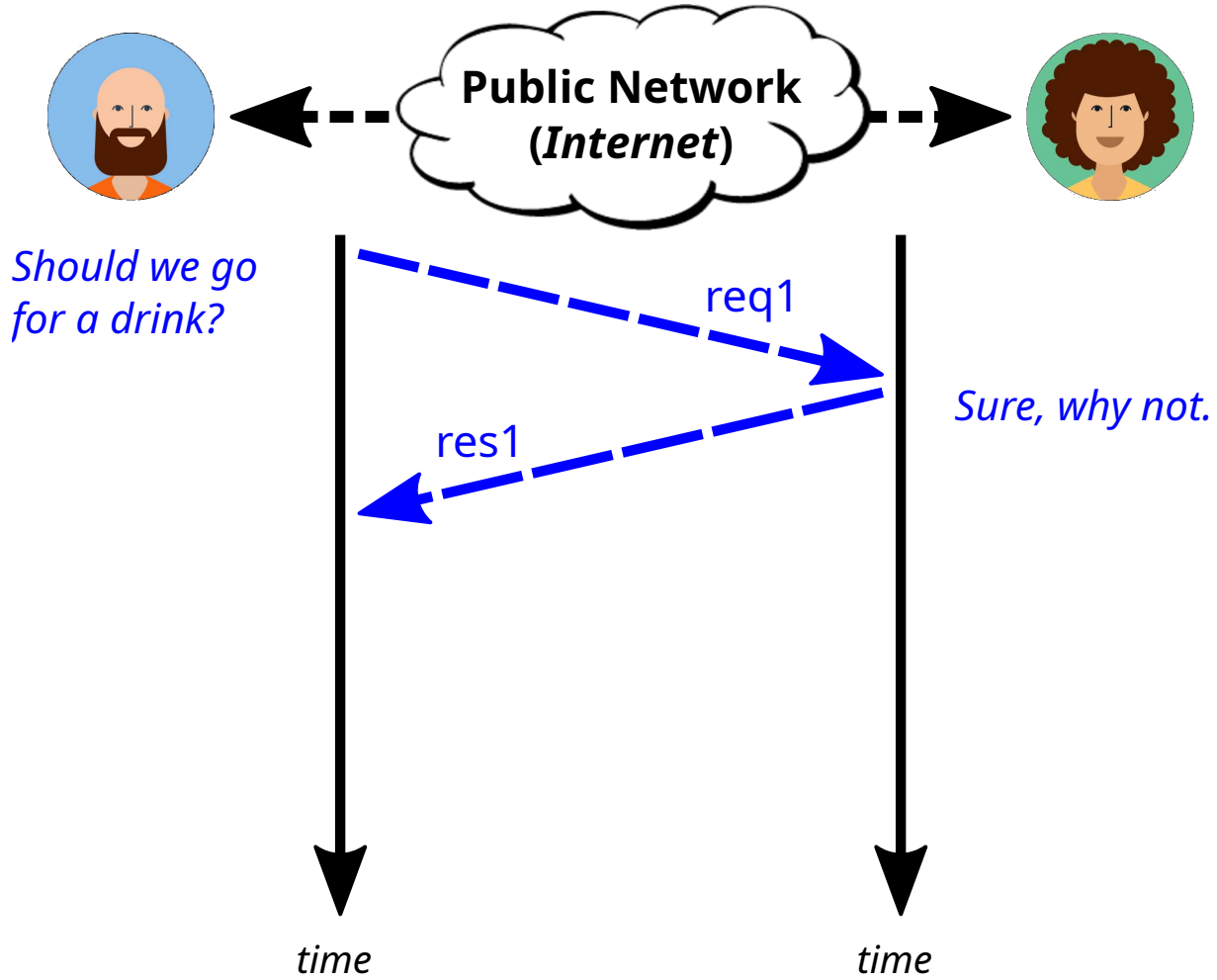
Outline

An Example of a (Network) Communication



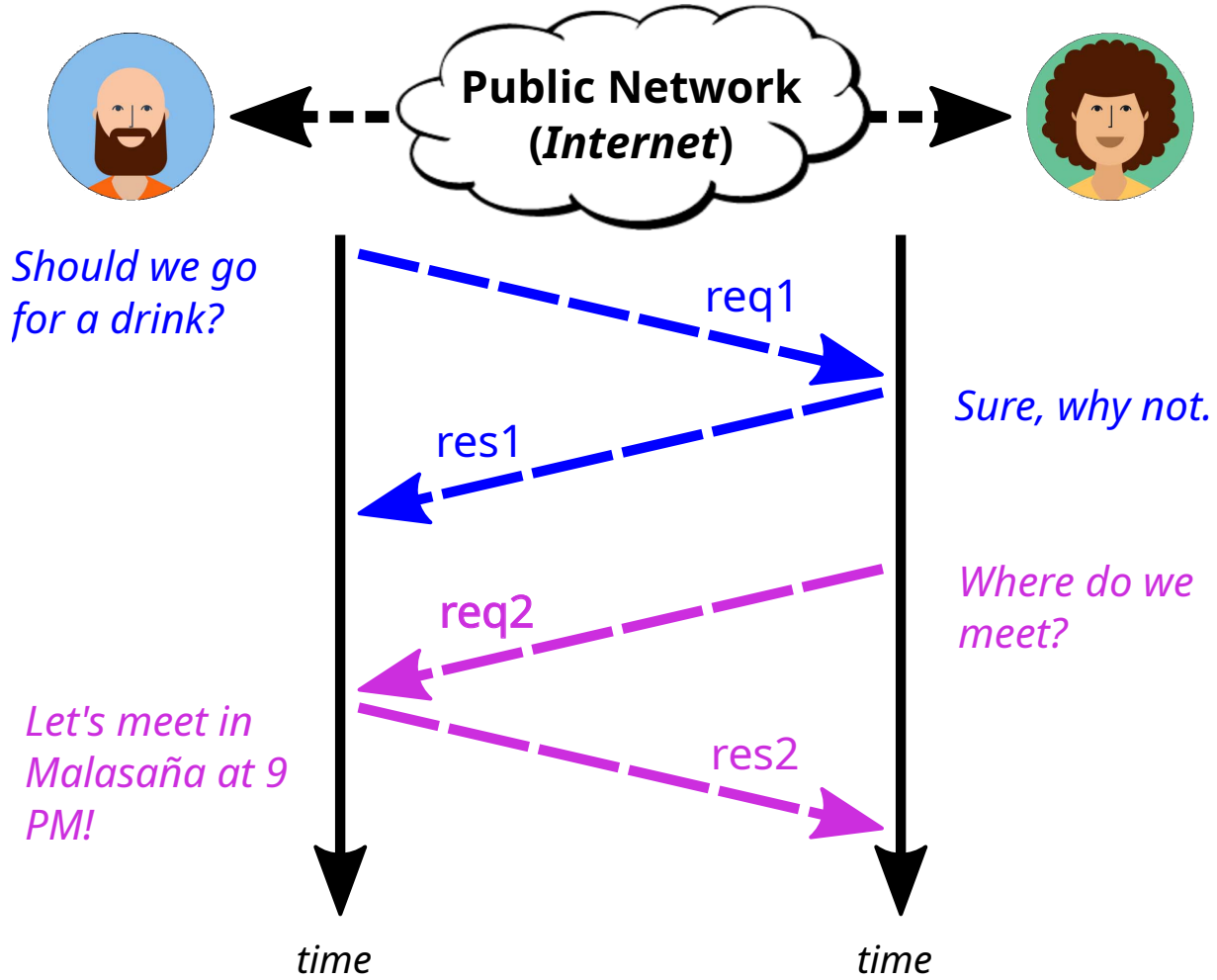
- Only two participants.

An Example of a (Network) Communication



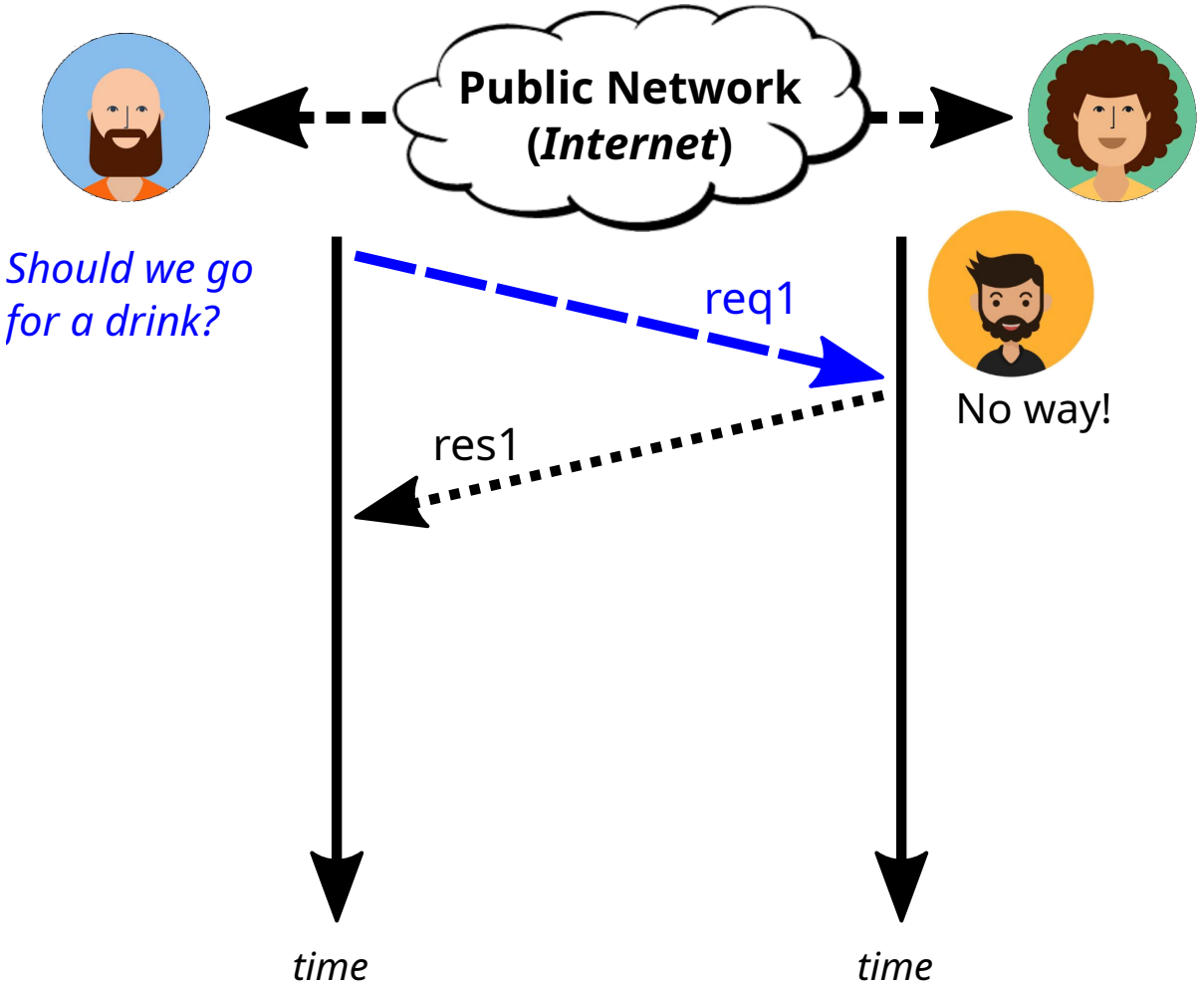
- Only two participants.
- Bi-directional communications.

An Example of a (Network) Communication



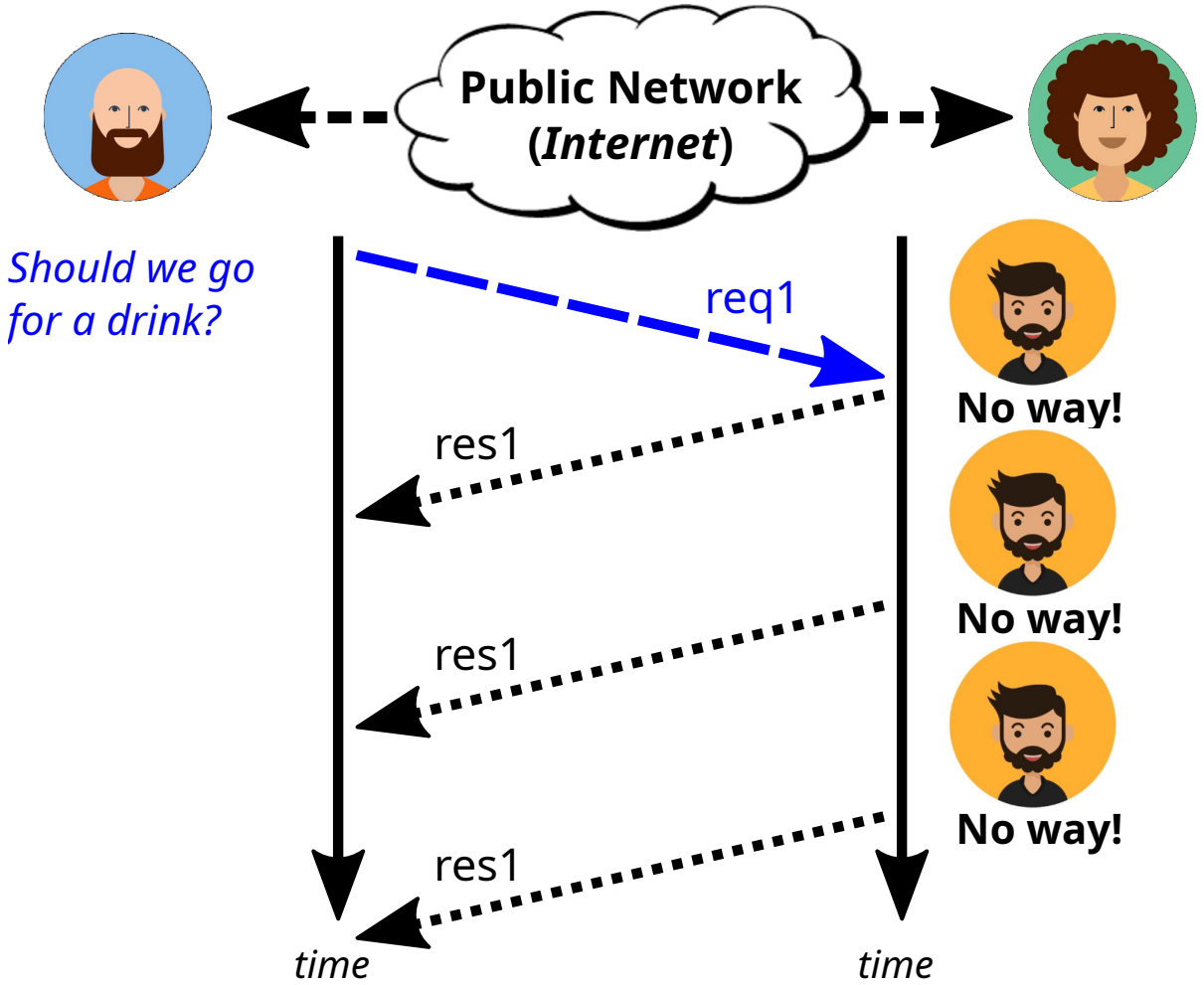
- Only two participants.
- Bi-directional communications.
- Request $\Longrightarrow$ response.

An Example of a (Network) Communication



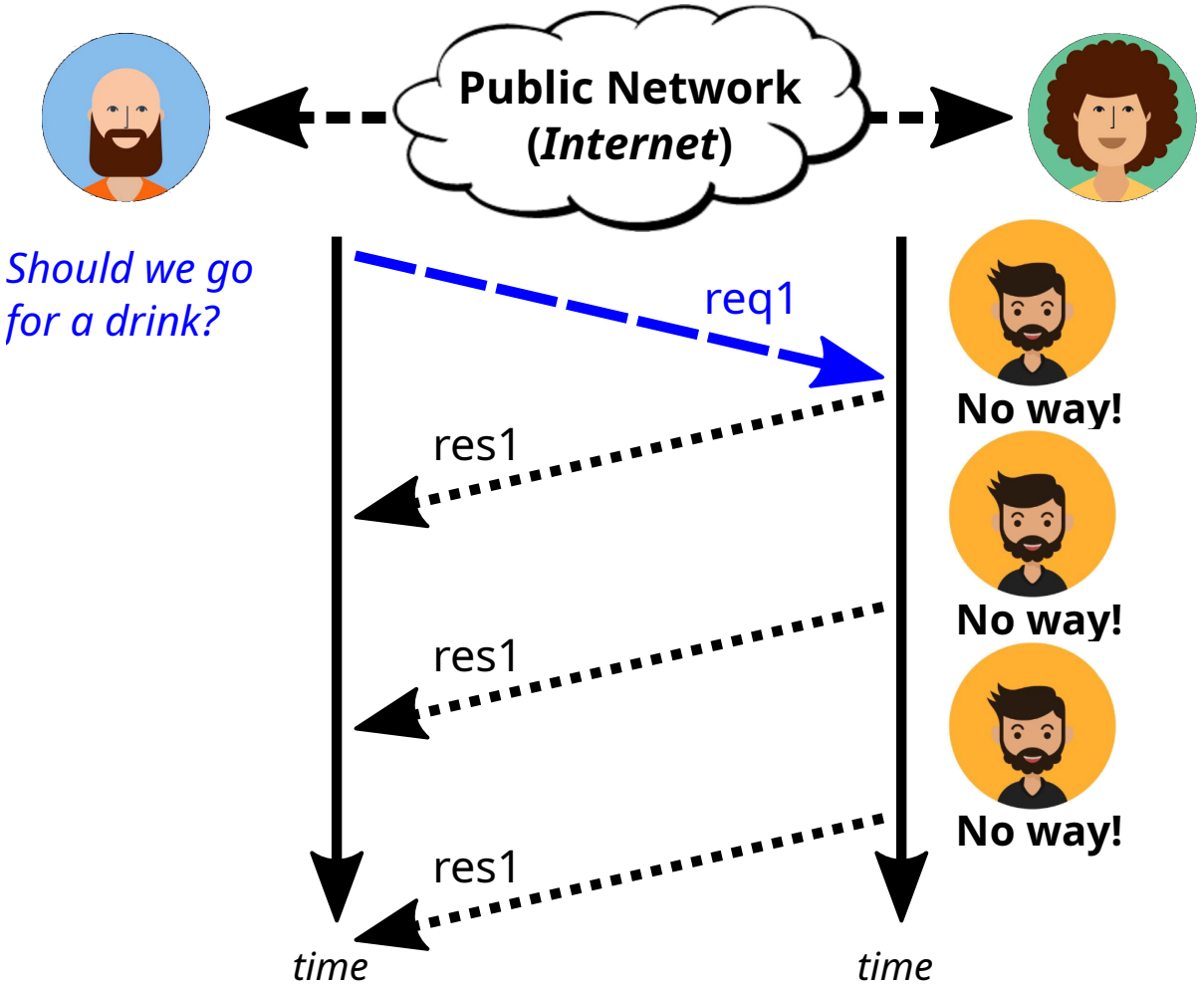
- Only two participants.
- Bi-directional communications.
- Request $\Longrightarrow$ response.

An Example of a (Network) Communication



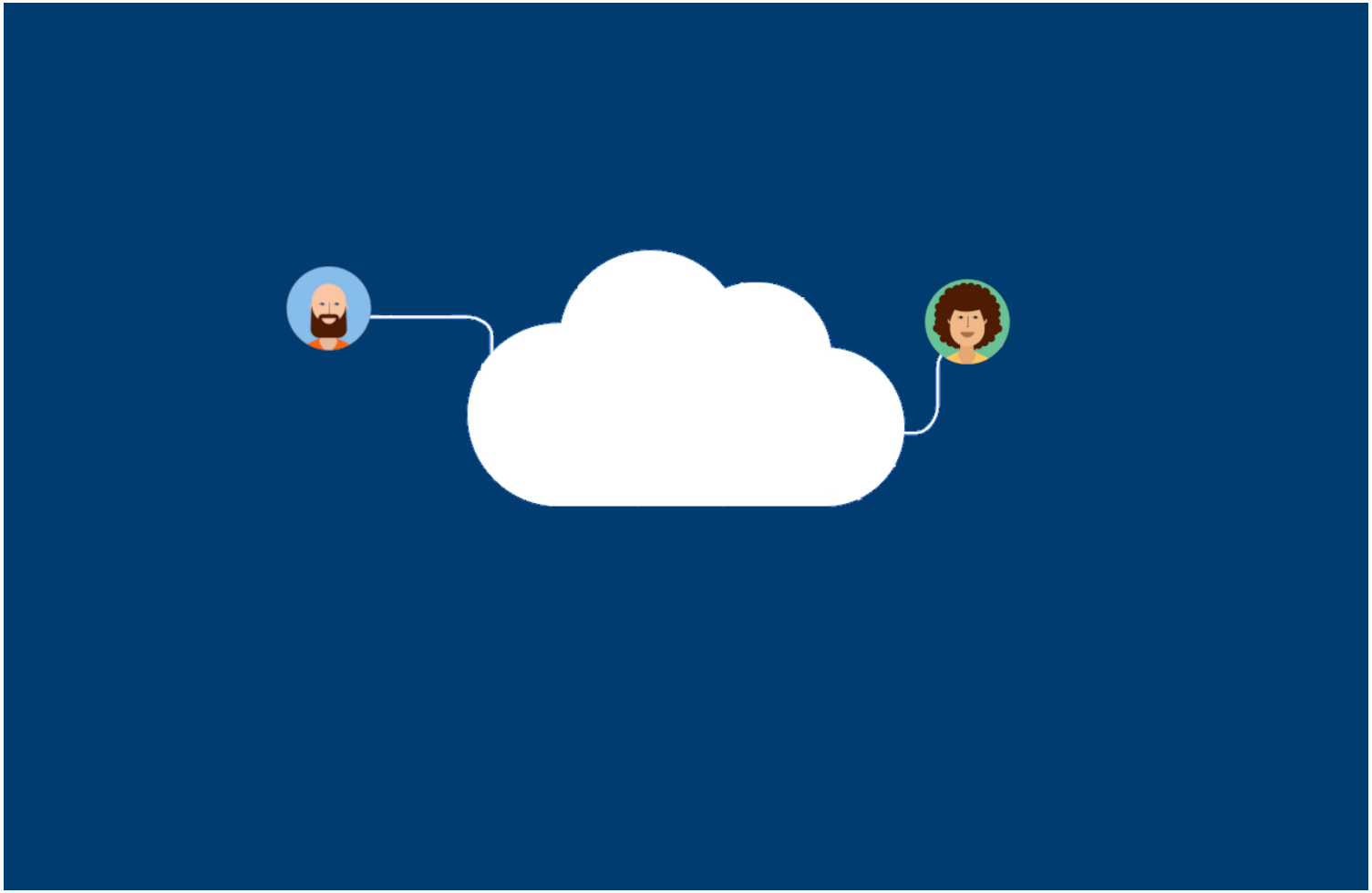
- Only two participants.
- Bi-directional communications.
- Request $\Longrightarrow$ response.

An Example of a (Network) Communication

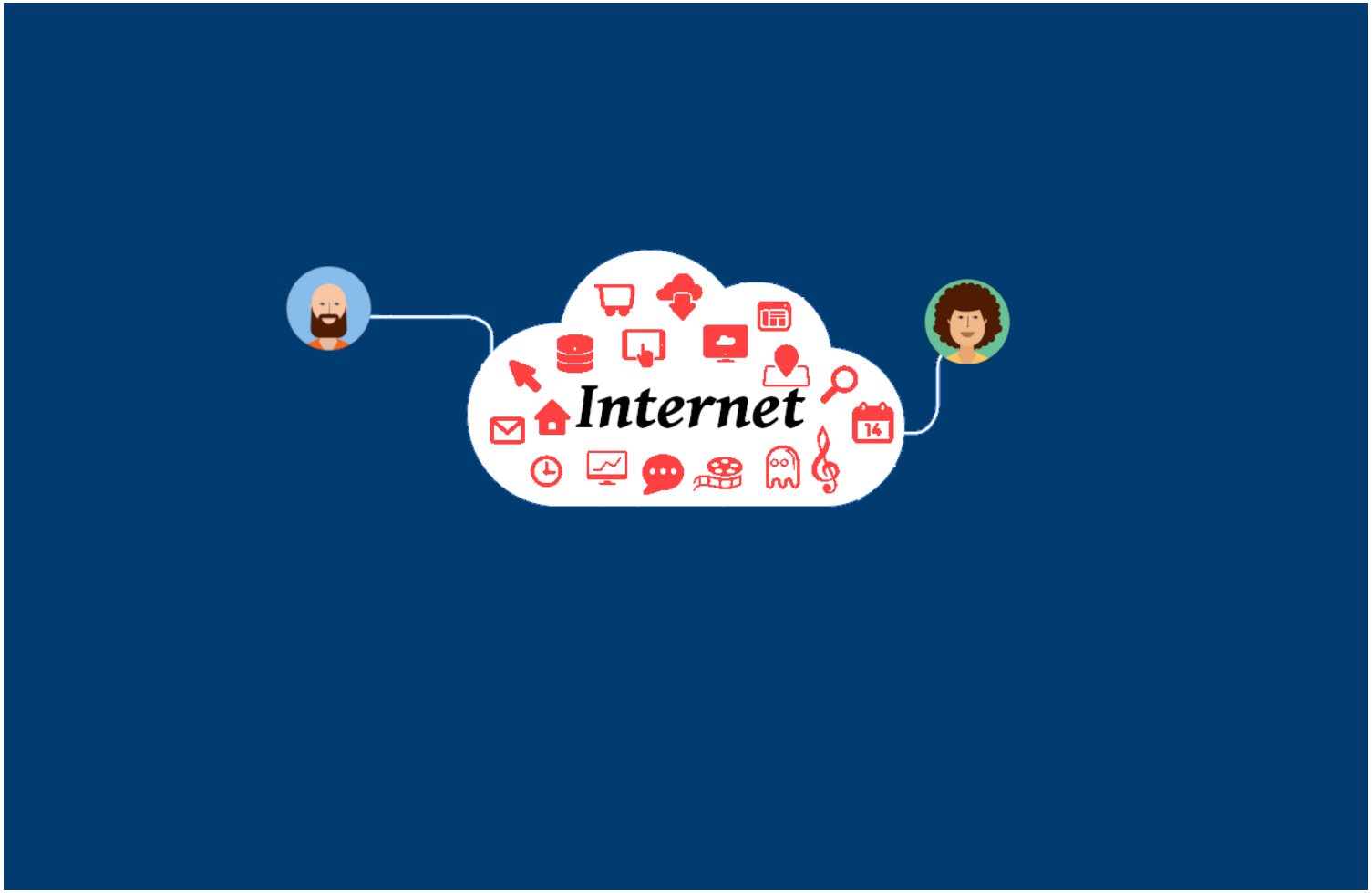


- Only two participants.
- Bi-directional communications.
- Request $\Longrightarrow$ response.
- *Confidentiality, integrity, authenticity.*

Communications over Internet

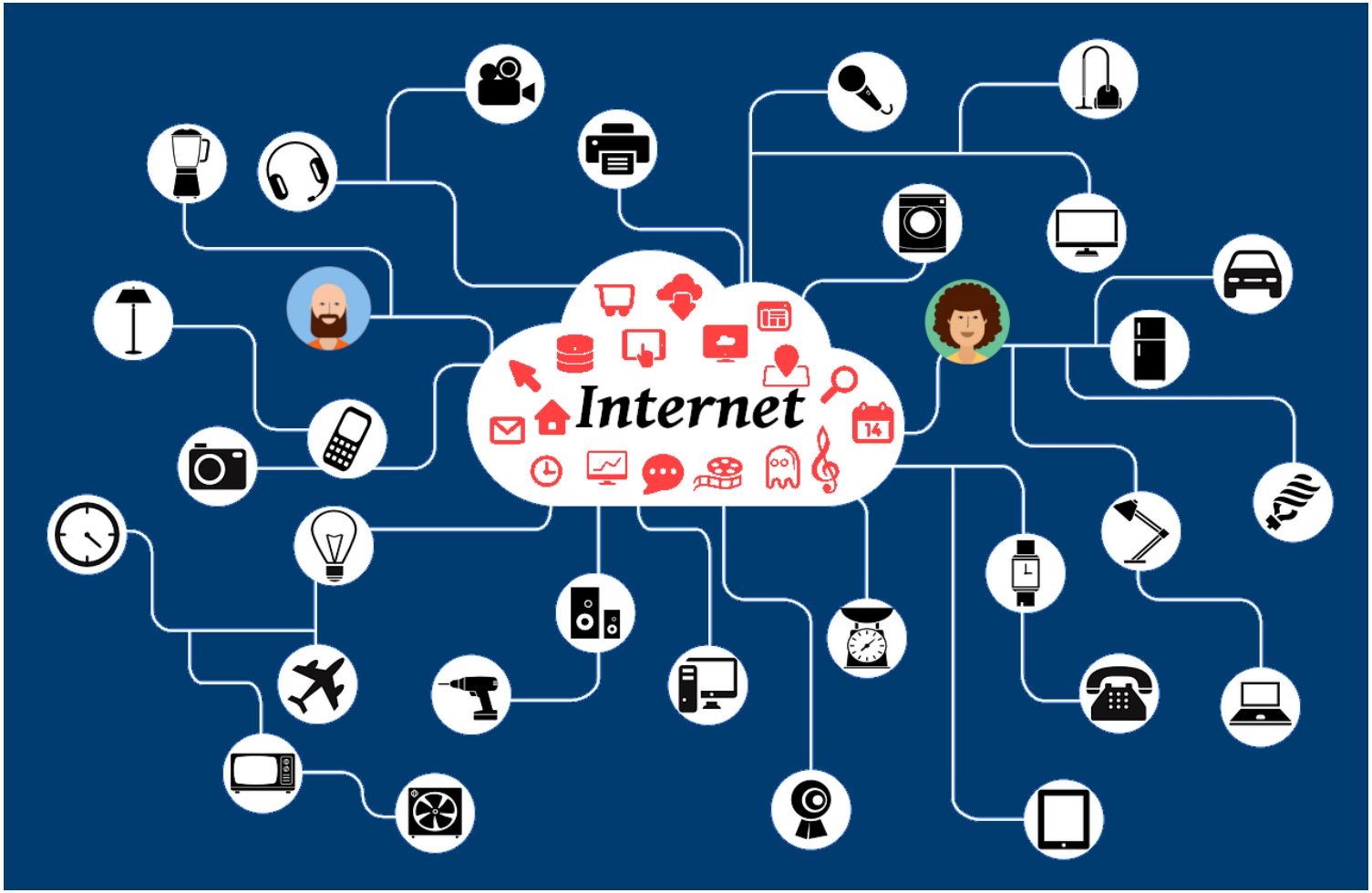


Communications over Internet



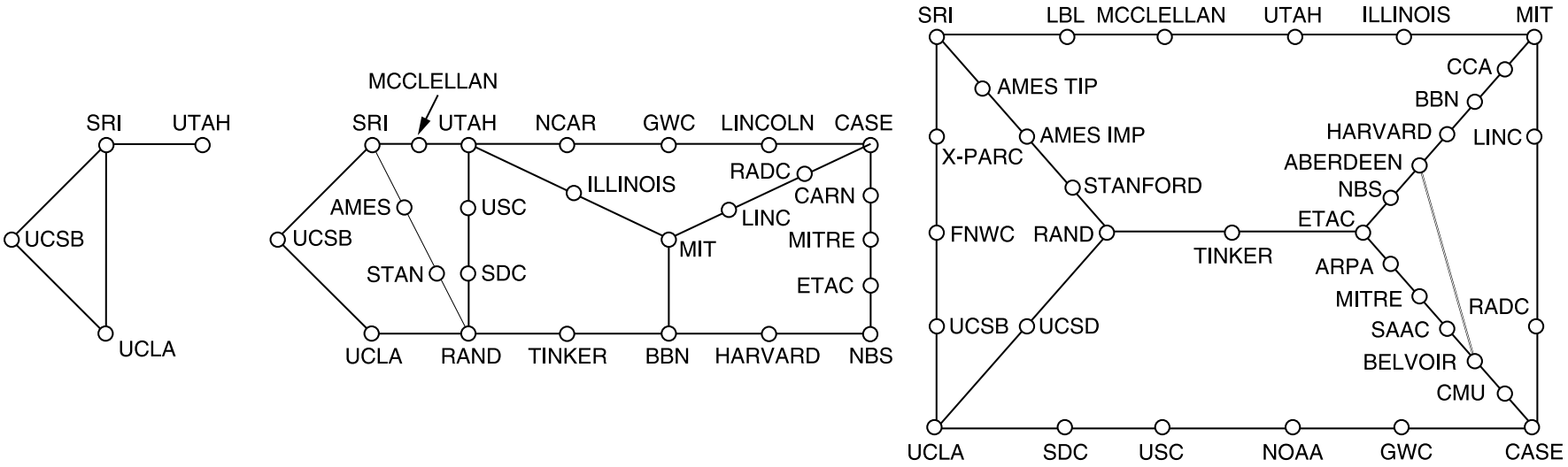
1. Internet: **<u>thousands</u> of protocols and services**.

Communications over Internet



1. Internet: **thousands of protocols and services**.
2. Users: **billions of devices**.

History of Internet



(a) (b) (c)

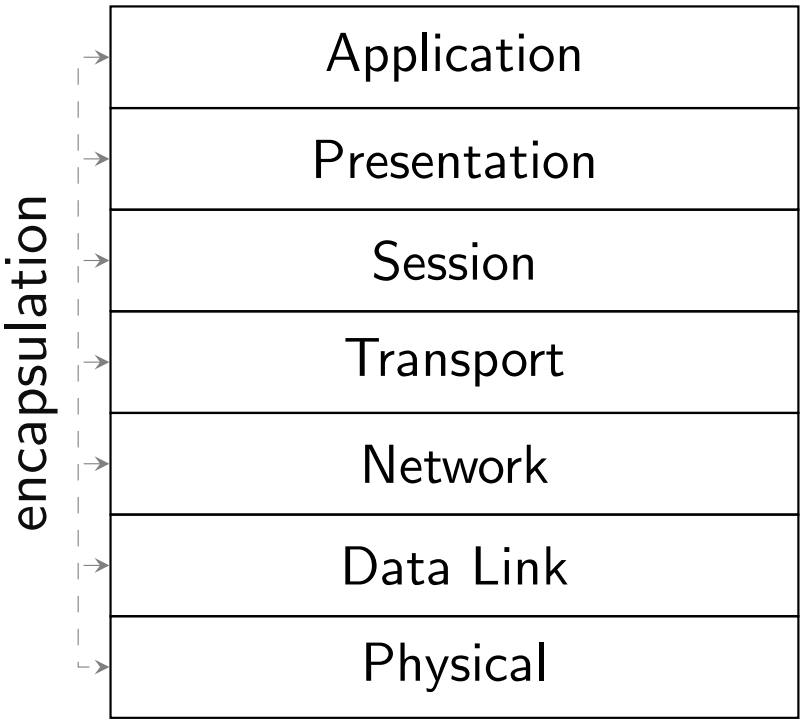
- ARPANET growth:
  (a) December 1969. (b) July 1970. (c) September 1972[1].
- Introduction of new protocols:
  - TCP/IP (1974)
  - DNS (1980s)

[1] Source: A. Tanenbaum, "Computer Networks", 5ed (2011)

ISO/OSI Model

The *ISO OSI (Open Systems Interconnection) Reference Model* was developed in 1983 to allow connecting open systems.

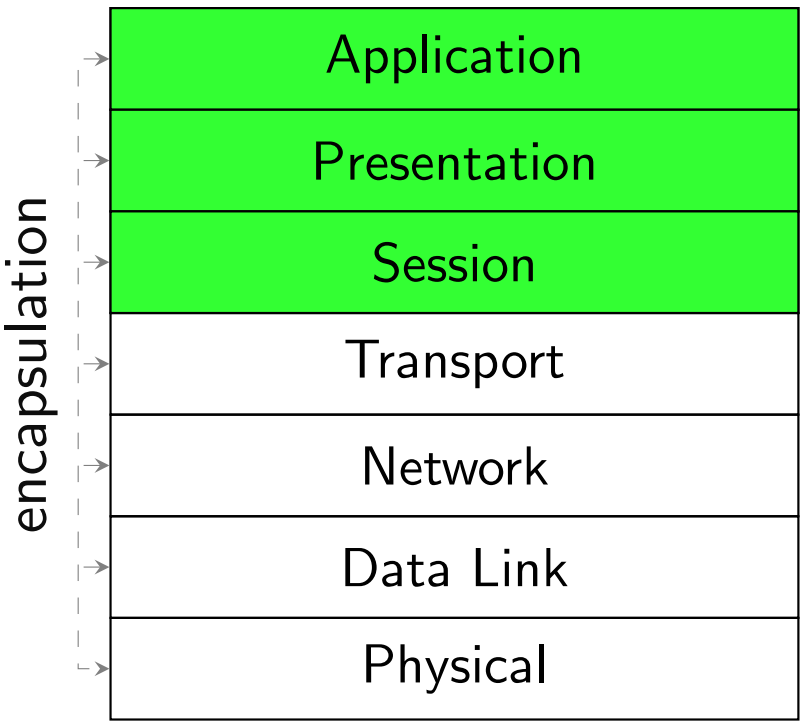
It contains seven layers, and each layer is defined as a separate international standard.



ISO/OSI Model

The *ISO OSI (Open Systems Interconnection) Reference Model* was developed in 1983 to allow connecting open systems.

It contains seven layers, and each layer is defined as a separate international standard.

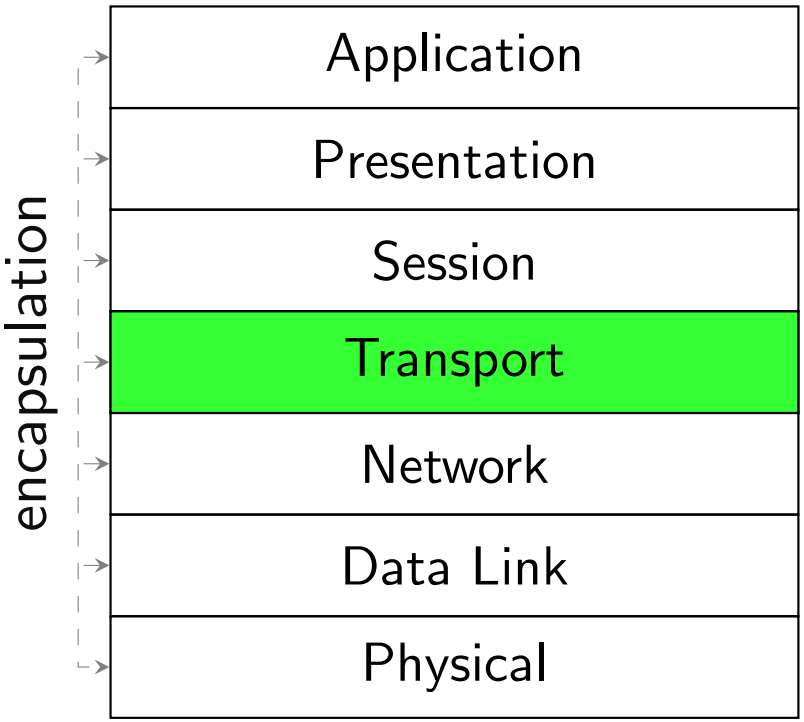


Contains protocols commonly needed by users, it manages and synchronizes the sessions among users on different machines (telnet, ftp, HTTP).

ISO/OSI Model

The *ISO OSI (Open Systems Interconnection) Reference Model* was developed in 1983 to allow connecting open systems.

It contains seven layers, and each layer is defined as a separate international standard.

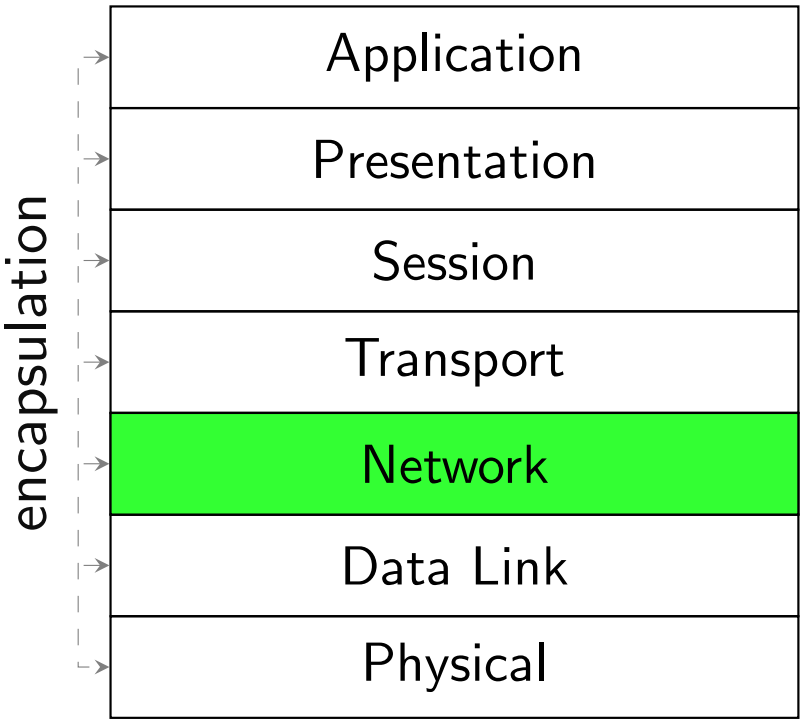


Determines the type of service that is offered to upper layers and acts as an interface among the *Network* and *higher layers* (TCP, UDP).

ISO/OSI Model

The *ISO OSI (Open Systems Interconnection) Reference Model* was developed in 1983 to allow connecting open systems.

It contains seven layers, and each layer is defined as a separate international standard.

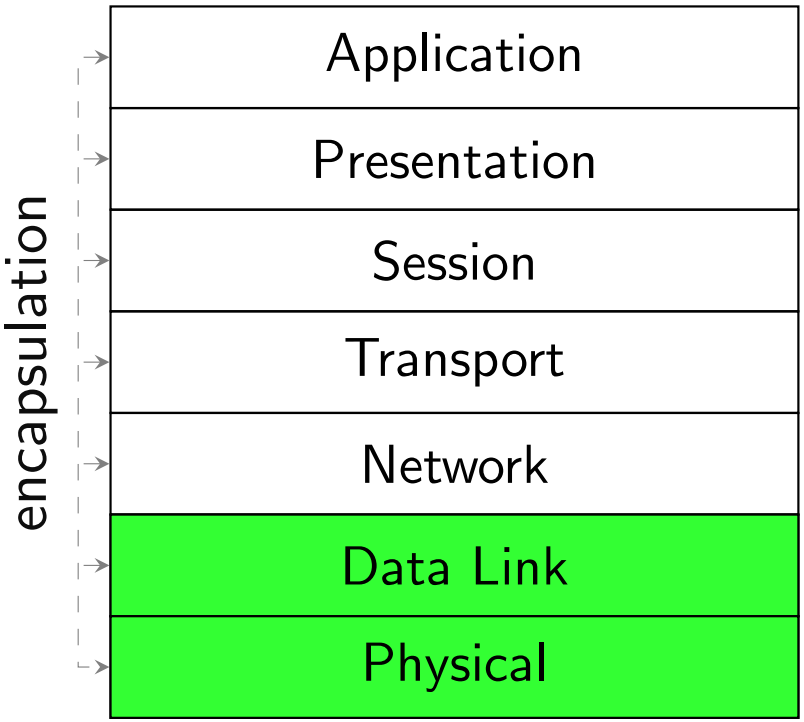


Manages how packets are routed from sources to destinations (IP, ICMP, ARP).

ISO/OSI Model

The *ISO OSI (Open Systems Interconnection) Reference Model* was developed in 1983 to allow connecting open systems.

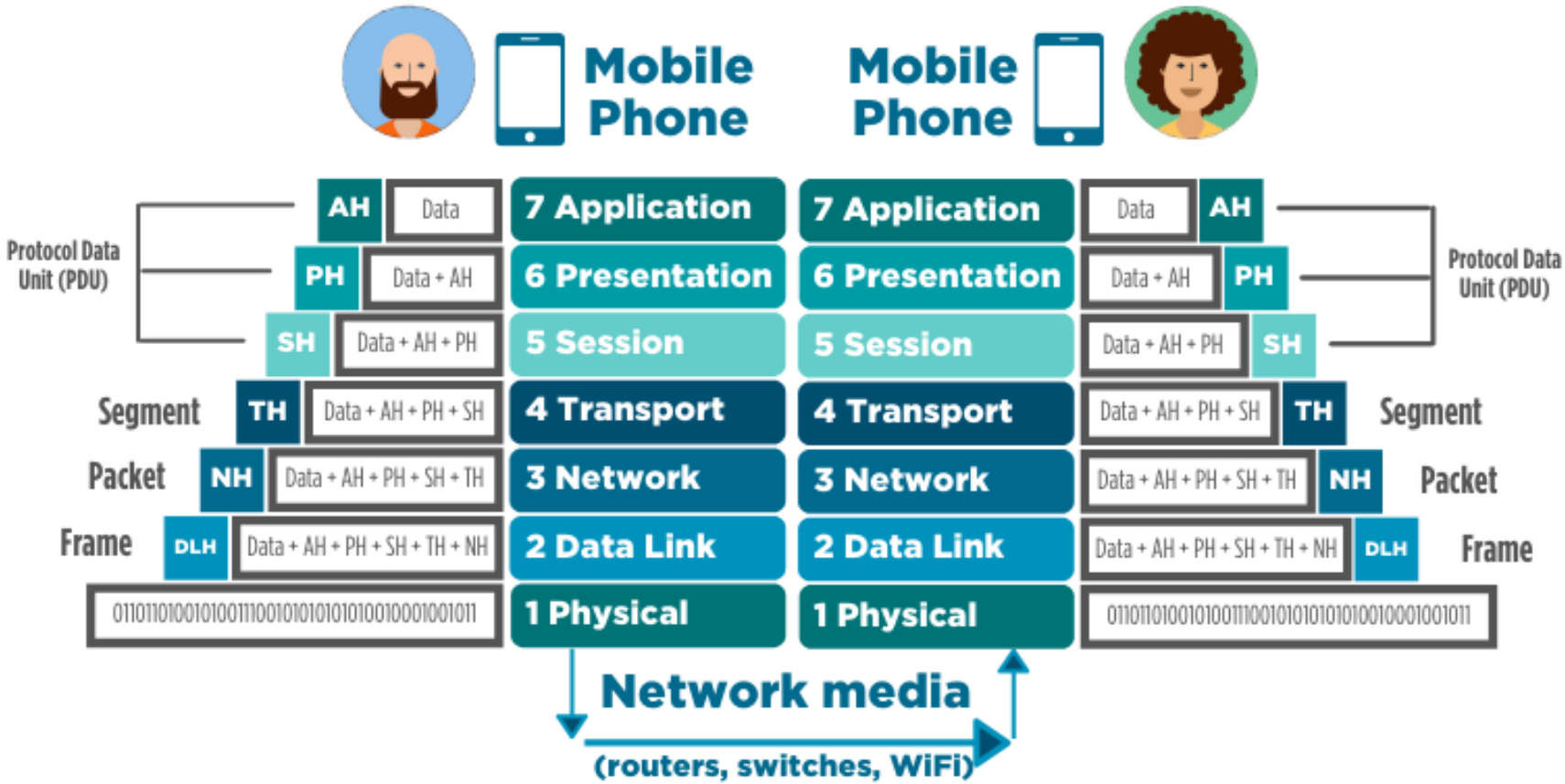
It contains seven layers, and each layer is defined as a separate international standard.



Handle the errors and the transmissions of raw bits over the physical communication channel.

Information Flow

How messages are exchanged in in the ISO/OSI model:



- ⇓ data **encapsulation**
- ⇑ data **de-capsulation**

Possible Source of Issues

1. No authentication $\Longrightarrow$ **spoofing, hijacking**

Possible Source of Issues

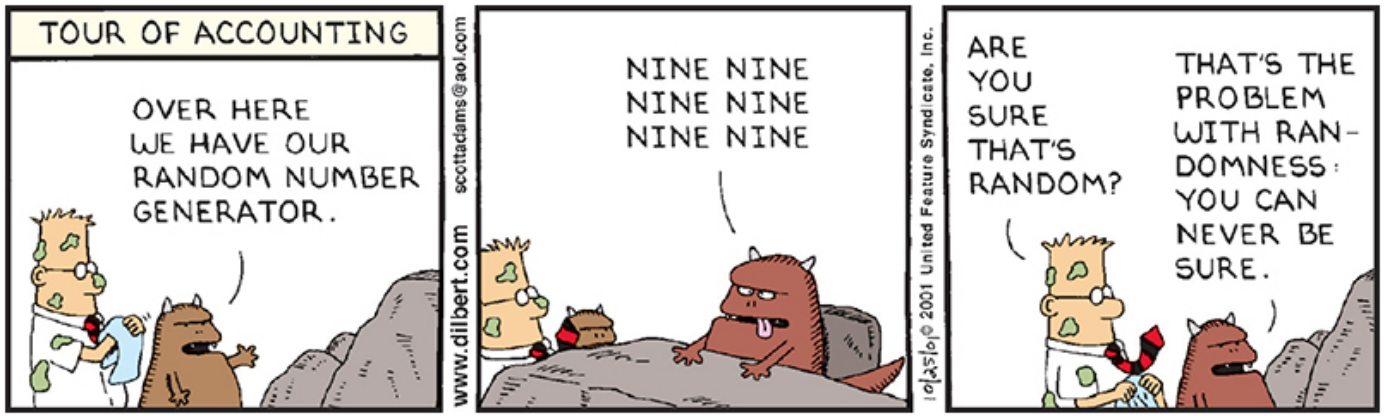
1. No authentication $\Longrightarrow$ **spoofing, hijacking**
2. No timestamps $\Longrightarrow$ **replay attacks**

Possible Source of Issues

1. No authentication $\Longrightarrow$ **spoofing, hijacking**
2. No timestamps $\Longrightarrow$ **replay attacks**
3. Insufficient checks $\Longrightarrow$ **poisoning**

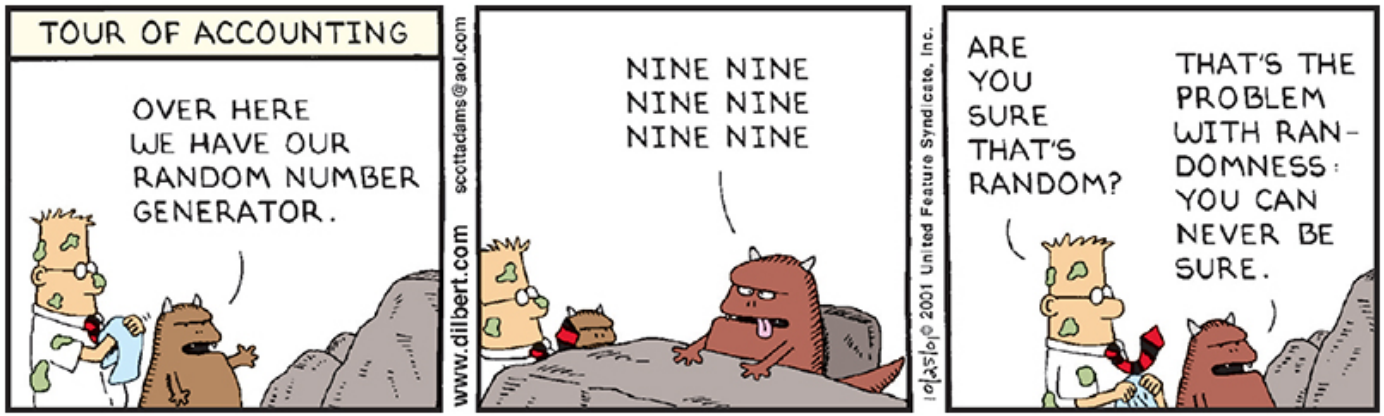
Possible Source of Issues

1. No authentication $\Longrightarrow$ **spoofing, hijacking**
2. No timestamps $\Longrightarrow$ **replay attacks**
3. Insufficient checks $\Longrightarrow$ **poisoning**
4. Poor implementations $\Longrightarrow$ **memory corruption, weak randomness**



Possible Source of Issues

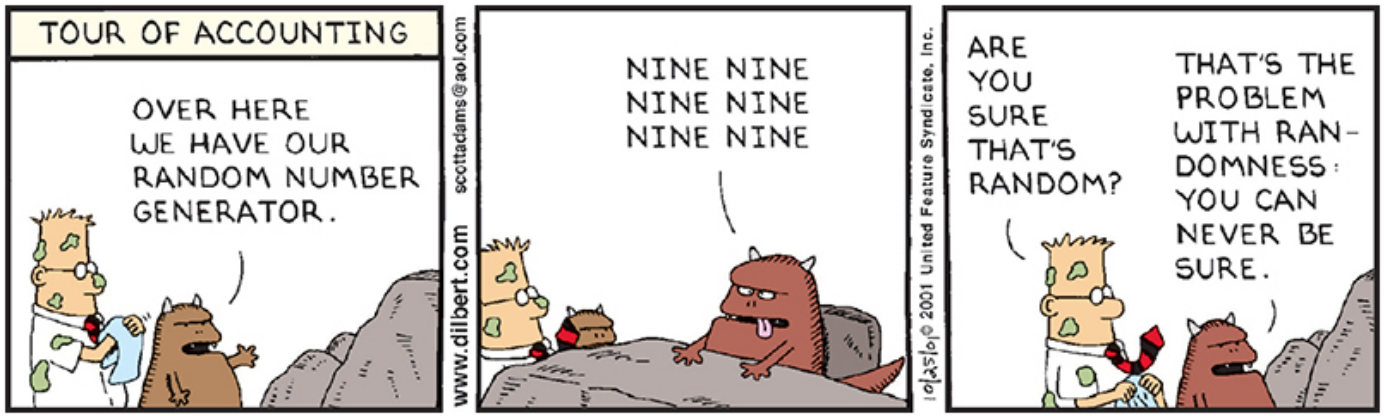
1. No authentication $\Longrightarrow$ **spoofing, hijacking**
2. No timestamps $\Longrightarrow$ **replay attacks**
3. Insufficient checks $\Longrightarrow$ **poisoning**
4. Poor implementations $\Longrightarrow$ **memory corruption, weak randomness**



Why do these issues still persist?

Possible Source of Issues

1. No authentication $\Longrightarrow$ **spoofing, hijacking**
2. No timestamps $\Longrightarrow$ **replay attacks**
3. Insufficient checks $\Longrightarrow$ **poisoning**
4. Poor implementations $\Longrightarrow$ **memory corruption, weak randomness**

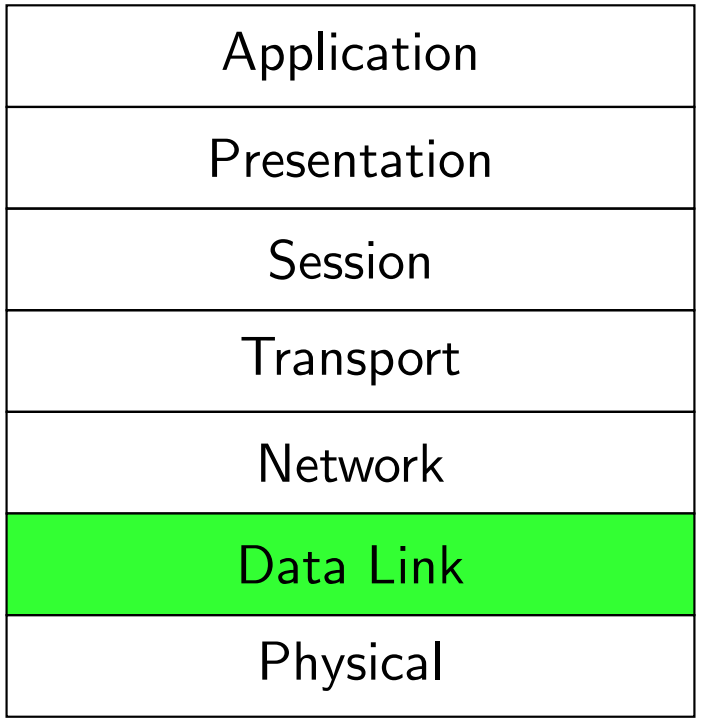


Why do these issues still persist?

Legacy requirements:

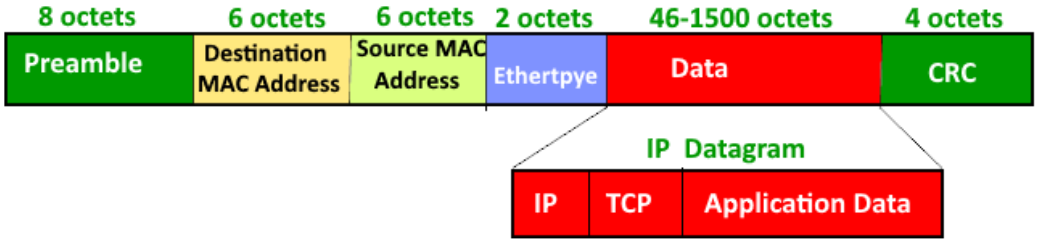
fixing issues might require to re-deploy **millions** of servers.

The Data Link Layer

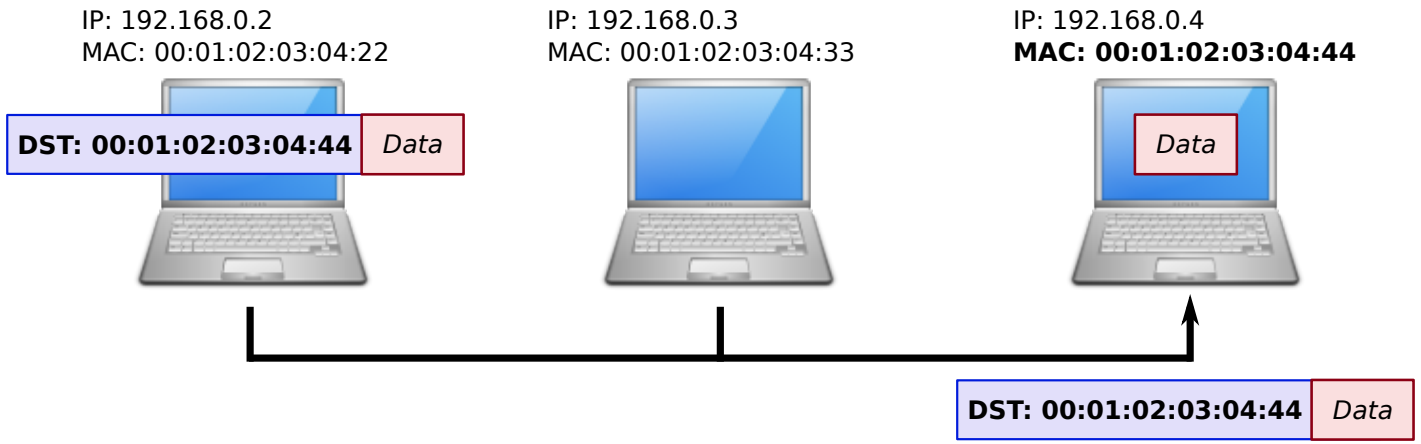


Local Network - Delivery

To reach machines in the *same network*, packets (e.g., IP datagrams) are encapsulated in *Data Link frames*. The Data Link layer implements basic functionalities such as *collision avoidance* and *error handling* (CRC, checksum). The most common type is the *Ethernet* frame.

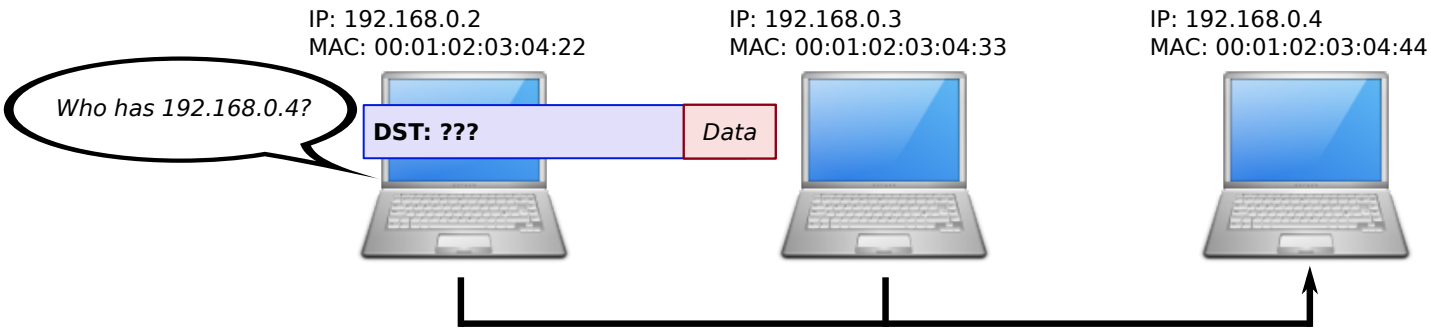


The frame includes the MAC address of the destination. Frames are **broadcasted** and the network cards pass to the operating system only those frames that *are indented for a particular host* (i.e., with the same MAC address of the host).



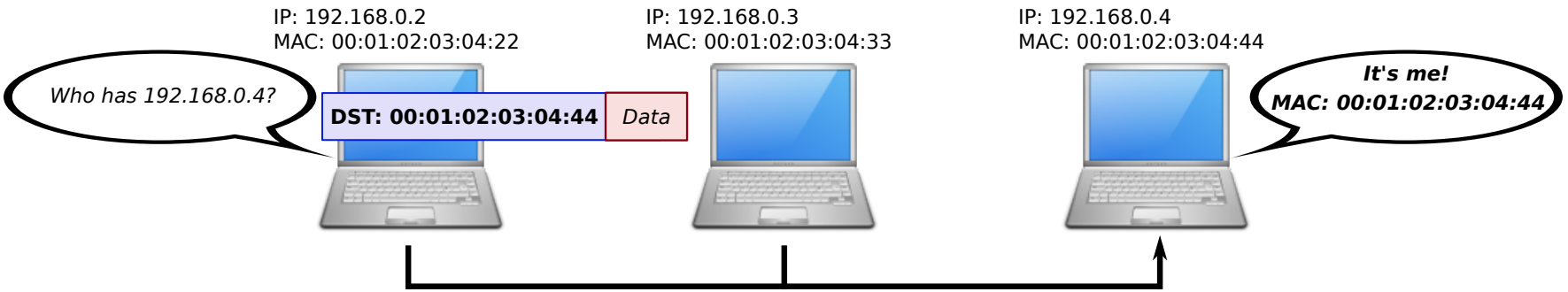
The Address Resolution Protocol (ARP)

Hosts leverage ARP to learn which MAC address is associated to a given IP.
The ARP response is cached when the host needs to contact the same machine.



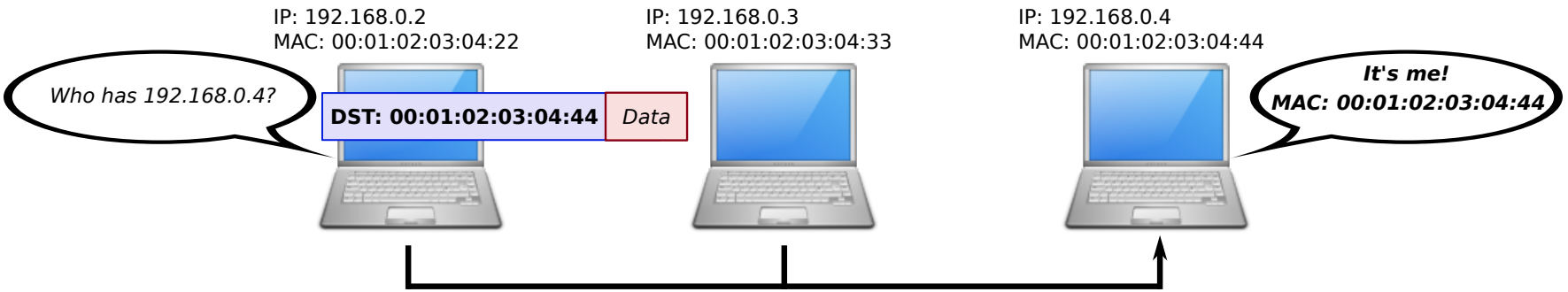
The Address Resolution Protocol (ARP)

Hosts leverage ARP to learn which MAC address is associated to a given IP.
The ARP response is cached when the host needs to contact the same machine.



The Address Resolution Protocol (ARP)

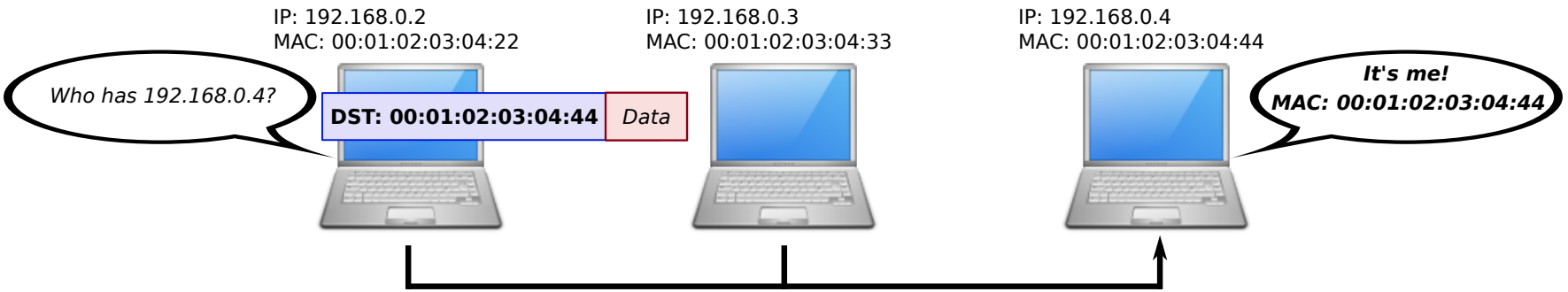
Hosts leverage ARP to learn which MAC address is associated to a given IP.
The ARP response is cached when the host needs to contact the same machine.



Can you spot any issue?

The Address Resolution Protocol (ARP)

Hosts leverage ARP to learn which MAC address is associated to a given IP.
The ARP response is cached when the host needs to contact the same machine.

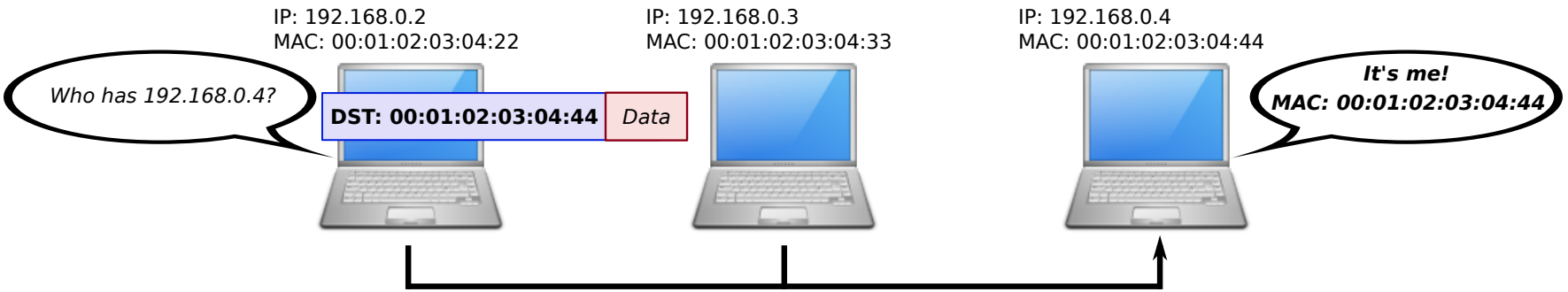


ARP does not provide authentication:

- attackers compete with legitimate hosts to provide a *fake MAC/IP mapping*
  $\Rightarrow$ the traffic gets redirected to the attacker

The Address Resolution Protocol (ARP)

Hosts leverage ARP to learn which MAC address is associated to a given IP.
The ARP response is cached when the host needs to contact the same machine.

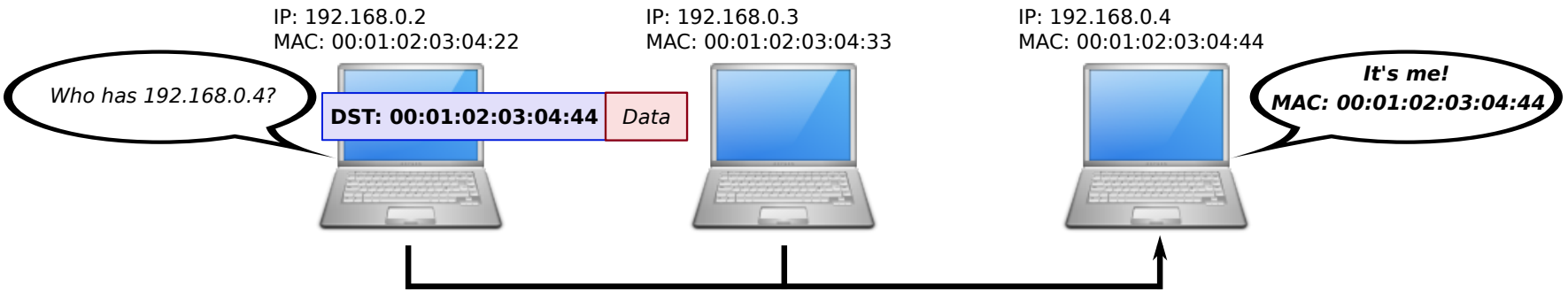


ARP does not provide authentication:

- attackers compete with legitimate hosts to provide a *fake MAC/IP mapping*
  $\Rightarrow$ the traffic gets redirected to the attacker
- legitimate ARP replies will restore the IP/MAC mapping

The Address Resolution Protocol (ARP)

Hosts leverage ARP to learn which MAC address is associated to a given IP.
The ARP response is cached when the host needs to contact the same machine.

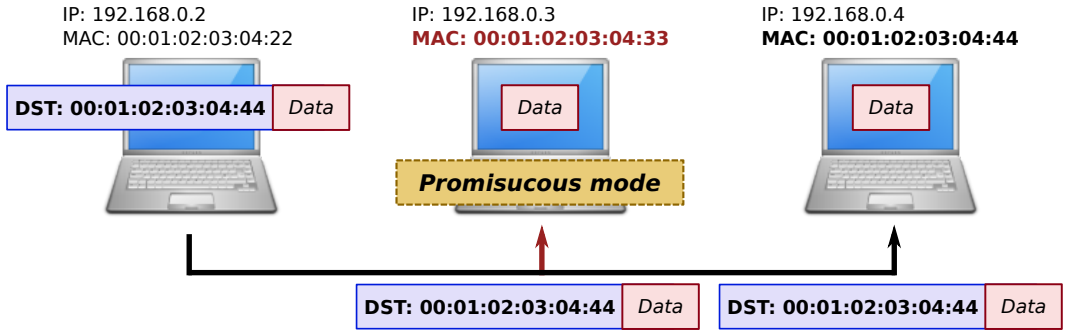


ARP does not provide authentication:

- attackers compete with legitimate hosts to provide a *fake MAC/IP mapping*
  $\Rightarrow$ the traffic gets redirected to the attacker
- legitimate ARP replies will restore the IP/MAC mapping
  - solution: the attacker *continuously sends* spoofed ARP replies

Local Network - Sniffing

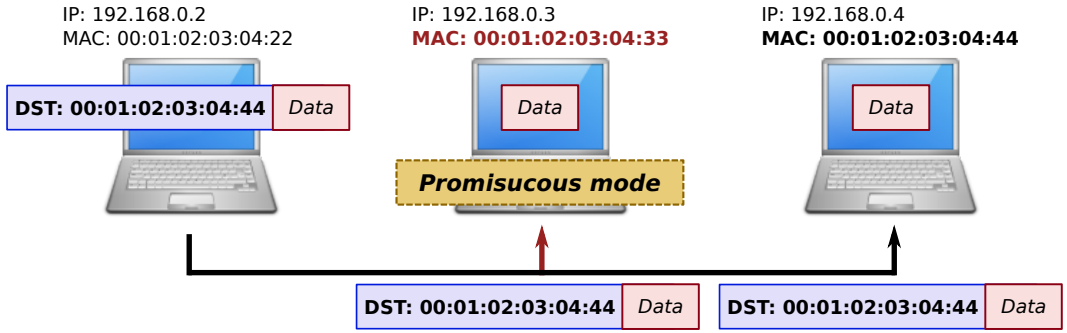
All the hosts on the local network "see" the traffic, but they discard it. If a host sets his network card in *promiscuous mode*, it will gain access to all the traffic. $\Longrightarrow$ the host becomes a **network sniffer**.



Additional information: `https://www.just.edu.jo/~tawalbeh/nyit/incs745/presentations/Sniffers.pdf`

Local Network - Sniffing

All the hosts on the local network "see" the traffic, but they discard it. If a host sets his network card in *promiscuous mode*, it will gain access to all the traffic. $\Longrightarrow$ the host becomes a **network sniffer**.



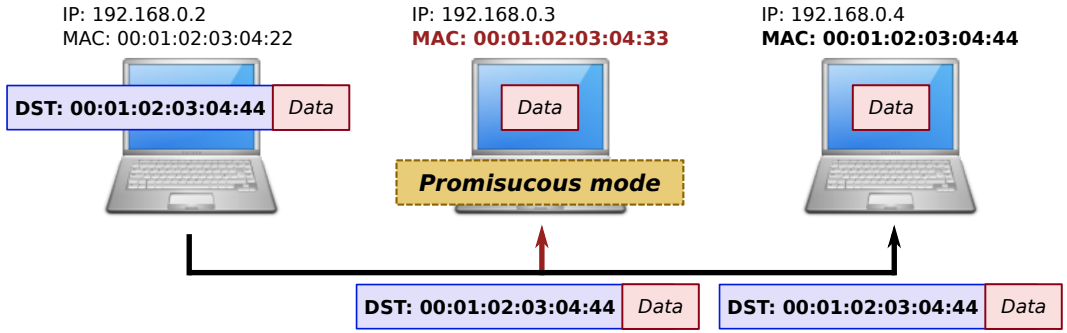
What is the issue?

Unencrypted traffic (e.g, passwords, usernames, files content)

Additional information: `https://www.just.edu.jo/~tawalbeh/nyit/incs745/presentations/Sniffers.pdf`

Local Network - Sniffing

All the hosts on the local network "see" the traffic, but they discard it. If a host sets his network card in *promiscuous mode*, it will gain access to all the traffic. $\Longrightarrow$ the host becomes a **network sniffer**.



What is the issue?

Unencrypted traffic (e.g, passwords, usernames, files content)

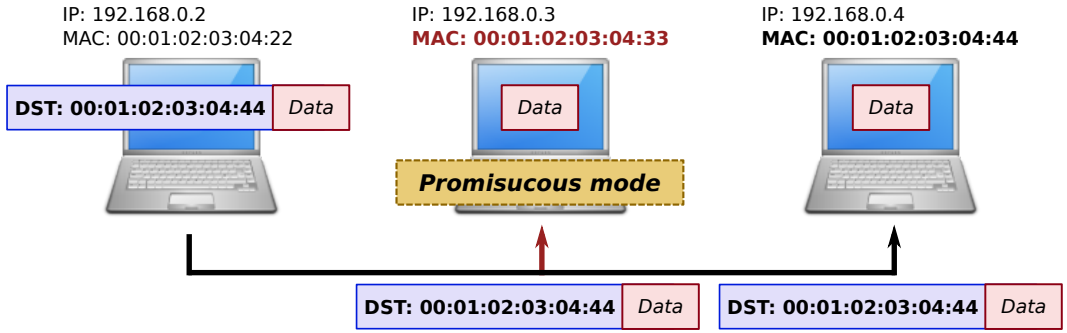
Sniffer detection:

- **ping**: send a frame with the correct IP and incorrect MAC $\rightarrow$ only sniffers will reply;

Additional information: `https://www.just.edu.jo/~tawalbeh/nyit/incs745/presentations/Sniffers.pdf`

Local Network - Sniffing

All the hosts on the local network "see" the traffic, but they discard it. If a host sets his network card in *promiscuous mode*, it will gain access to all the traffic. $\Longrightarrow$ the host becomes a **network sniffer**.



What is the issue?

Unencrypted traffic (e.g, passwords, usernames, files content)

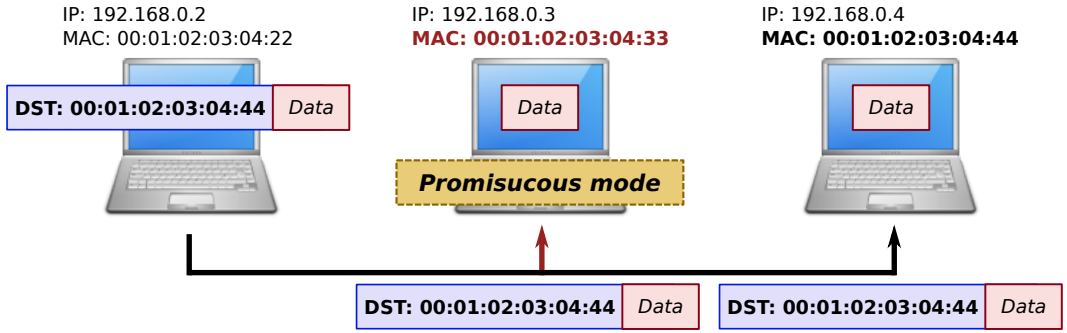
Sniffer detection:

- **ping**: send a frame with the correct IP and incorrect MAC $\rightarrow$ only sniffers will reply;
- **latency**: send huge amount of data, and check latency on the candidate sniffer *before* and *during* the data flooding;

Additional information: `https://www.just.edu.jo/~tawalbeh/nyit/incs745/presentations/Sniffers.pdf`

Local Network - Sniffing

All the hosts on the local network "see" the traffic, but they discard it. If a host sets his network card in *promiscuous mode*, it will gain access to all the traffic. $\Longrightarrow$ the host becomes a **network sniffer**.



What is the issue?

Unencrypted traffic (e.g, passwords, usernames, files content)

Sniffer detection:

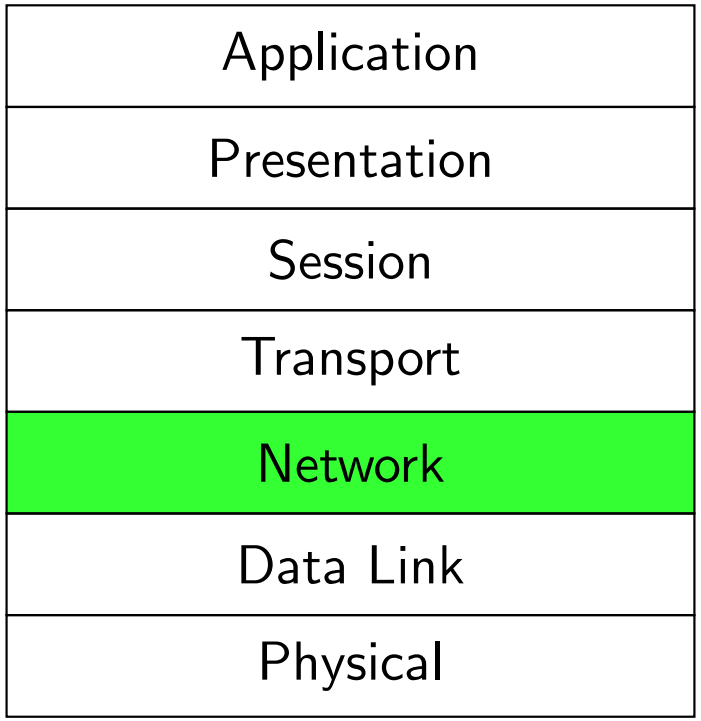
- **ping**: send a frame with the correct IP and incorrect MAC $\rightarrow$ only sniffers will reply;
- **latency**: send huge amount of data, and check latency on the candidate sniffer *before* and *during* the data flooding;

Sniffer prevention:

- use **switched Ethernet** instead of a hub: maps MAC(s) $\leftrightarrow$ physical port(s) and forwards the frames to the right host.
  - can be bypassed with MAC flooding

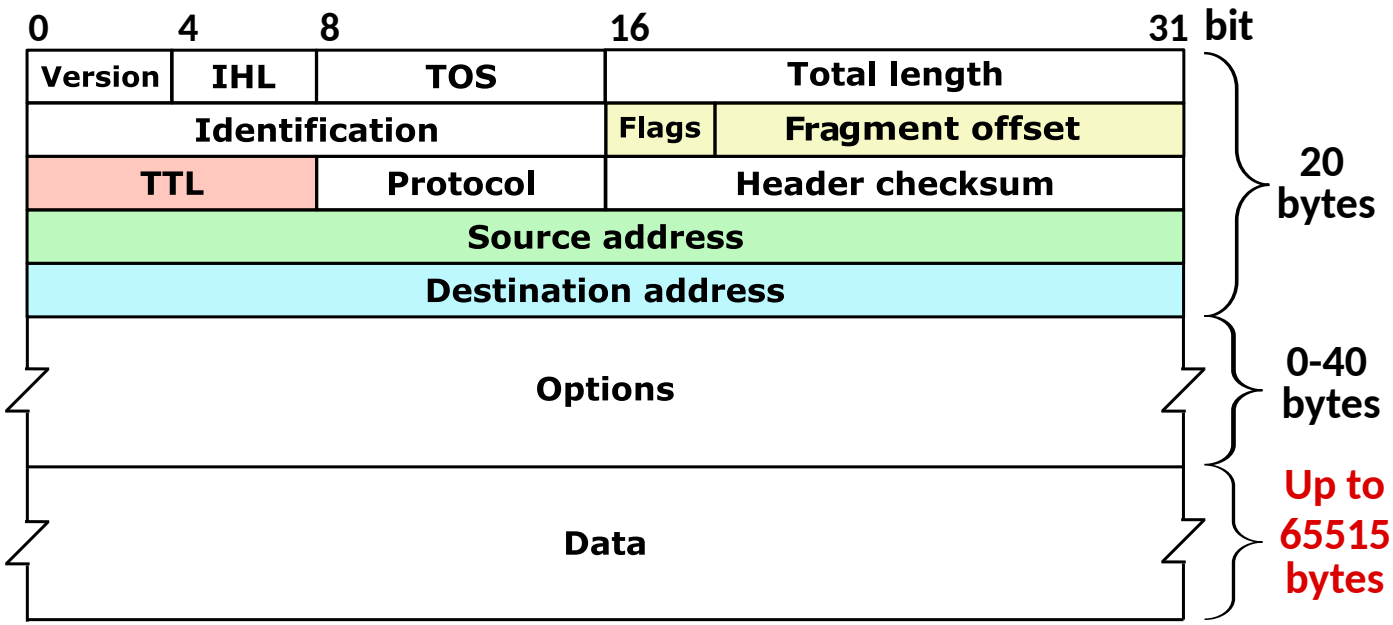
Additional information: `https://www.just.edu.jo/~tawalbeh/nyit/incs745/presentations/Sniffers.pdf`

The Network Layer



IPv4 Datagram

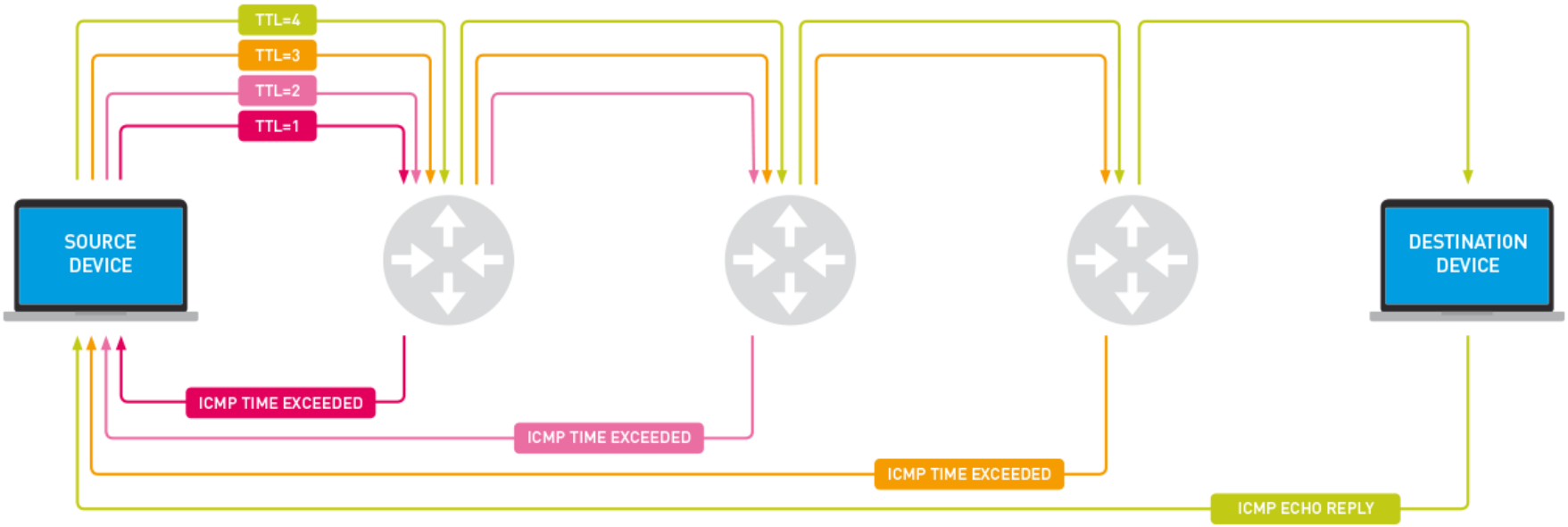
Encapsulates data from higher layers - best effort, no reliability guaranteed.



- **Source/Destination addresses**: where the packet originates/goes to; no authentication provided.
- **Time To Live (TTL)**: decreased at each router; ensures that datagrams do not roam forever.
- **Flags + Fragment Offset**: datagrams can be fragmented if the *lower transmission layers* can handle only packets of a certain size. These fields are used to reassemble the datagram on the receiving end.

ICMP and Traceroute

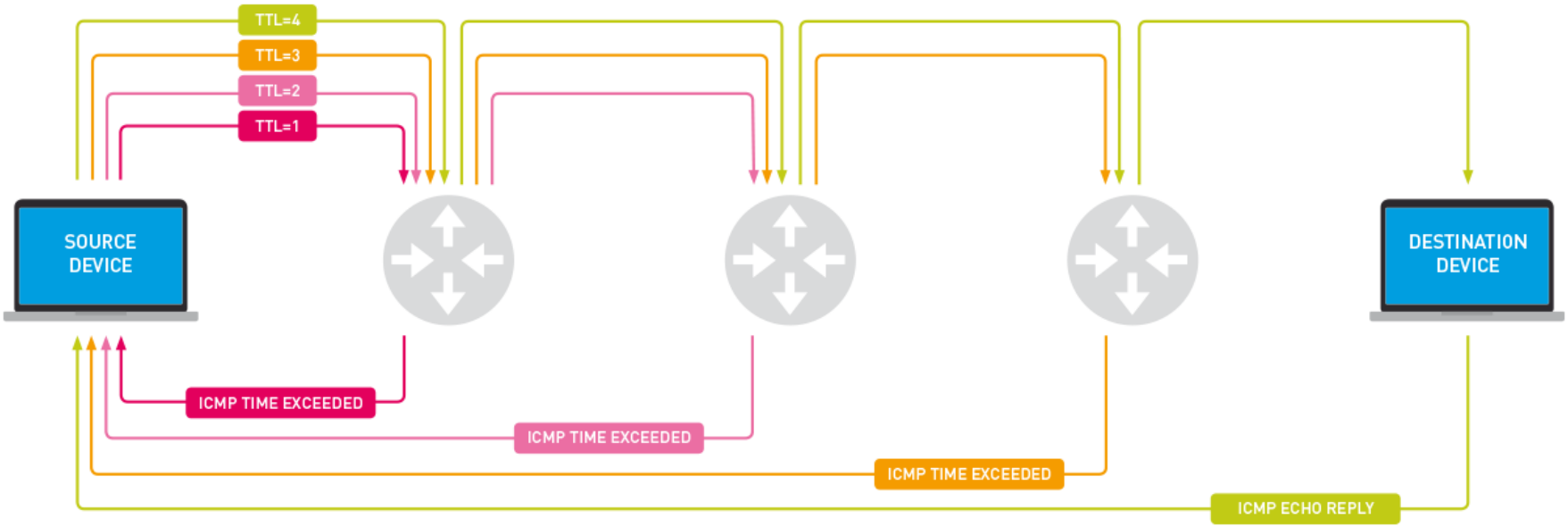
ICMP (Internet Control Message Protocol) is used to *test connectivity*.
About a dozen types of ICMP messages are defined, and each ICMP message type is carried encapsulated in an IP packet.



1. Routers *decrement* the TTL by one every time they route a datagram.

ICMP and Traceroute

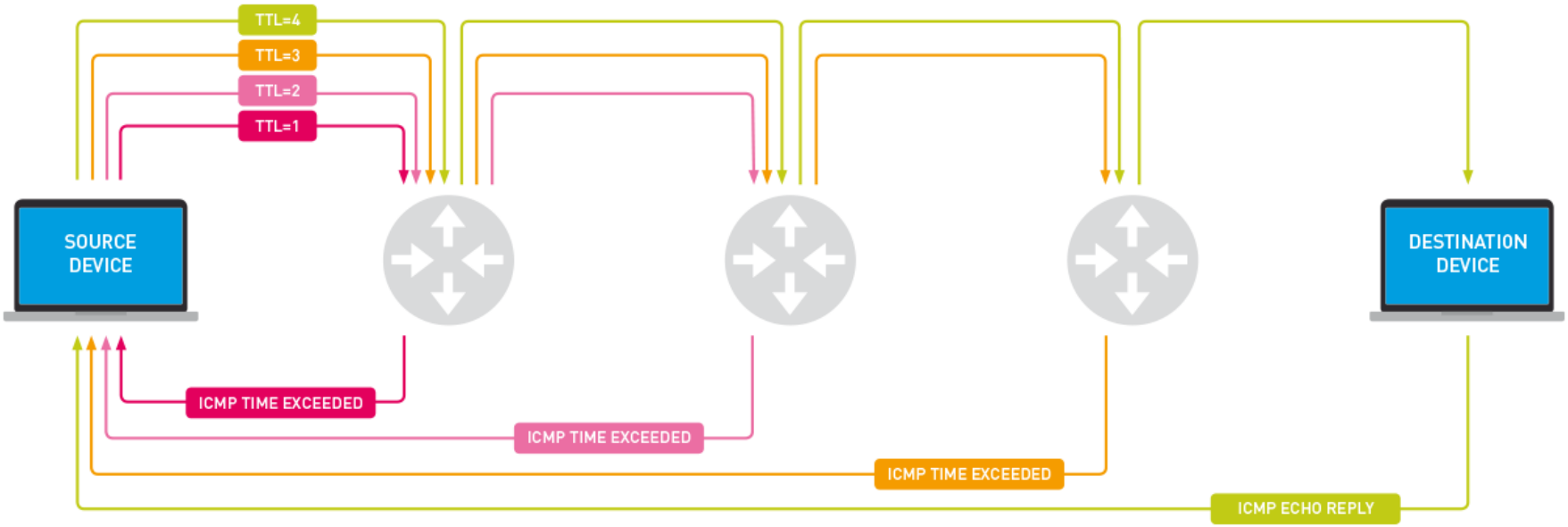
ICMP (Internet Control Message Protocol) is used to *test connectivity*.
About a dozen types of ICMP messages are defined, and each ICMP message type is carried encapsulated in an IP packet.



1. Routers *decrement* the TTL by one every time they route a datagram.
2. When TTL is 0, the datagram is discarded and the router sends a *"ICMP Time Exceeded"* message to the sender.

ICMP and Traceroute

ICMP (Internet Control Message Protocol) is used to *test connectivity*.
About a dozen types of ICMP messages are defined, and each ICMP message type is carried encapsulated in an IP packet.



1. Routers *decrement* the TTL by one every time they route a datagram.
2. When TTL is 0, the datagram is discarded and the router sends a *"ICMP Time Exceeded"* message to the sender.
3. A host can enumerate the routers on a path by sending datagrams with increasing TTL ($\rightarrow$ *Traceroute*).

Ping of Death (1997)

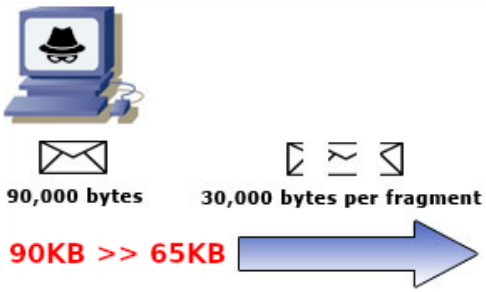
1. The attacker wants to send a datagram larger than 65KB, but this would violate the TCP/IP rules;

90,000 bytes

90KB >> 65KB

[2] https://en.wikipedia.org/wiki/Ping_of_death
[3] https://www.cve.org/CVERecord?id=CVE-2013-3183
[4] https://www.cve.org/CVERecord?id=CVE-2020-16898

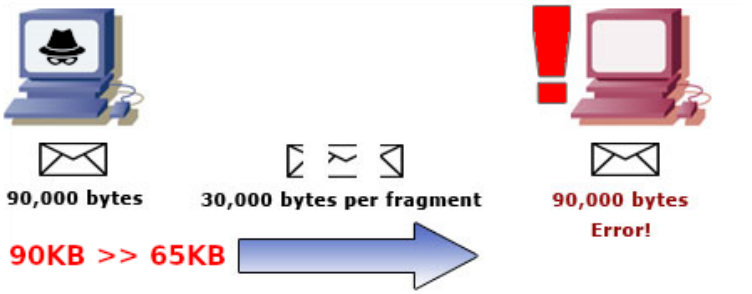
Ping of Death (1997)



1. The attacker wants to send a datagram larger than 65KB, but this would violate the TCP/IP rules;
2. The attacker splits the datagram into *multiple fragments*;

[2] https://en.wikipedia.org/wiki/Ping_of_death
[3] https://www.cve.org/CVERecord?id=CVE-2013-3183
[4] https://www.cve.org/CVERecord?id=CVE-2020-16898

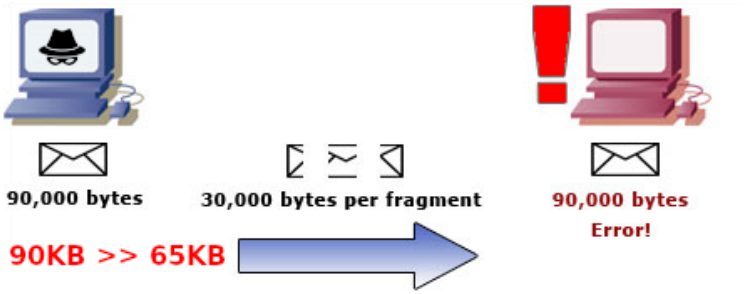
Ping of Death (1997)



1. The attacker wants to send a datagram larger than 65KB, but this would violate the TCP/IP rules;
2. The attacker splits the datagram into *multiple fragments*;
3. The total length is not known a priori and the receiver needs to process all the fragments before it can reassemble them. Once reassembled at the destination, the datagram will exceed the maximum allowed size, triggering a buffer overflow!

[2] https://en.wikipedia.org/wiki/Ping_of_death
[3] https://www.cve.org/CVERecord?id=CVE-2013-3183
[4] https://www.cve.org/CVERecord?id=CVE-2020-16898

Ping of Death (1997)



1. The attacker wants to send a datagram larger than 65KB, but this would violate the TCP/IP rules;
2. The attacker splits the datagram into *multiple fragments*;
3. The total length is not known a priori and the receiver needs to process all the fragments before it can reassemble them. Once reassembled at the destination, the datagram will exceed the maximum allowed size, triggering a buffer overflow!

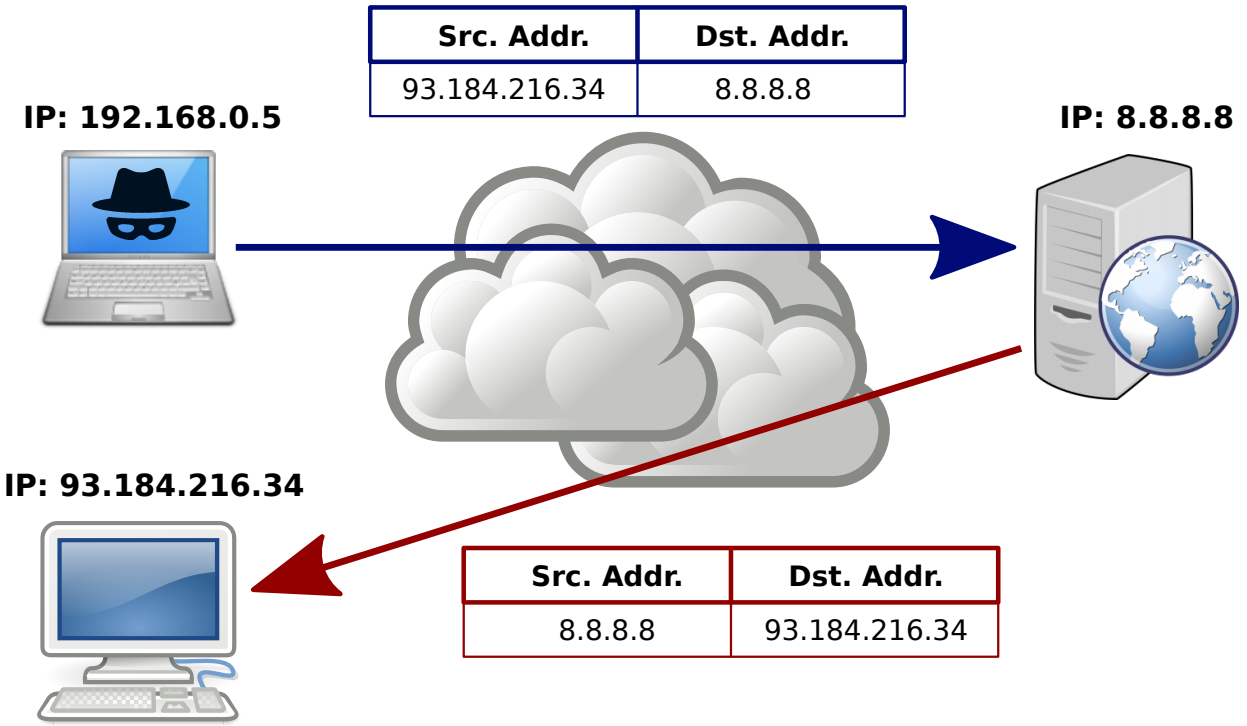
A few decades ago, this vulnerability affected the majority of operating systems, but nowadays is patched (*almost*[2,3,4]) everywhere...

Good example of how sometimes standards $\neq$ implementation

[2] https://en.wikipedia.org/wiki/Ping_of_death
[3] https://www.cve.org/CVERecord?id=CVE-2013-3183
[4] https://www.cve.org/CVERecord?id=CVE-2020-16898

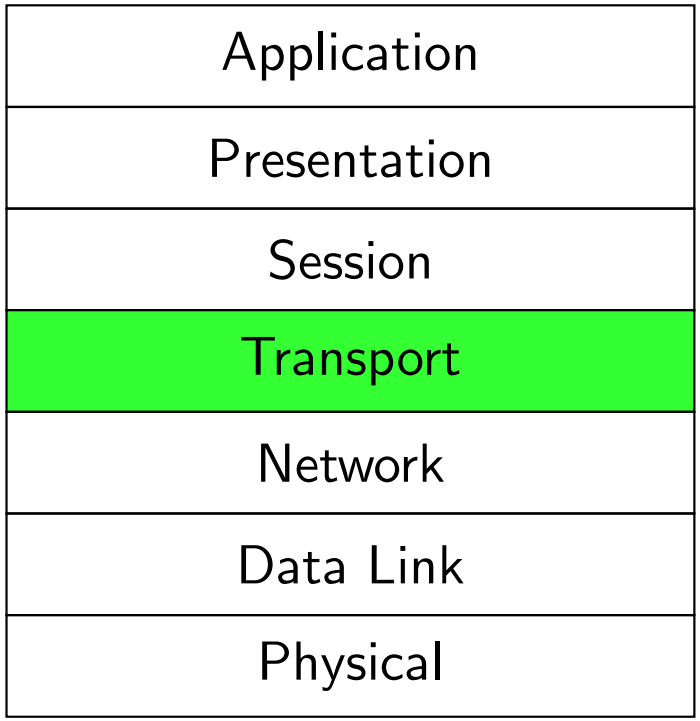
IP Spoofing

1. The attacker sends a message from a *different IP* from the one that he controls.
2. The recipient of the message will reply to the *spoofed IP* address.



- In the local network: IP spoofing == ARP spoofing.
- IP spoofing is commonly used in **Denial of Service (DoS) attacks** → we will see more about this in the next slides...

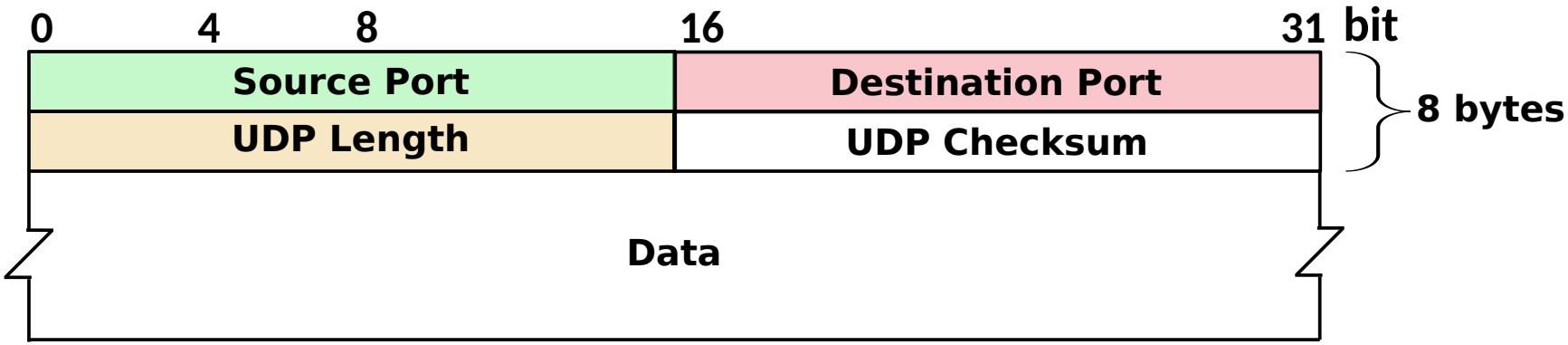
The Transport Layer



UDP Datagram

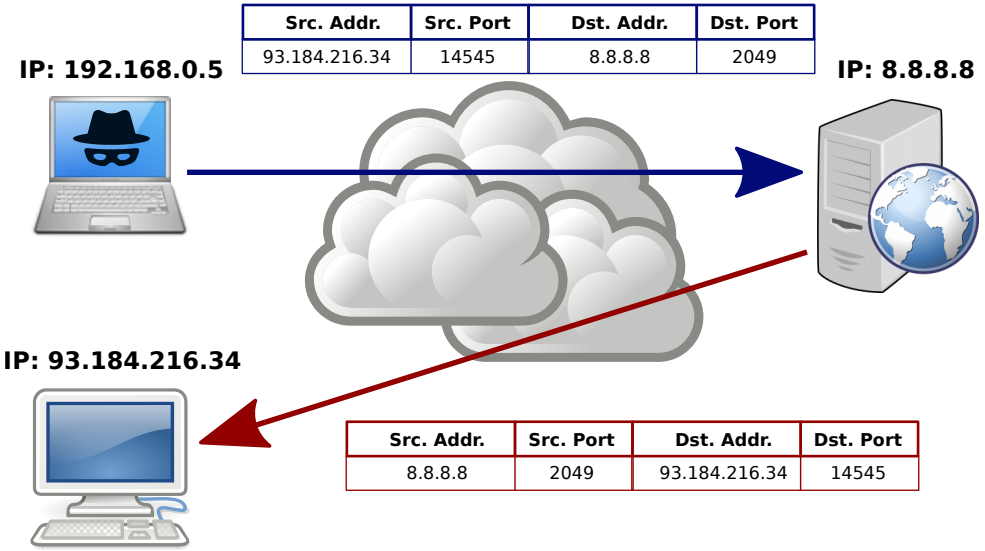
Is the simplest transport protocol that provides port abstraction to programs.
Best effort: messages are not acknowledged, reception is not guaranteed.
Popular protocols relying on UDP:

- Domain Name System (port 53);
- Dynamic Host Configuration Protocol (ports 67, 68);
- Network File System (port 2049).



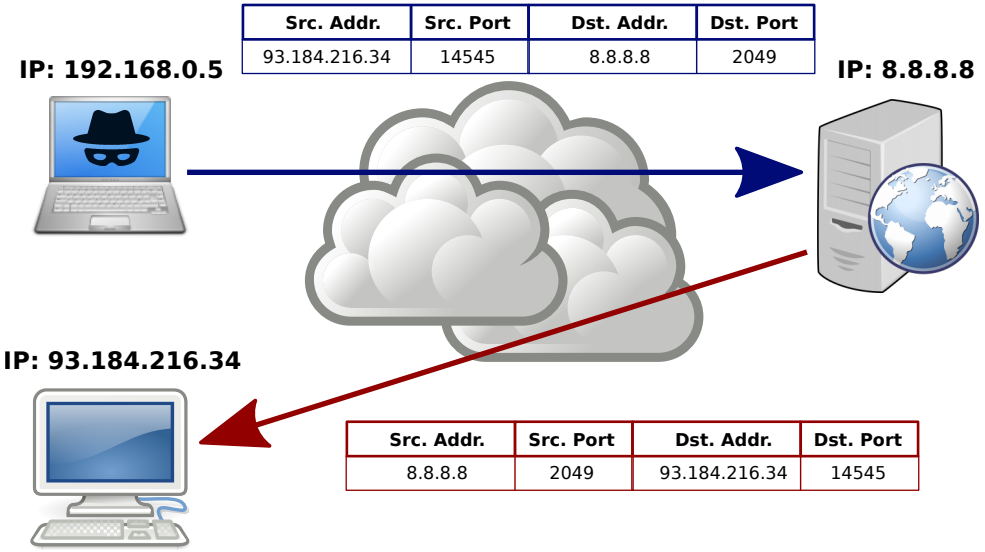
- **Source/Destination Port**: 65,535 possible options.
  - The service(s) listen on a *given port(s)*;
  - The client(s) create connections from *random ports*.
- **UDP Length**: total number of bytes (header + payload).
  The minimum length of an UDP packet is 8 bytes.

UDP Spoofing/Hijacking

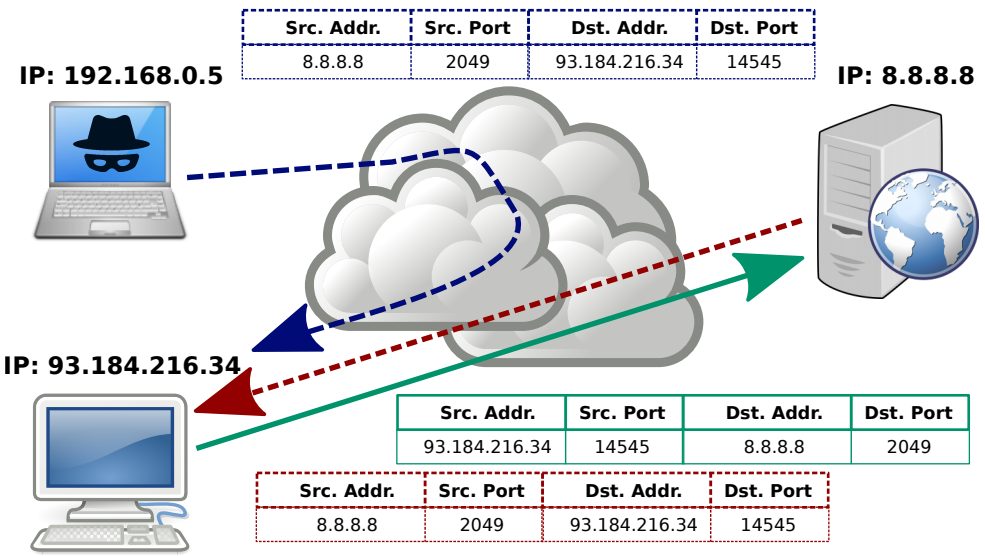


UDP spoofing: identical approach as for IP spoofing.

UDP Spoofing/Hijacking



UDP spoofing: identical approach as for IP spoofing.



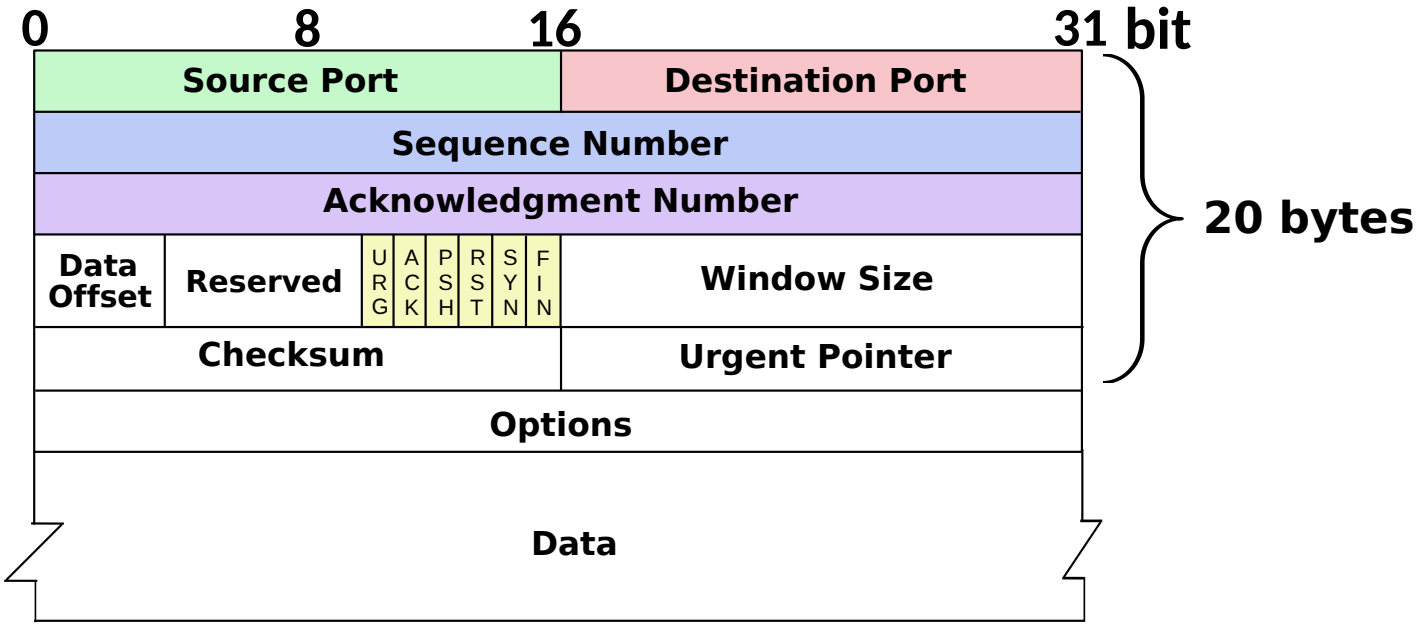
UDP hijacking: the attacker *competes* with the server to send to the victim a reply before the the packet from server is received.

The attacker sends to the victim the same header as the server.

TCP Segments

Identical port abstraction as UDP, but with a **connection-oriented and reliable** stream delivery service. It leverages *virtual circuits* which are composed of two streams and a full-duplex connection.
Popular protocols relying on TCP:

- Secure Shell (22);
- Hyper Text Transfer Protocol (port 80 and 443).

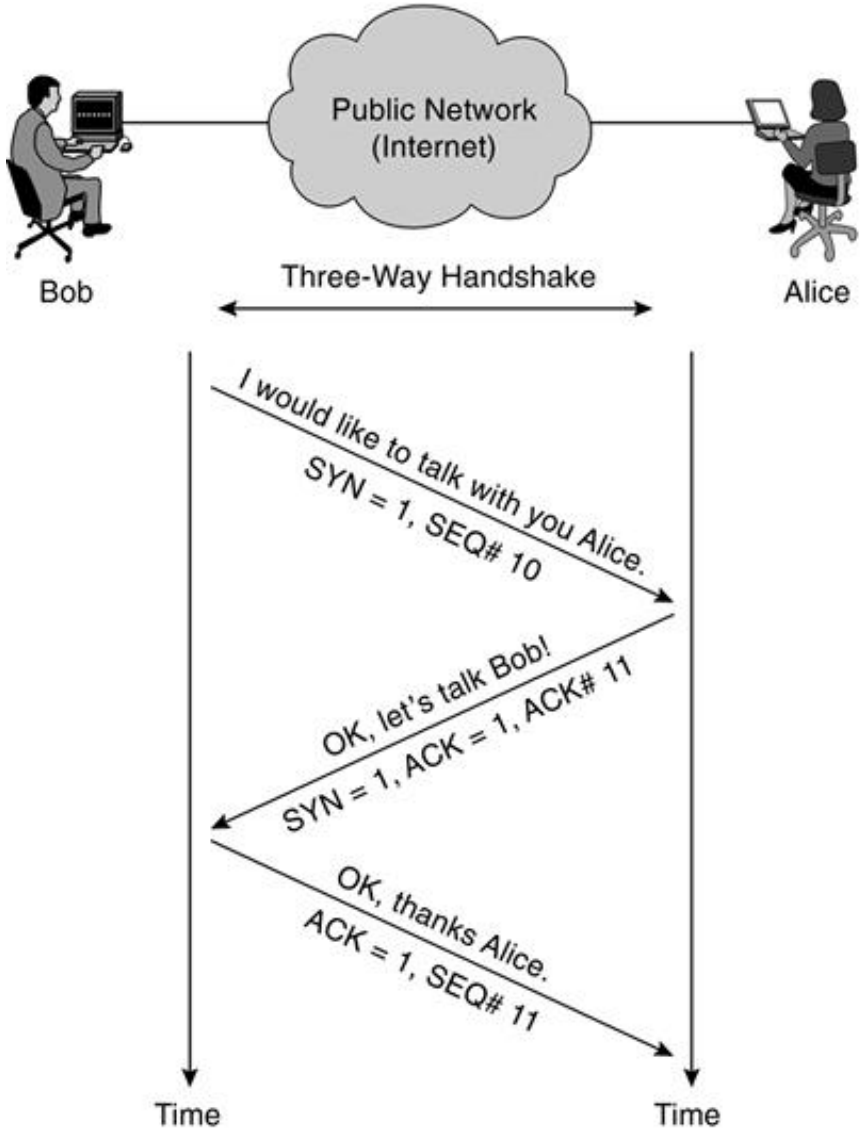


- **Source/Destination Port**: service that.
- **(Initial) Sequence Number (ISN)**: segment position in the stream.
- **Acknowledgment Number**: last segment received.
- **TCP flags**: used to setup and shutdown the virtual circuit.

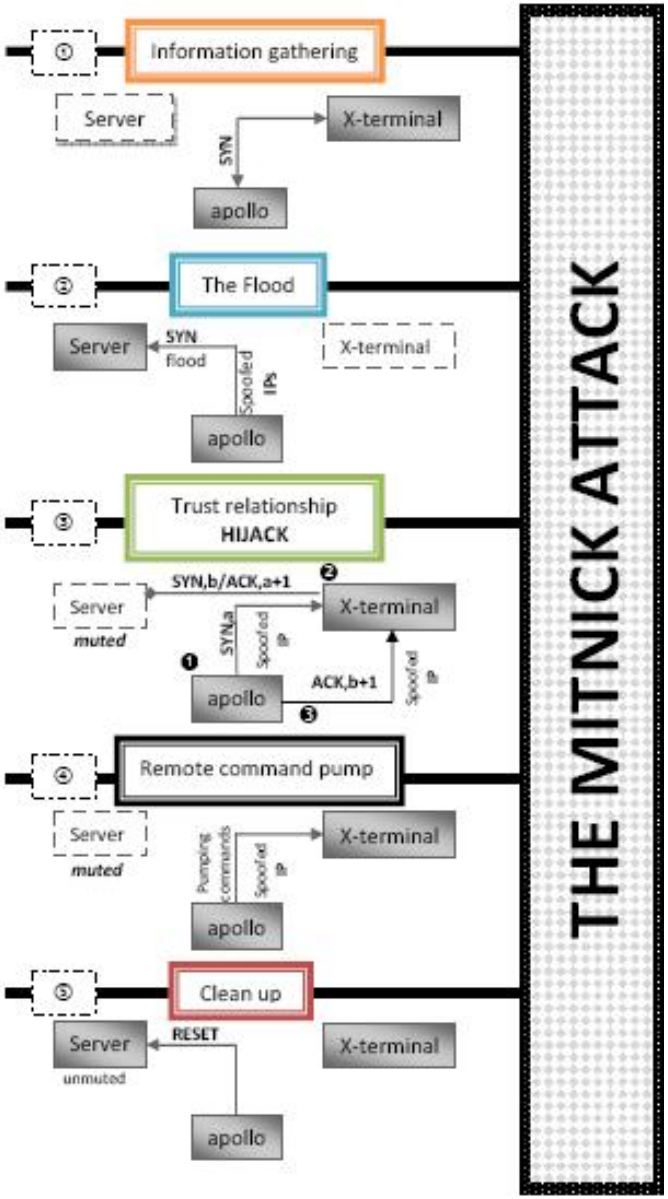
TCP Flags and the Three-way Handshake

Flags usage:

1. **SYN**: requests for synchronization of sequence and acknowledgment (ACK) numbers.
2. **ACK**: acknowledges the receipt of a segment.
   - ACK = sequence number + 1
3. **FIN**: requests the shutdown of the circuit (*gracious*).
4. **RST**: requests to immediately shutdown the circuit (*abrupt*).
5. **PSH**: requests to pass the data stream to the user level (program) as soon as possible.
6. **URG**: specifies that the data is urgent.



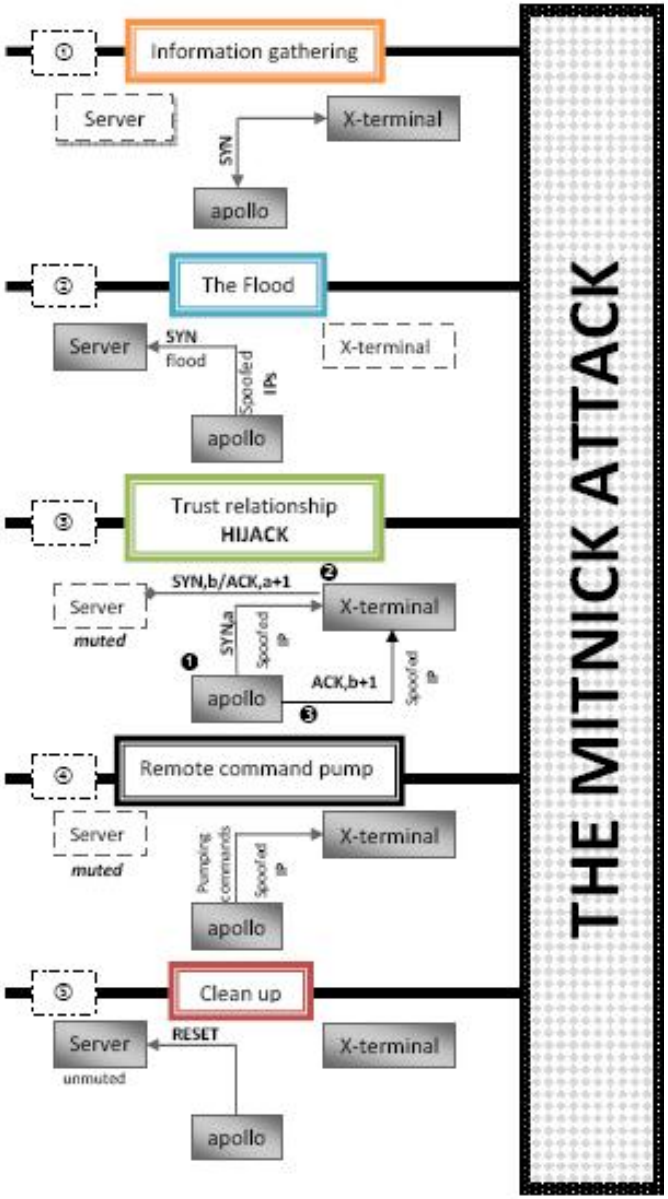
Mitnick's TCP Session Hijack Attack (1994)



1. Information gathering:
   - determine trusted relationships (IP, TCP ISN);
   - determine the TCP ISN

Source: `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`

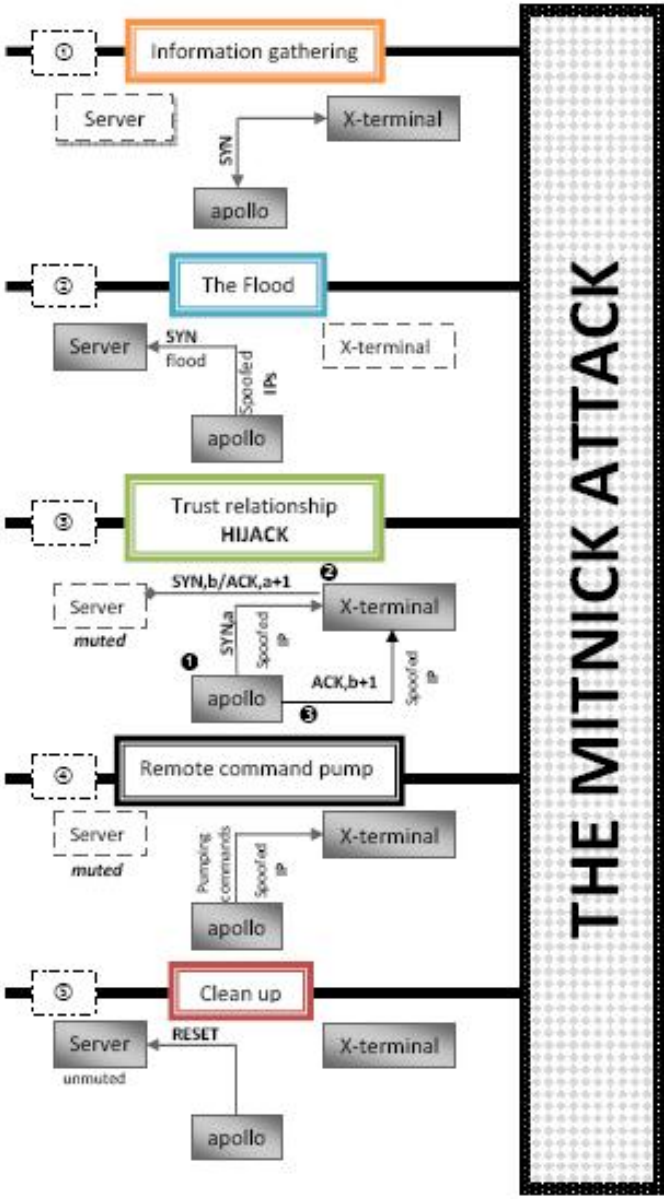
Mitnick's TCP Session Hijack Attack (1994)



1. Information gathering:
   - determine trusted relationships (IP, TCP ISN);
   - determine the TCP ISN
2. Server flooding:
   - fill the memory of server the with half-open SYN requests from spoofed IPs.

Source: `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`

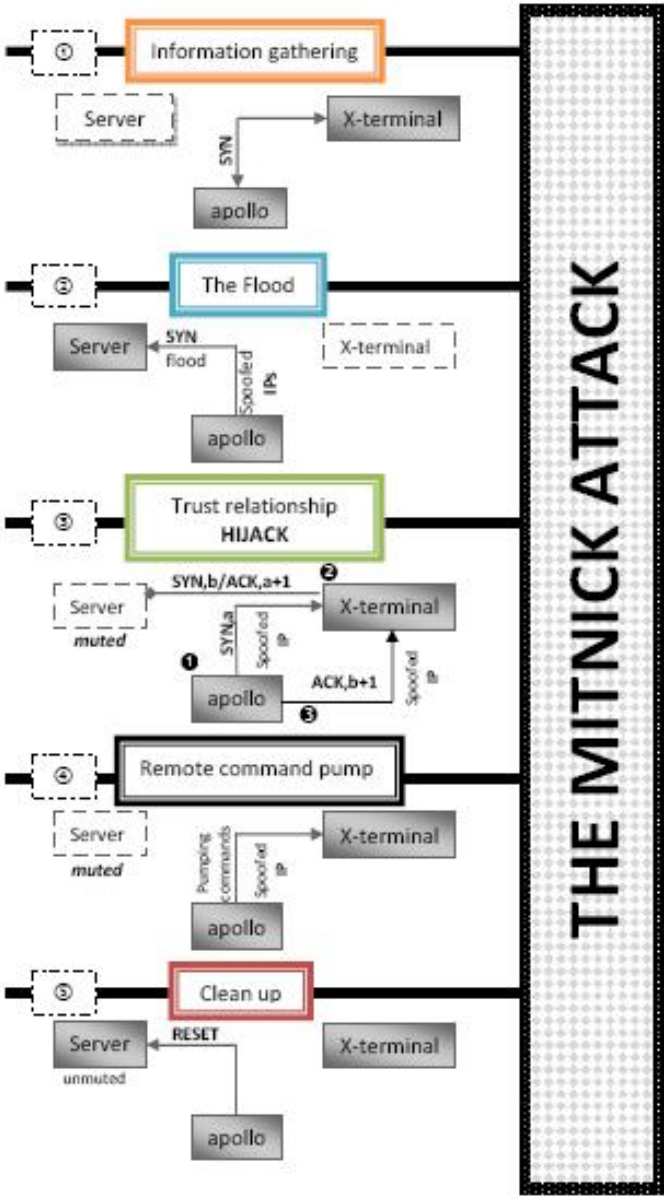
Mitnick's TCP Session Hijack Attack (1994)



1. Information gathering:
   - determine trusted relationships (IP, TCP ISN);
   - determine the TCP ISN
2. Server flooding:
   - fill the memory of server the with half-open SYN requests from spoofed IPs.
3. Session hijacking:

Source: `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`

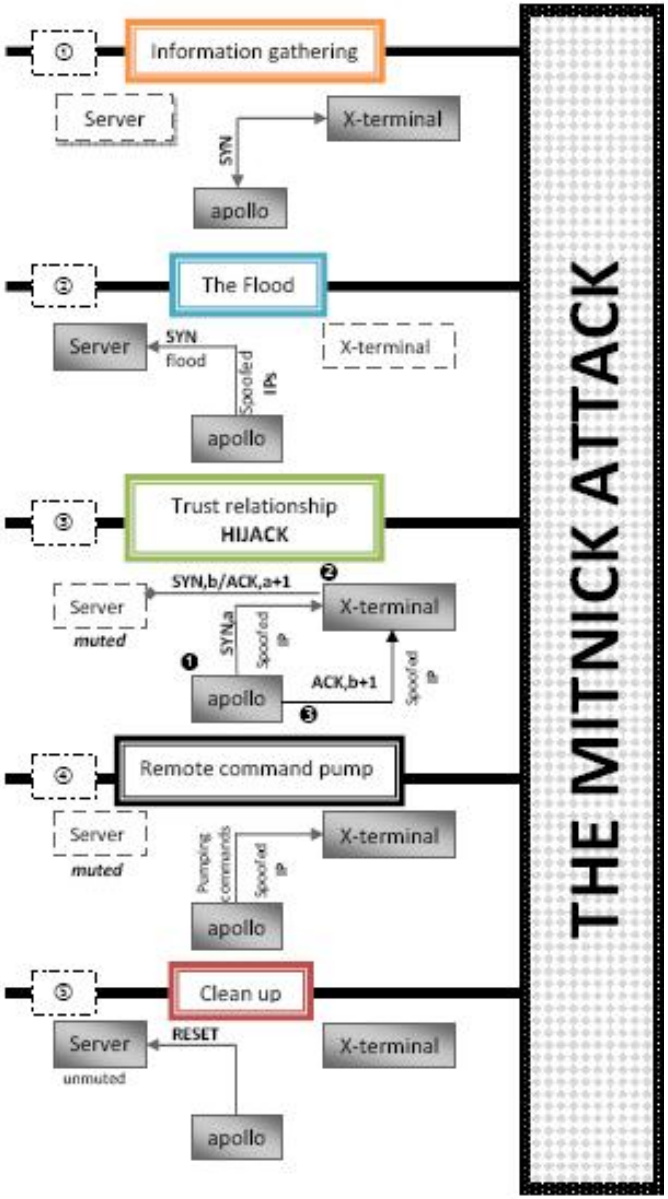
Mitnick's TCP Session Hijack Attack (1994)



1. Information gathering:
   - determine trusted relationships (IP, TCP ISN);
   - determine the TCP ISN
2. Server flooding:
   - fill the memory of server the with half-open SYN requests from spoofed IPs.
3. Session hijacking:
   - send SYN to the client with the spoofed address of the server;

Source: `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`

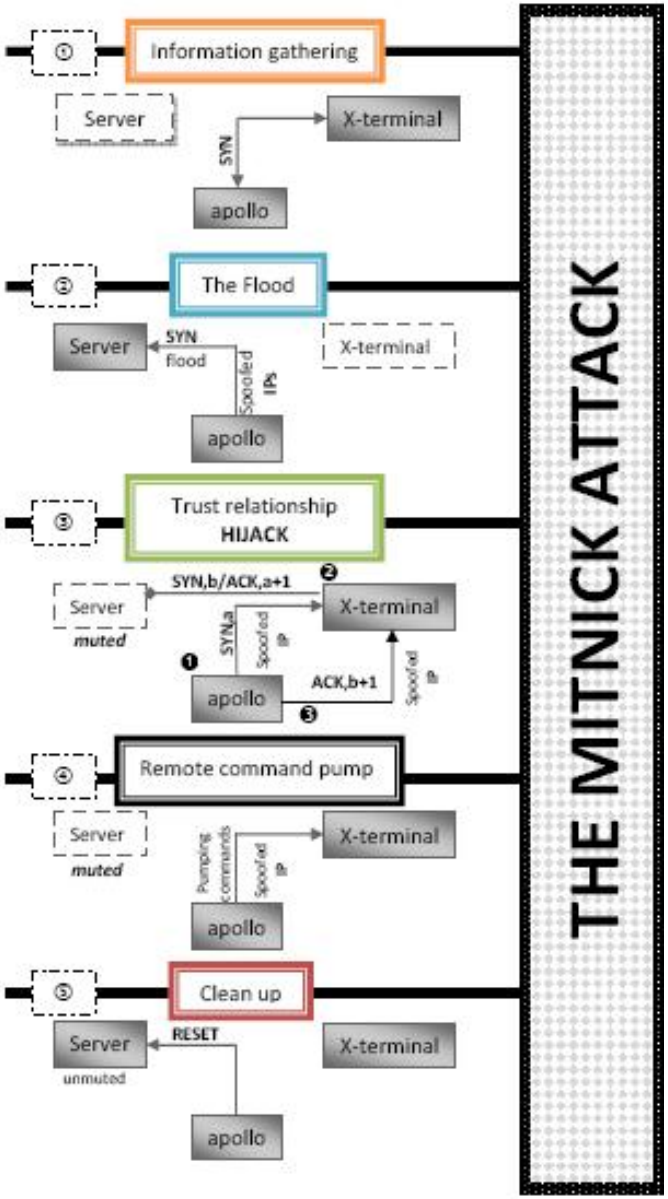
Mitnick's TCP Session Hijack Attack (1994)



1. Information gathering:
   - determine trusted relationships (IP, TCP ISN);
   - determine the TCP ISN
2. Server flooding:
   - fill the memory of server the with half-open SYN requests from spoofed IPs.
3. Session hijacking:
   - send SYN to the client with the spoofed address of the server;
   - client sends SYN/ACK to the server (which will not reply);

Source: `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`

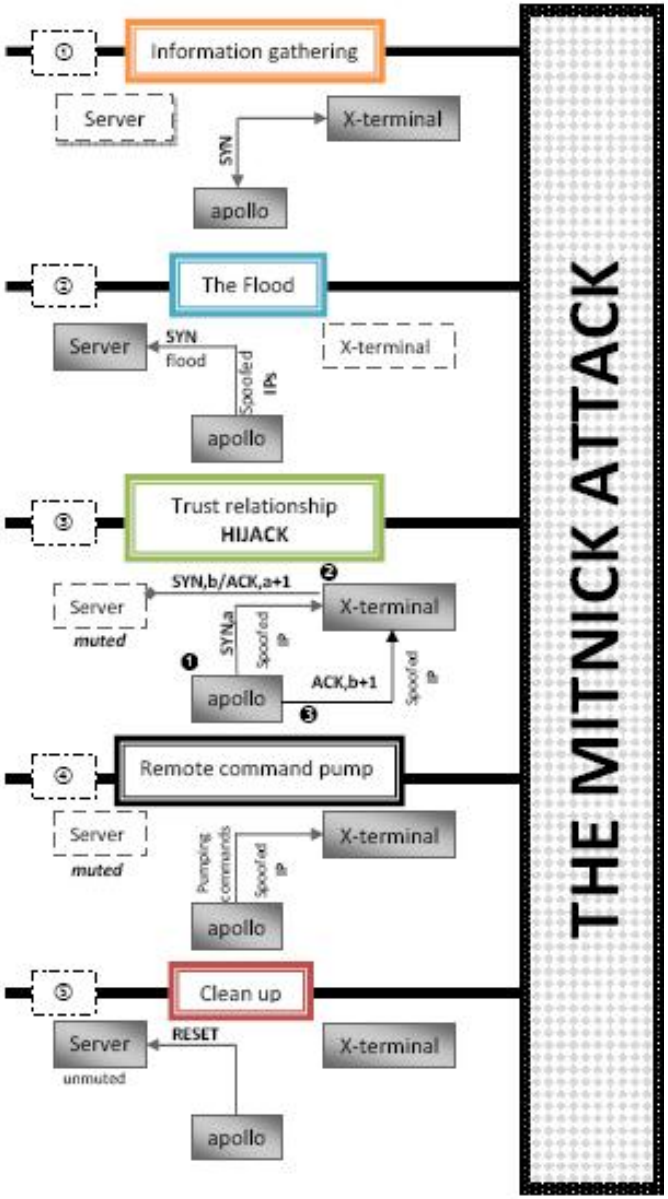
Mitnick's TCP Session Hijack Attack (1994)



1. Information gathering:
   - determine trusted relationships (IP, TCP ISN);
   - determine the TCP ISN
2. Server flooding:
   - fill the memory of server the with half-open SYN requests from spoofed IPs.
3. Session hijacking:
   - send SYN to the client with the spoofed address of the server;
   - client sends SYN/ACK to the server (which will not reply);
   - send spoofed ACK to the client and complete the three-way handshake.

Source: `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`

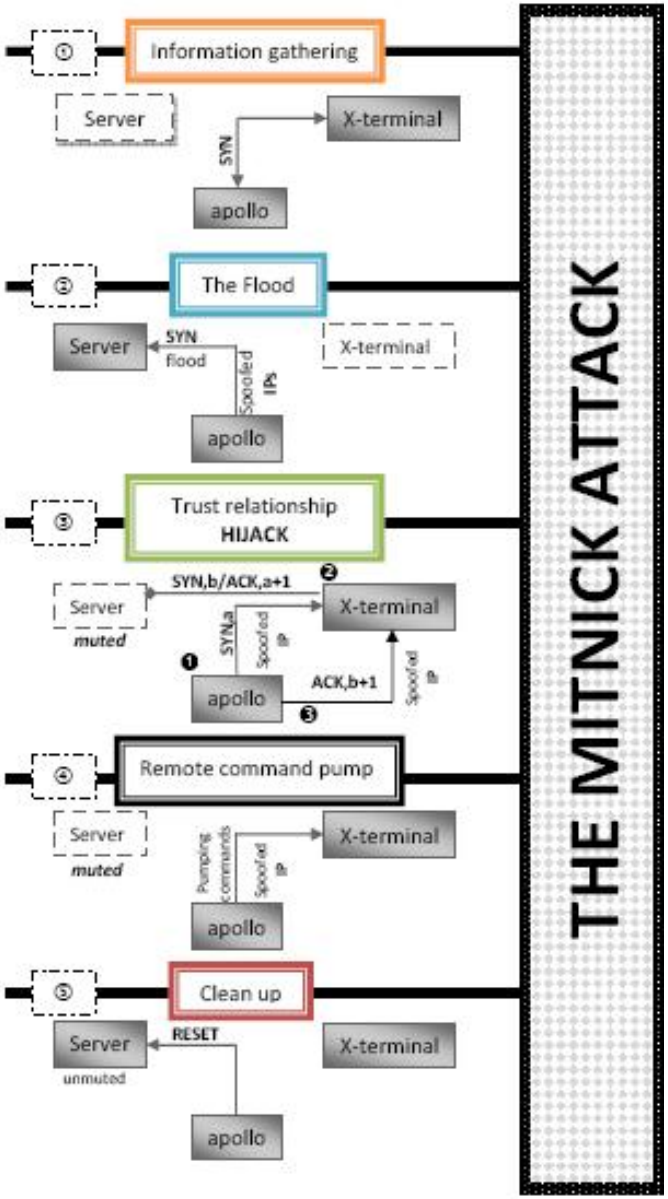
Mitnick's TCP Session Hijack Attack (1994)



1. Information gathering:
   - determine trusted relationships (IP, TCP ISN);
   - determine the TCP ISN
2. Server flooding:
   - fill the memory of server the with half-open SYN requests from spoofed IPs.
3. Session hijacking:
   - send SYN to the client with the spoofed address of the server;
   - client sends SYN/ACK to the server (which will not reply);
   - send spoofed ACK to the client and complete the three-way handshake.
4. Create the backdoor.

Source: `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`

Mitnick's TCP Session Hijack Attack (1994)



1. Information gathering:
   - determine trusted relationships (IP, TCP ISN);
   - determine the TCP ISN
2. Server flooding:
   - fill the memory of server the with half-open SYN requests from spoofed IPs.
3. Session hijacking:
   - send SYN to the client with the spoofed address of the server;
   - client sends SYN/ACK to the server (which will not reply);
   - send spoofed ACK to the client and complete the three-way handshake.
4. Create the backdoor.
5. Clean up:
   - send RST to server to cancel all the SYN requests.

Source: `http://wiki.cas.mcmaster.ca/index.php/The_Mitnick_attack`

Zalewski's analysis of ISNs (2001, 2002)

The security of the protocol relies on the *difficulty* to guess the TCP sequence number. Any attacker that can generate (*and send*!) 2^{32} packets would be able to guess the sequence number.
Unfortunately, many OS had a flow in the way they used to generate sequence numbers[5,6]

Zalewski's assumptions:

- the attacker collected around 50,000 SN values;
- the attacker will perform at most 5,000 attempts to guess the SN
  $\Rightarrow$ this is achievable in a *few seconds*.

Test date: April 2001
OS: Windows 98 SE
Attack feasibility: 100%

Test date: April 2001
OS: MacOS 9
Attack feasibility: 89%

Test date: October 2002
OS: MacOS X
Attack feasibility: 0%

[5] https://lcamtuf.coredump.cx/oldtcp/tcpseq.html
[6] https://lcamtuf.coredump.cx/newtcp/

TCP Hijacking in 2016

In the original TCP Linux implementation, TCP segments that fall within a certain range can be accepted - this opens the field to blind RST attacks and data injection attacks.

RFC 5961 introduces a challenge ACKs, which are sent to the source IP upon receiving a SYN packet for resetting an ongoing a TCP connection. Challenge ACKs are rate limited, and monitored using a single global variable for all TCP connections.

An attacker can leverage this information to:

- **check if two arbitrary hosts are communicating** using a TCP connection (and discover the port numbers on each host);
- **infer the sequence number and the ACK number** on each host.

Leveraging the above knowledge, the attacker can tear down connections or inject arbitrary data. Attack success rate: 88-97%; time required to carry out the attack: 40-60 seconds.

More details about the attack in the paper:
`https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/cao`

Port Scanning (UDP and TCP)

The attacker needs to know which services are active on a host.
The process of enumerating the listening services is called **portscan**.
Sweeping: when the attacker scans multiple hosts on a single port.

Port Scanning (UDP and TCP)

The attacker needs to know which services are active on a host.
The process of enumerating the listening services is called **portscan**.
Sweeping: when the attacker scans multiple hosts on a single port.

UDP port scan: send an empty UDP datagram; the response is:

- **any response**
  $\Rightarrow$ port is open
- **no response**
  $\Rightarrow$ port is open *or filtered (e.g. firewall)*
- **ICMP port unreachable error**
  $\Rightarrow$ port is closed

Port Scanning (UDP and TCP)

The attacker needs to know which services are active on a host.
The process of enumerating the listening services is called **portscan**.
Sweeping: when the attacker scans multiple hosts on a single port.

UDP port scan: send an empty UDP datagram; the response is:

- **any response**
  ⇒ port is open
- **no response**
  ⇒ port is open *or filtered (e.g. firewall)*
- **ICMP port unreachable error**
  ⇒ port is closed

TCP SYN scan: send a segment with the SYN flag; the response is:

- **SYN + ACK**
  ⇒ port is open
- **RST**
  ⇒ the port is closed
- **no response**
  ⇒ port is filtered

Port Scanning (UDP and TCP)

The attacker needs to know which services are active on a host.
The process of enumerating the listening services is called **portscan**.
Sweeping: when the attacker scans multiple hosts on a single port.

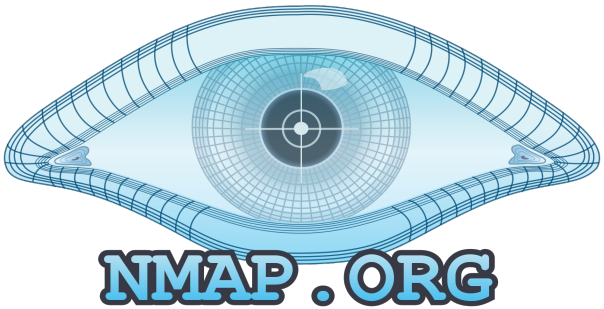
UDP port scan: send an empty UDP datagram; the response is:

- **any response**
  ⇒ port is open
- **no response**
  ⇒ port is open *or filtered (e.g. firewall)*
- **ICMP port unreachable error**
  ⇒ port is closed

TCP SYN scan: send a segment with the SYN flag; the response is:

- **SYN + ACK**
  ⇒ port is open
- **RST**
  ⇒ the port is closed
- **no response**
  ⇒ port is filtered

Popular open-source tools:



Port Scanning (UDP and TCP)

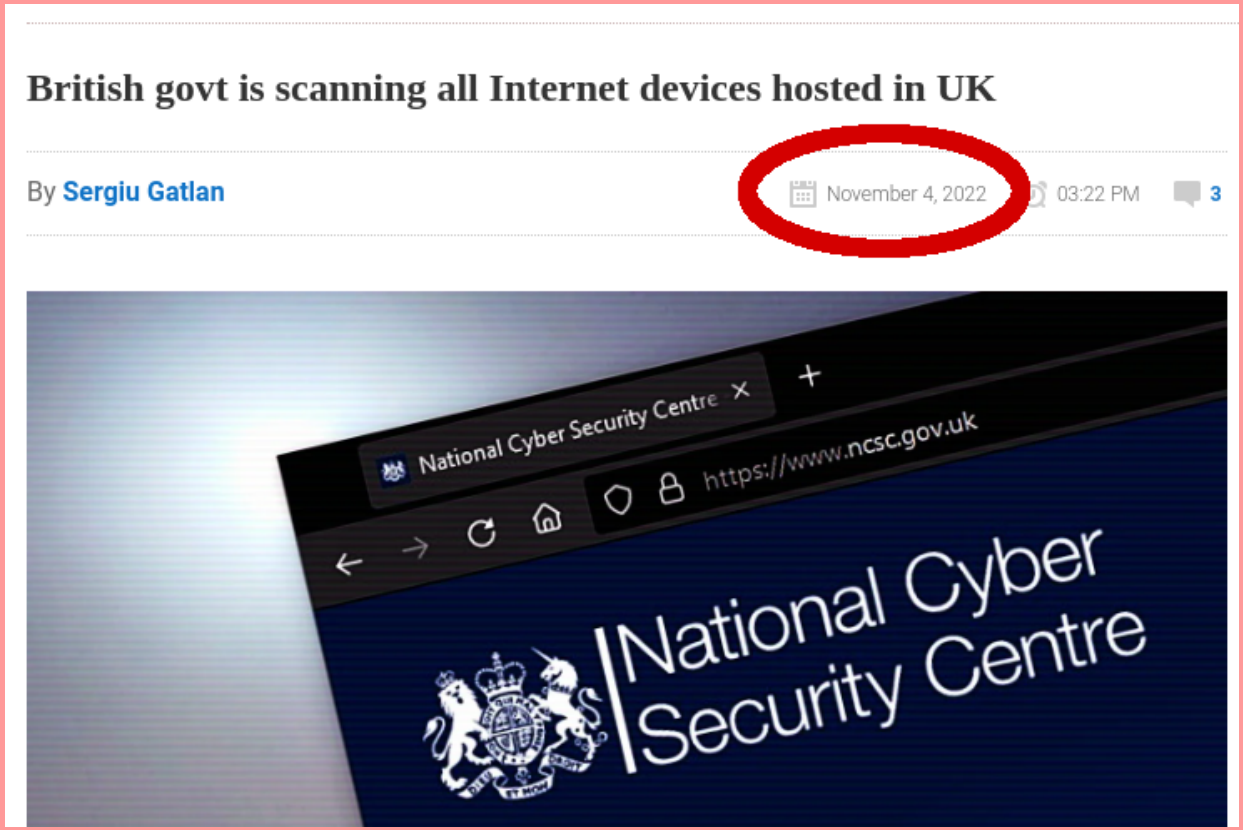
The
The
Swe

UDP
data

- a
- r
- I

Pop

In case you thought that this is just *"hackers business"*



Sources:

https://www.bleepingcomputer.com/news/security/british-govt-is-scanning-all-internet-devices-hosted-in-uk/

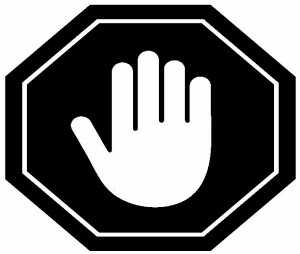
https://www.ncsc.gov.uk/information/ncsc-scanning-information

https://www.ncsc.gov.uk/blog-post/scanning-the-internet-for-fun-and-profit

Port Scanning (TCP) - Alternative Techniques

From RFC 793:

- port is closed && segment without RST flag arrives ⇒ reply with RST
- port is open && a segment without SYN/RST/ACK flag arrives ⇒ drop the segment



NULL, FIN, PSH, URG scans response contains:

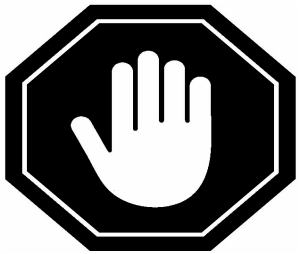
- **RST**: closed
- **No response**: open *or filtered* (e.g., firewall)
- **ICMP unreachable error**: filtered

References:
`https://www.rfc-editor.org/rfc/rfc793` (page 65) and
`https://nmap.org/book/scan-methods-ack-scan.html`

Port Scanning (TCP) - Alternative Techniques

From RFC 793:

- port is closed && segment without RST flag arrives ⇒ reply with RST
- port is open && a segment without SYN/RST/ACK flag arrives ⇒ drop the segment



NULL, FIN, PSH, URG scans response contains:

- **RST**: closed
- **No response**: open *or filtered* (e.g., firewall)
- **ICMP unreachable error**: filtered

ACK scan response contains:

- **RST**: not filtered
- **No response**: filtered
- **ICMP unreachable error**: filtered

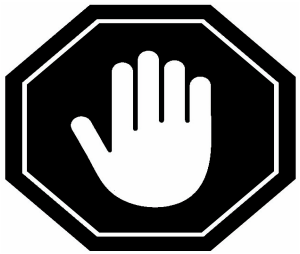
⇒ is used for testing firewall rules

References:
`https://www.rfc-editor.org/rfc/rfc793` (page 65) and
`https://nmap.org/book/scan-methods-ack-scan.html`

Port Scanning (TCP) - Alternative Techniques

From RFC 793:

- port is closed && segment without RST flag arrives ⇒ reply with RST
- port is open && a segment without SYN/RST/ACK flag arrives ⇒ drop the segment



NULL, FIN, PSH, URG scans response contains:

- **RST**: closed
- **No response**: open *or filtered* (e.g., firewall)
- **ICMP unreachable error**: filtered

ACK scan response contains:

- **RST**: not filtered
- **No response**: filtered
- **ICMP unreachable error**: filtered

⇒ is used for testing firewall rules

connect() scan:

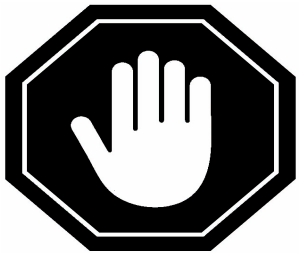
- Invasive ⇒ Why do we need it?

References:
`https://www.rfc-editor.org/rfc/rfc793` (page 65) and
`https://nmap.org/book/scan-methods-ack-scan.html`

Port Scanning (TCP) - Alternative Techniques

From RFC 793:

- port is closed && segment without RST flag arrives ⇒ reply with RST
- port is open && a segment without SYN/RST/ACK flag arrives ⇒ drop the segment



NULL, FIN, PSH, URG scans response contains:

- **RST**: closed
- **No response**: open *or filtered* (e.g., firewall)
- **ICMP unreachable error**: filtered

ACK scan response contains:

- **RST**: not filtered
- **No response**: filtered
- **ICMP unreachable error**: filtered

⇒ is used for testing firewall rules

connect() scan:

- Invasive ⇒ Why do we need it?
- **Allows to grab banners**

References:
`https://www.rfc-editor.org/rfc/rfc793` (page 65) and
`https://nmap.org/book/scan-methods-ack-scan.html`

OS Fingerprinting

Why?

- Can detect old, outdated or unpatched services.
- Allows to focus on particular vulnerabilities and develop exploits.
- Provides information on the hosts/elements of a network.
- Can identify "suspicious" machines which are not expected to be in the network.

How?

Operating systems implement stacks in *slightly different* ways.
If we send **specially crafted packets** we can infer the OS that is running on a host.

Examples:

1. IP TTL: Linux sets the initial TTL to 64, Windows uses 128.
2. Windows and Cisco IOS do not reply with a RST to FIN packets (as in RFC 793).
3. Infer OS from the pattern used in the TCP sequence number.

Two ways for performing OS fingerprinting:

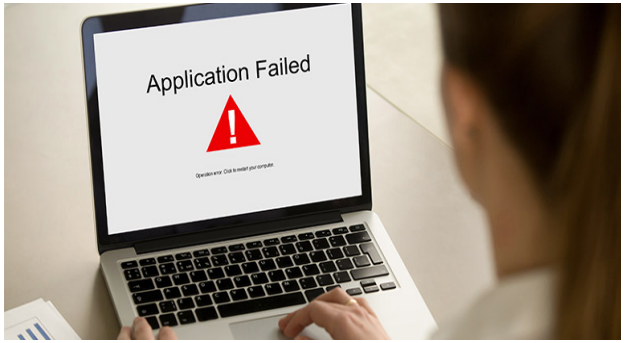
1. **Active**: send specific packets (e.g., nmap).
2. **Passive**: passively observe traffic and infer the OS (e.g., p0f).

References: `https://nmap.org/book/osdetect.html`

Denial of Service (DoS)

What is DoS?

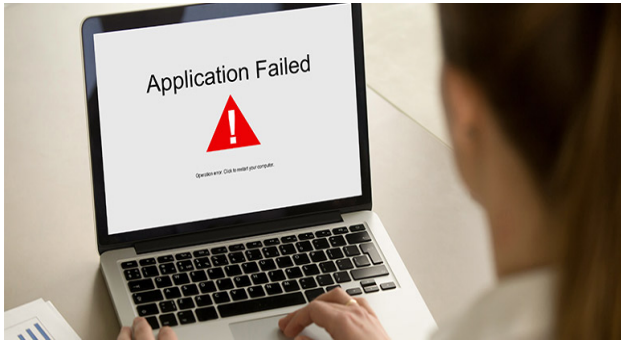
A type of **cyber attack** in which a malicious actor aims to render a computer or other device unavailable to its intended users by **interrupting the normal functioning of the device**.



Denial of Service (DoS)

What is DoS?

A type of **cyber attack** in which a malicious actor aims to render a computer or other device unavailable to its intended users by **interrupting the normal functioning of the device**.

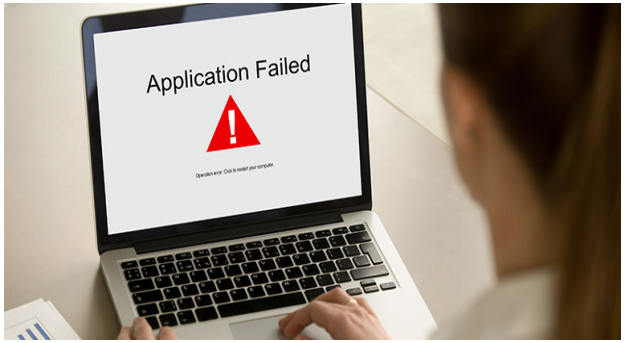


How?

Denial of Service (DoS)

What is DoS?

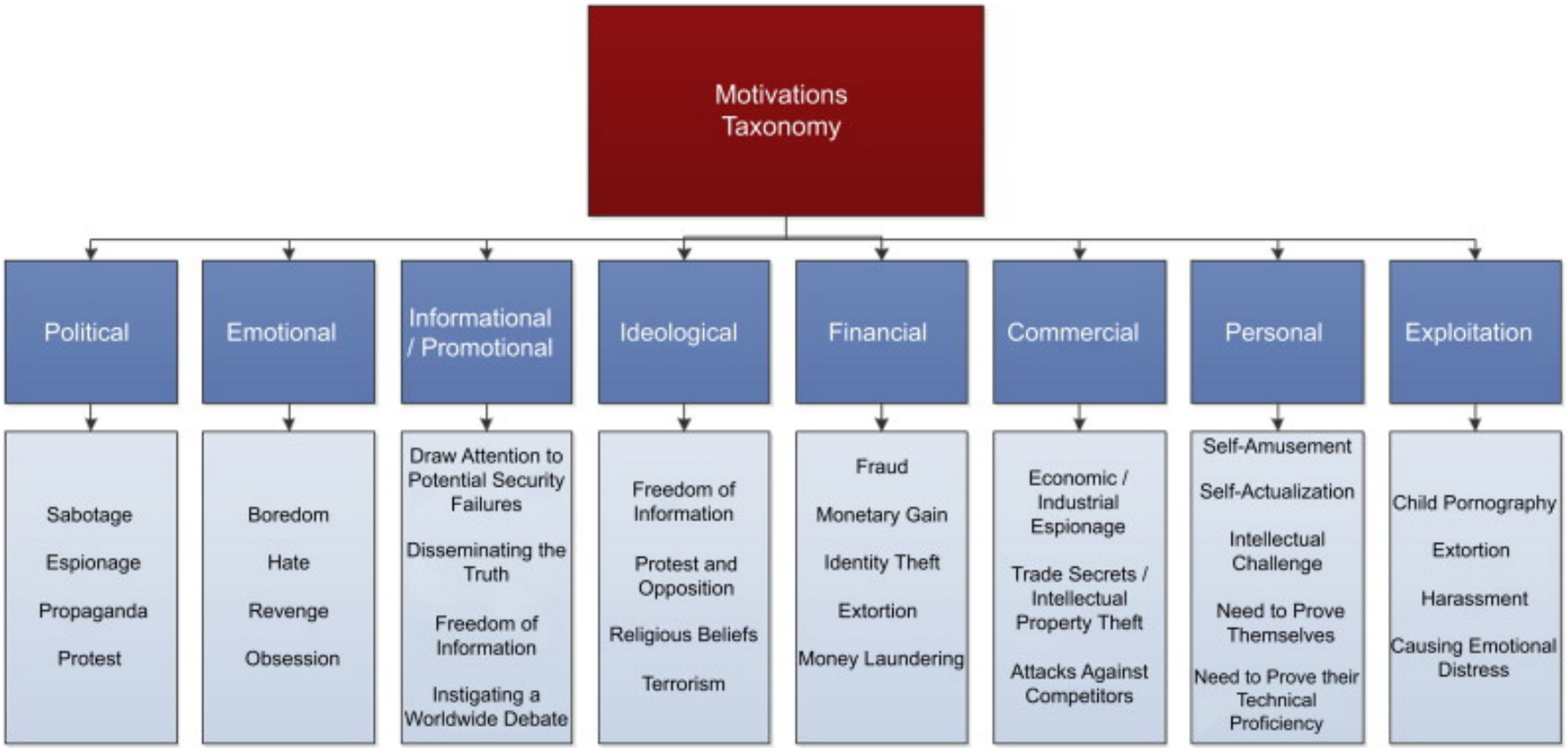
A type of **cyber attack** in which a malicious actor aims to render a computer or other device unavailable to its intended users by **interrupting the normal functioning of the device**.



The attacker sends huge volume of traffic over the network to **to starve victim's resources** in terms of:

- Network bandwidth ⇒ flooding.
- Network routing ⇒ sinkhole, router hijacking.
- CPU processing.
- Memory.
- Protocol state ⇒ SYN flooding (e.g., Mitnick's attack)

DoS - Motivations



Source: `https://www.perimeter81.com/blog/network/the-psychology-behind-ddos-attacks`

DoS - Attack Properties

- Type of resource being starved.
- Type of traffic sent:
  - UDP, TCP, SYN, ...

DoS - Attack Properties

- Type of resource being starved.
- Type of traffic sent:
  - UDP, TCP, SYN, ...

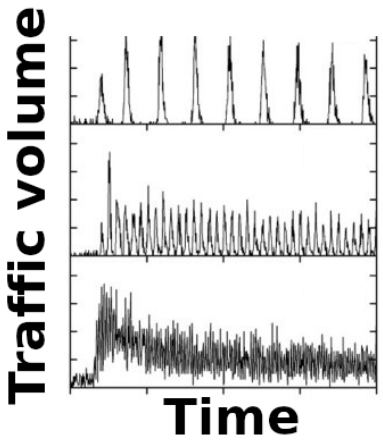
- Spoofing used.
- Source(s):
  - **Centralized** $\rightarrow$ single or few attackers.
  - **Distributed (DDoS)** $\rightarrow$ many attackers (e.g., botnet).
- Third-party involvement.
  - Reflector amplification.

DoS - Attack Properties

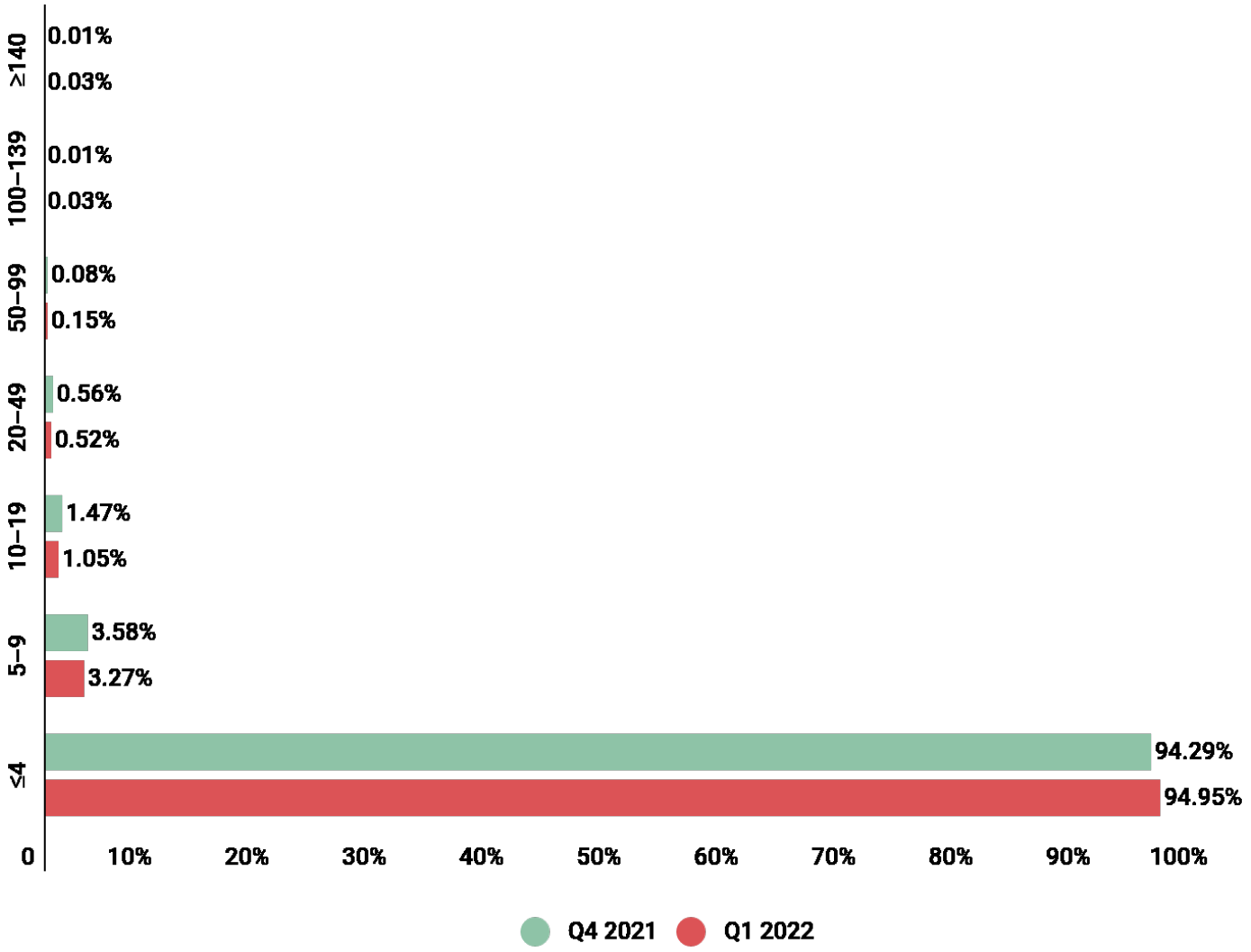
- Type of resource being starved.
- Type of traffic sent:
  - UDP, TCP, SYN, ...

- Spoofing used.
- Source(s):
  - **Centralized** $\rightarrow$ single or few attackers.
  - **Distributed (DDoS)** $\rightarrow$ many attackers (e.g., botnet).
- Third-party involvement.
  - Reflector amplification.

- Duration:
  - Constant, periodic or temporary.

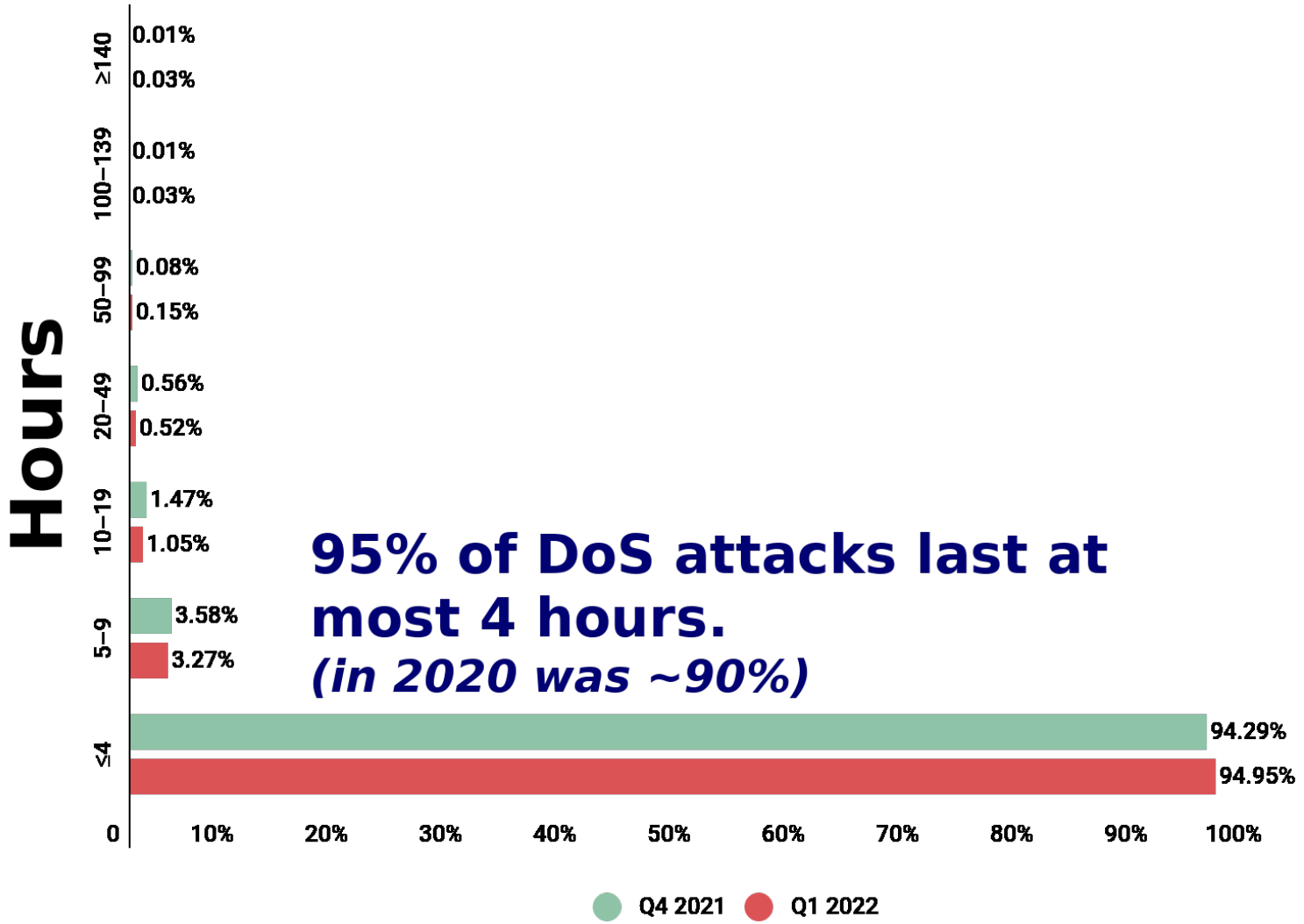


DoS - Duration



Sources:
`https://securelist.com/ddos-attacks-in-q1-2022/106358/`

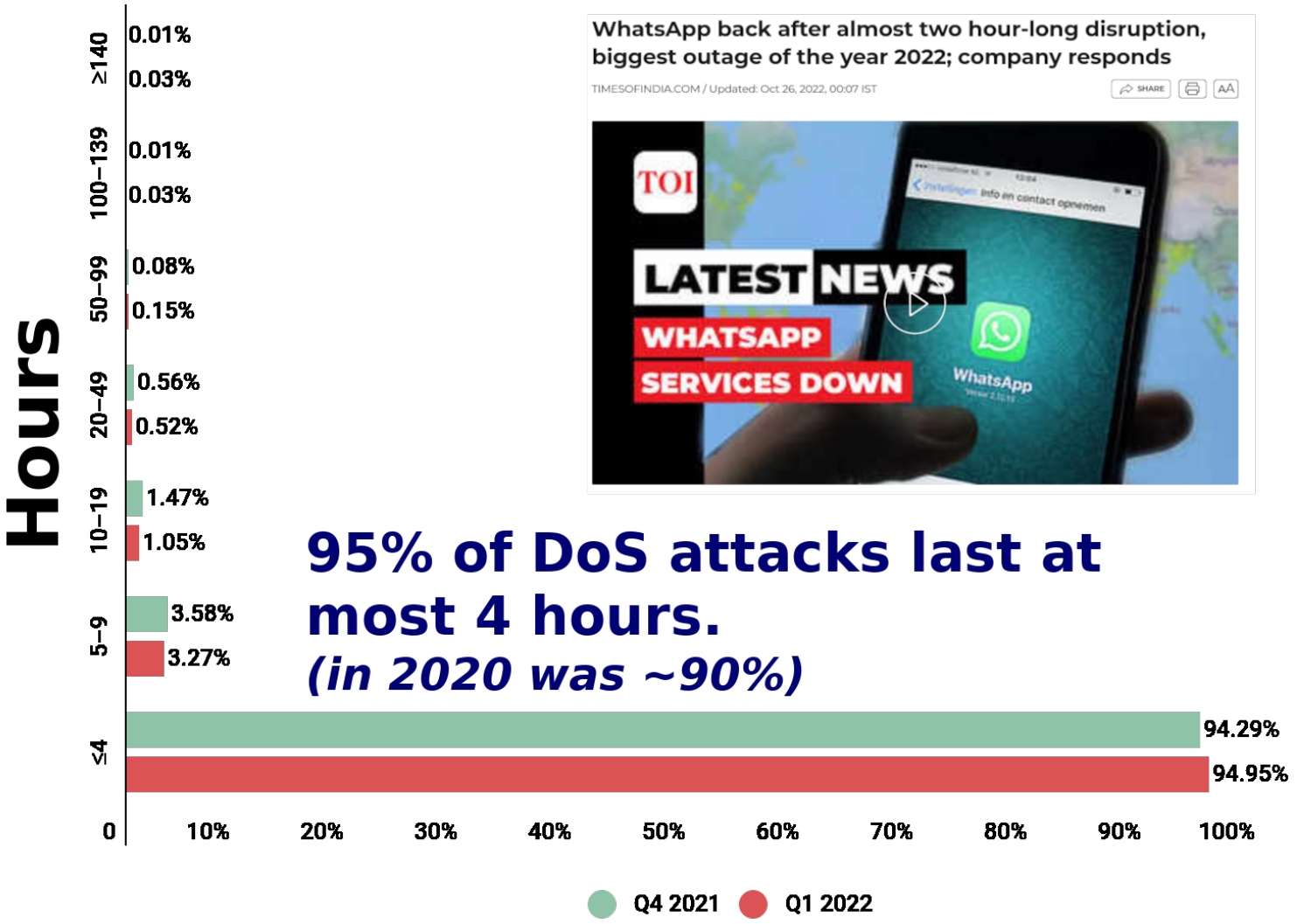
DoS - Duration



95% of DoS attacks last at most 4 hours.
(in 2020 was ~90%)

Sources:
`https://securelist.com/ddos-attacks-in-q1-2022/106358/`

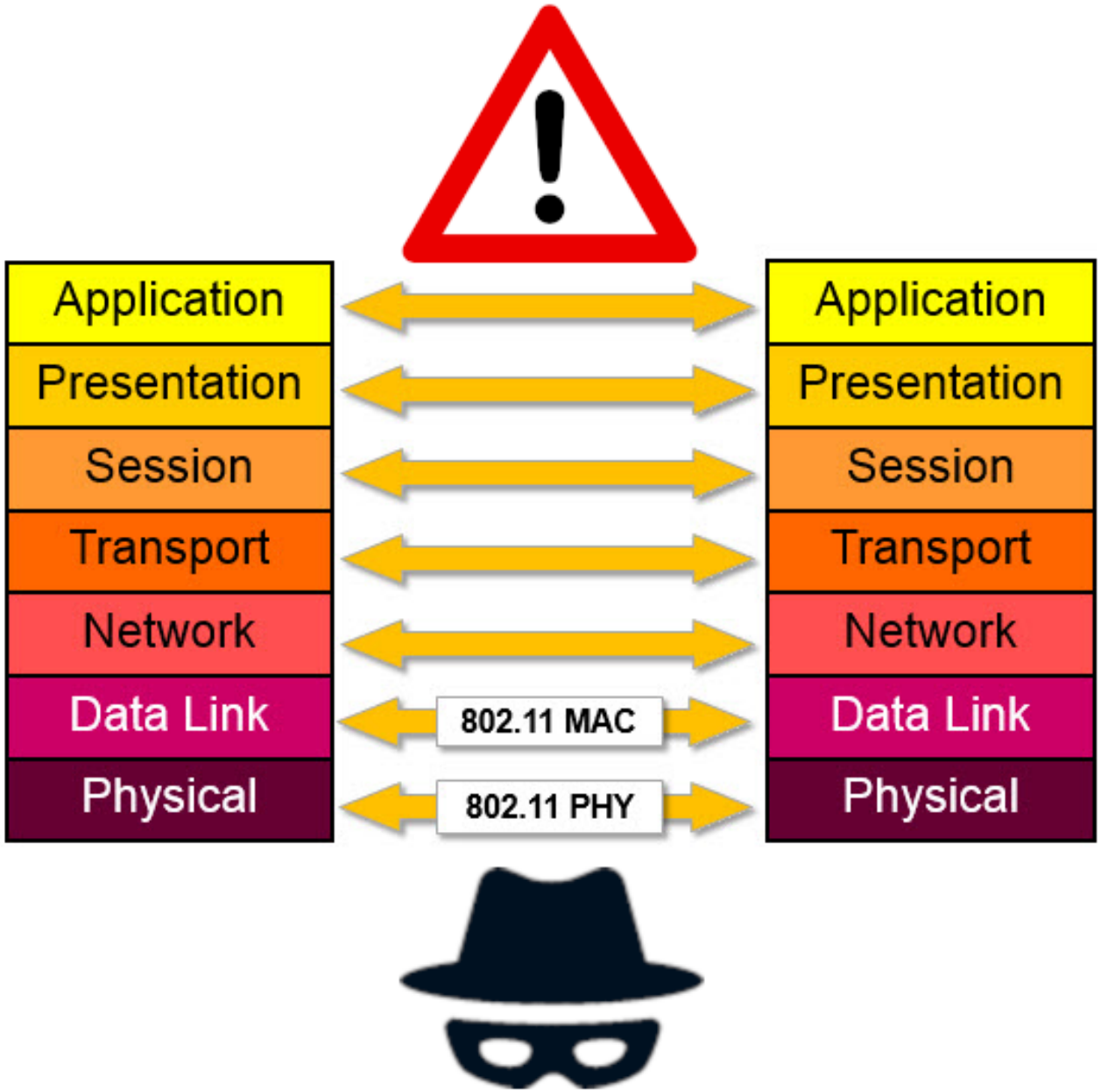
DoS - Duration



95% of DoS attacks last at most 4 hours.
(in 2020 was ~90%)

Sources:
https://securelist.com/ddos-attacks-in-q1-2022/106358/
https://timesofindia.indiatimes.com/gadgets-news/whatsapp-down-for-some-users/articleshow/95075496.cms

DoS can Target any Layer



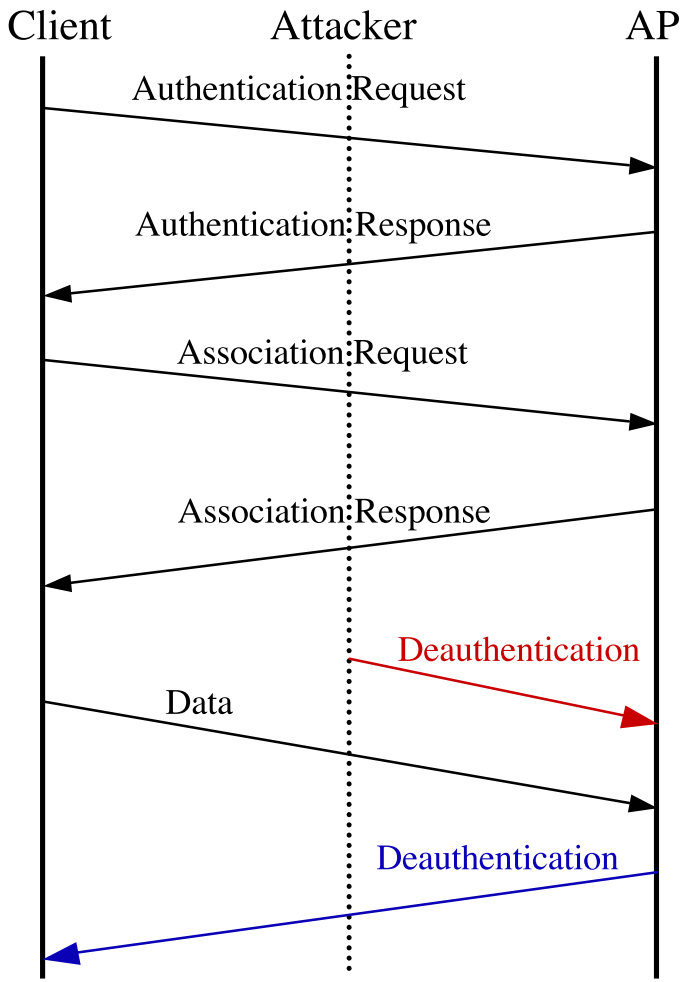
DoS Attacks on 802.11

1. Physical layer jamming

DoS Attacks on 802.11

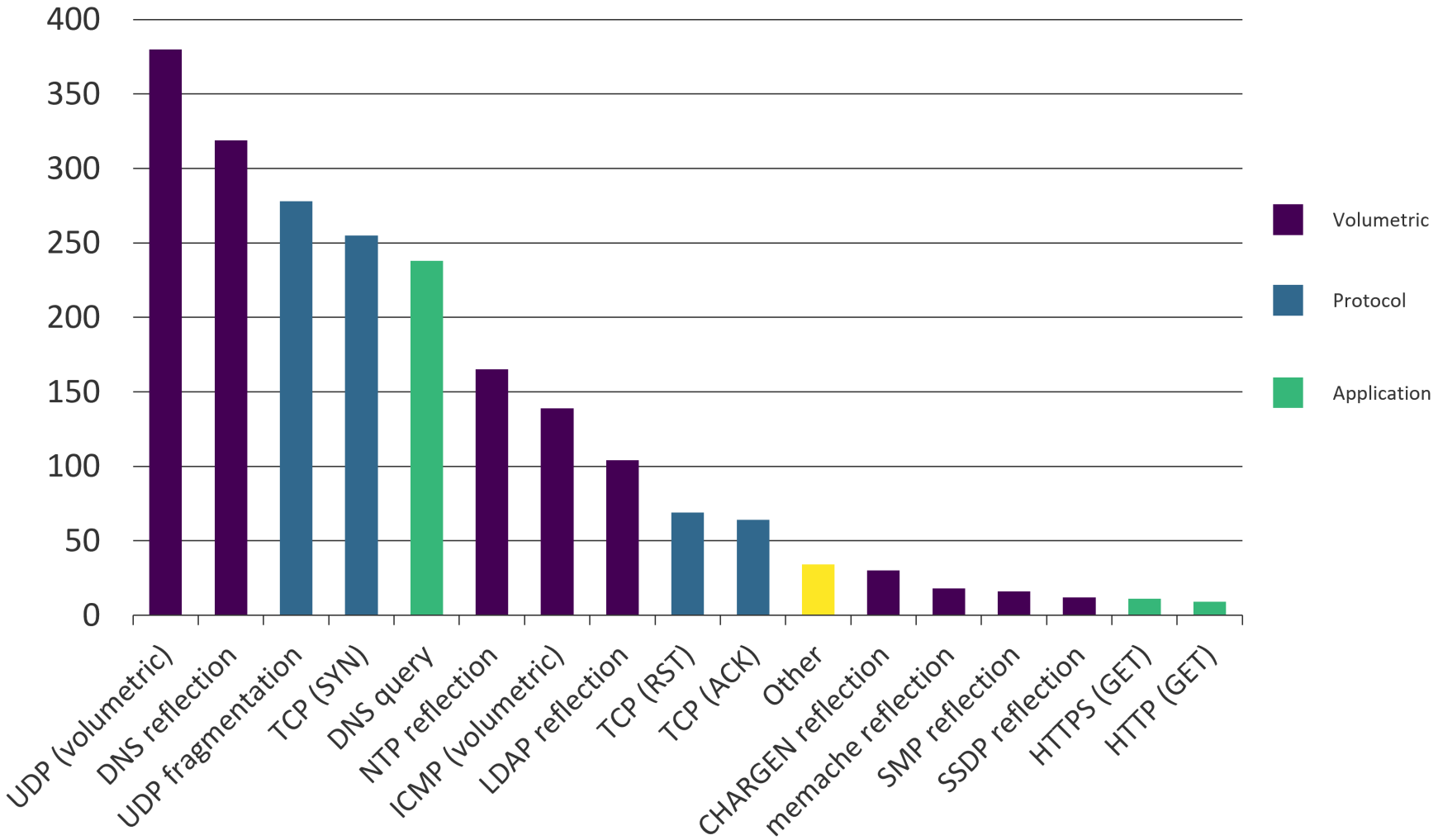
1. Physical layer jamming

2. Link Layer
   - **Deauthentication**
     - The deauthentication frame is not authenticated
     - one attacker packet every 6 victim packets
   - **Channel reservation**
     - Client can reserve channels for up to 2^{15} seconds
       $\Rightarrow$ the attacker can block a channel



References:
`https://www.usenix.org/legacy/events/sec03/tech/full_papers/bellardo/bellardo.pdf`

DoS Attack Types (2021)



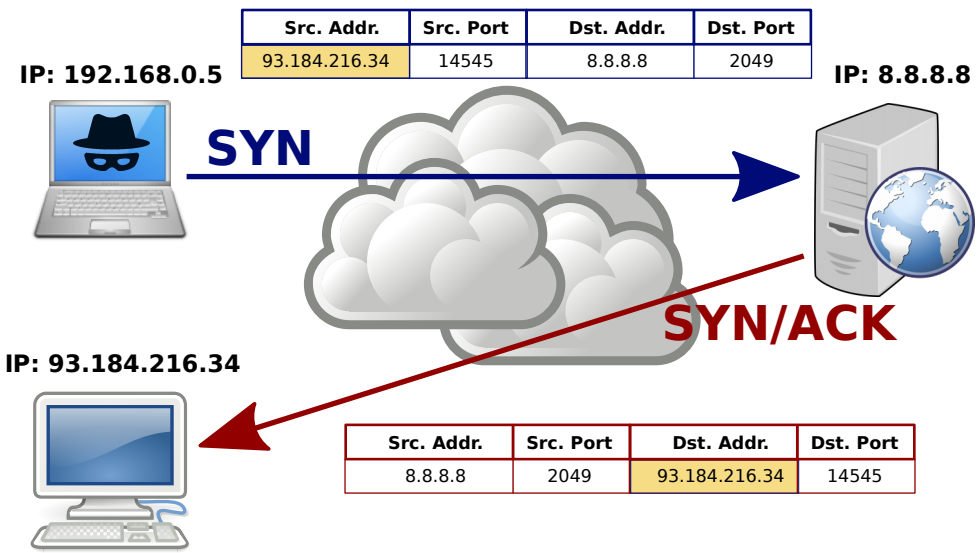
Source: `https://www.f5.com/labs/articles/threat-intelligence/2022-application-protection-report-ddos-attack-trends`

Spoofing

Issue:

Missing authentication for IP addresses
$\Rightarrow$ the attacker can *fake the source address*.

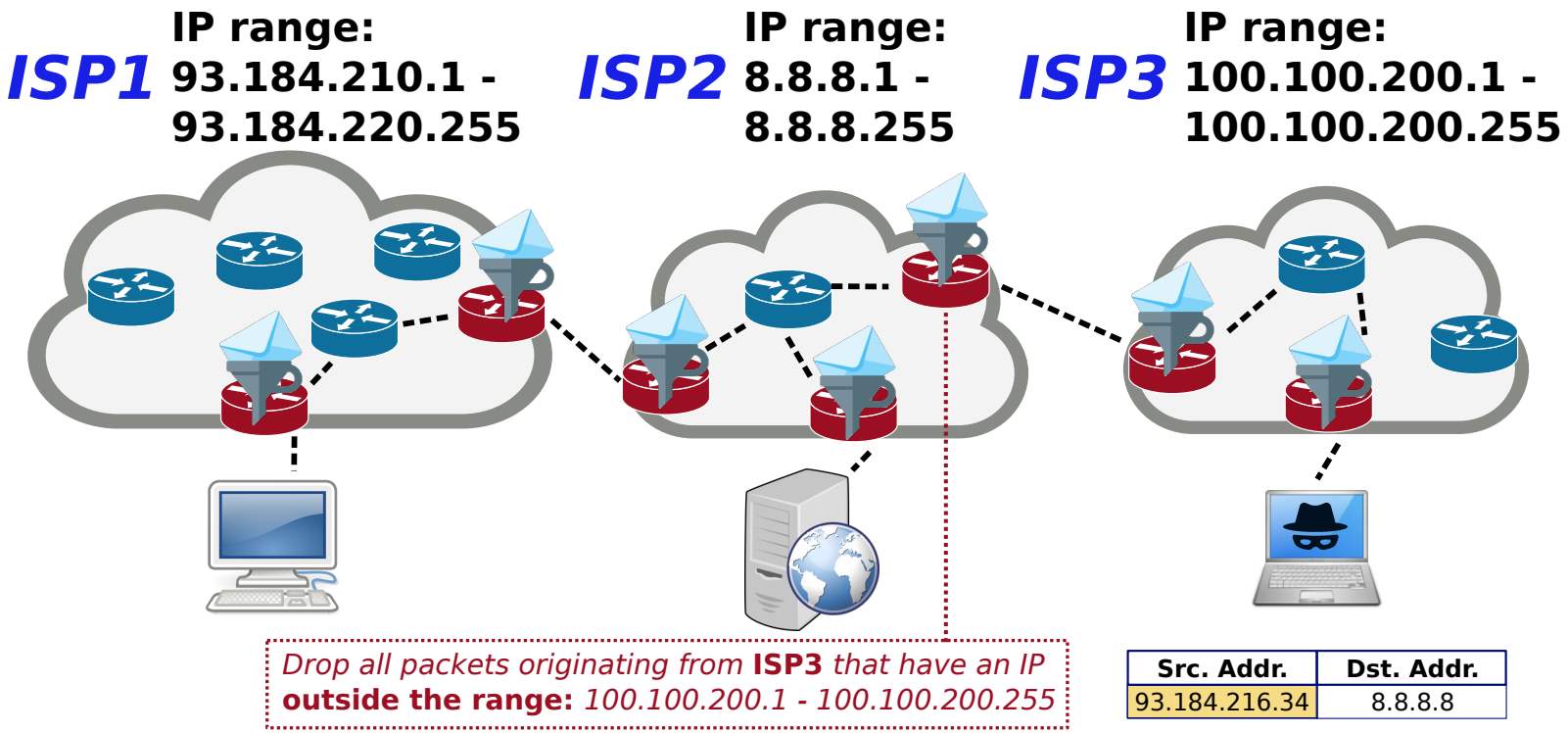
- *"Shoot and forget"* approach:
  - UDP packets
  - TCP packets without connection (e.g., SYN, ACK, RST)
- Tracing the origin of the attack is challenging!



Ingress Filtering

Solution:

ISPs filter incoming packets with spoofed IP addresses RFC 2827 and 3704.



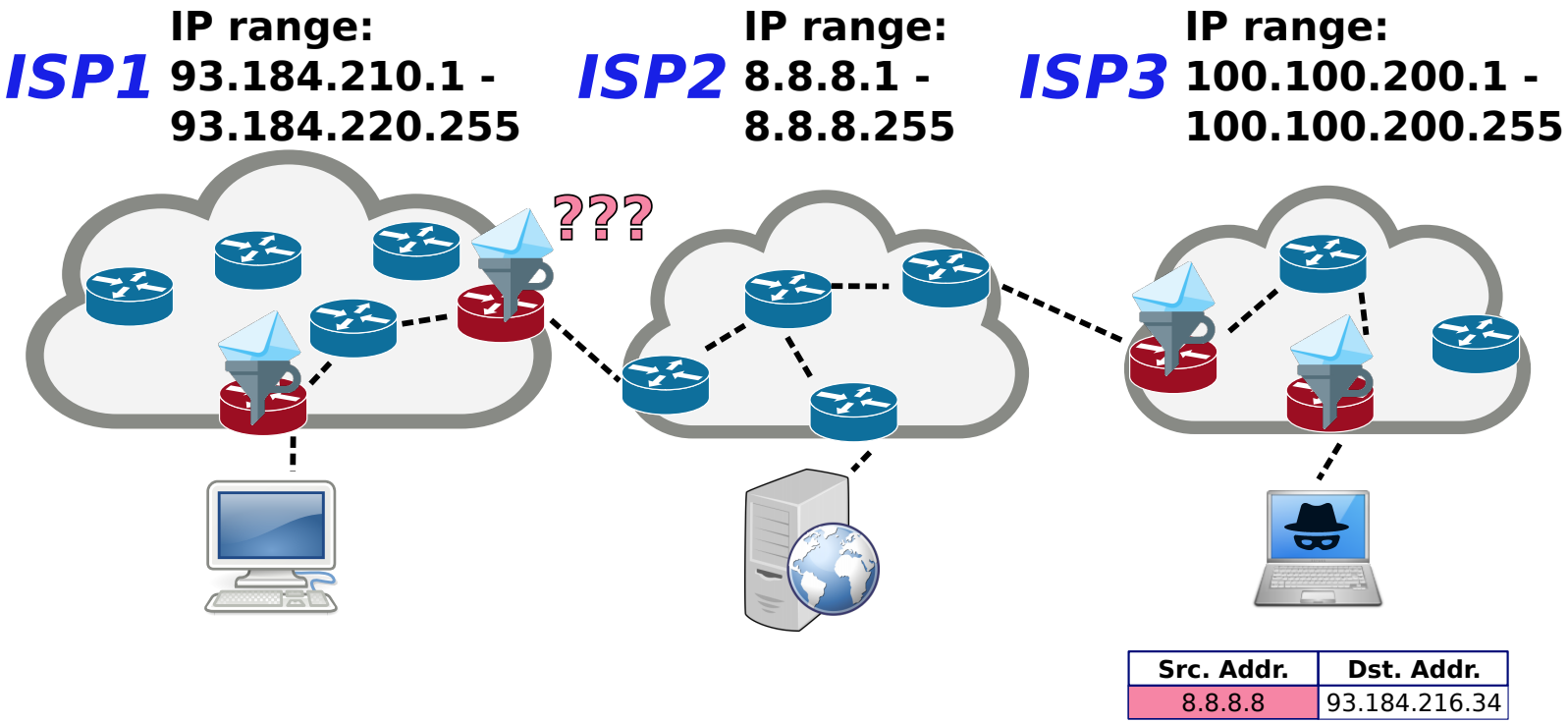
Ingress Filtering

Solution:

ISPs filter incoming packets with spoofed IP addresses RFC 2827 and 3704.

Challenges:

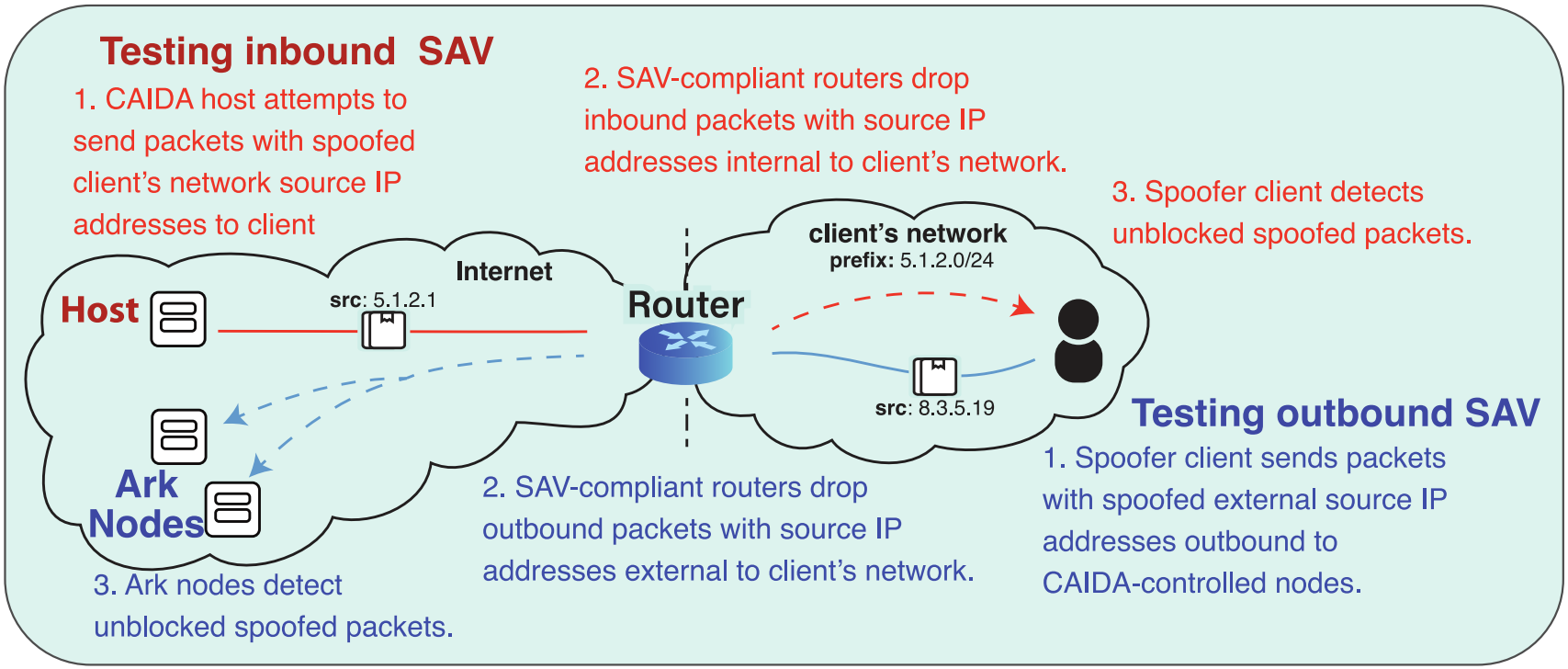
1. The majority/all ISPs must deploy filtering in order to work.
2. Benefits other networks → little incentive for deployment.



Deployment of Filters

Several projects focused on measuring the deployment of ingress filters.

- Spoofer project (started in 2005 at MIT → now CAIDA)
  1. Based on crowd-sourcing → volunteers run tests on their networks;
  2. spoofed packets from servers to clients (ingress filter);
  3. spoofed packets are sent from clients to servers (egress filter).



Source: `https://www.caida.org/projects/spoofer/`

Spoofer Project

- Recent tests: `https://spoofer.caida.org/recent_tests.php`
- Overall statistics: `https://spoofer.caida.org/summary.php`

1. IPv4 results for 2022:
   - 18.5% of Autonomous Systems are spoofable (20.8% in 2022);
     - 8.3% only "partly" (11.5% in 2022);
   - 17.3% of IPv4 block are spoofable (18.9% in 2022)
2. some inconsistencies among ASes when comparing IPv4 and IPv6:
   - 18.5% for IPv4; 25.3% for IPv6 blocks.
3. NATs behave differently[7]:
   - Some may block spoofed traffic.
   - Some do not rewrite addresses outside the NAT's internal prefix.
   - Some rewrite (source address $\leftrightarrow$ public IP) and pass spoofed packets.

[7] Spoofer project on NATs: `https://www.caida.org/projects/spoofer/faq/`

Darknet

Backscatter:

The attacker randomly chooses a spoofed source address and the selected sources will receives answers from the victim. $\Longrightarrow$ **can be leverage to study DoS!**

How: using a darknet *aka Network Telescope*.

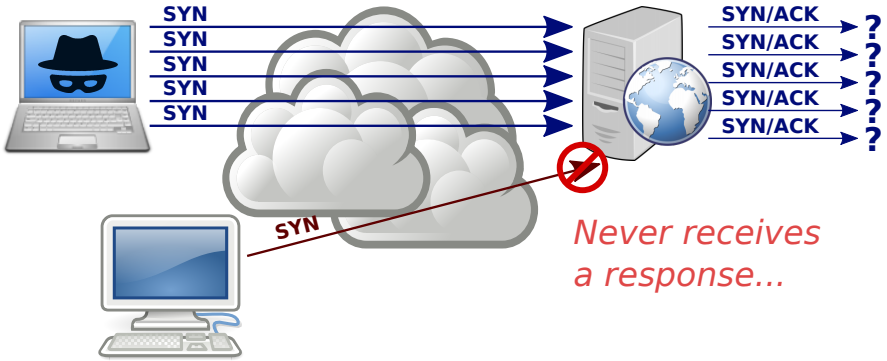
- Monitors unused address space and receives:
  - scans;
  - DoS backscatter.
- If *source* (for **spoofing**) or *destination* (for **scanning**) are chosen randomly... the darknet will receive traffic proportional to its size.
  - CAIDA telescope ($\approx$16M IPs) $\Rightarrow$ 2,000-3,000 attacks/week[8].
- Greynet: mixes used and unused address space.
  - Arbor Networks' Active Threat Level Analysis System (ATLAS) is formed by many ISP greynets[9].

[8] https://www.caida.org/catalog/papers/2006_backscatter_dos/backscatter_dos.pdf
[9] https://www.dlt.com/resources/ddos-attack-protection-arbor-network-s-atlas

SYN Flooding

Issue: SYN received → store all the info (i.e., *state*[10]) for processing segments.

Attacker's goal: fill victim's SYN queue so that it cannot accept other connections.



Features:

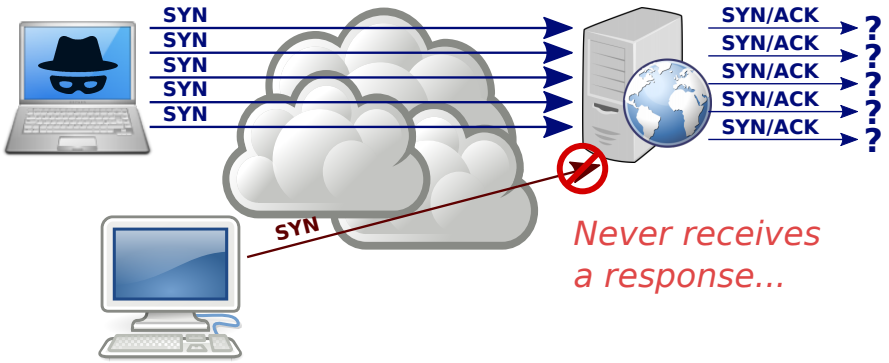
- Protocol state attack:
  - targets TCP state control mechanism;
  - low rate ⇒ no need to flood the network.
- Source address can be spoofed.

[10] RFC with all the details about the data structure used to keep the information for each connection: `https://datatracker.ietf.org/doc/html/rfc2140`

SYN Flooding

Issue: SYN received → store all the info (i.e., *state*[10]) for processing segments.

Attacker's goal: fill victim's SYN queue so that it cannot accept other connections.



Features:

- Protocol state attack:
  - targets TCP state control mechanism;
  - low rate ⇒ no need to flood the network.
- Source address can be spoofed.

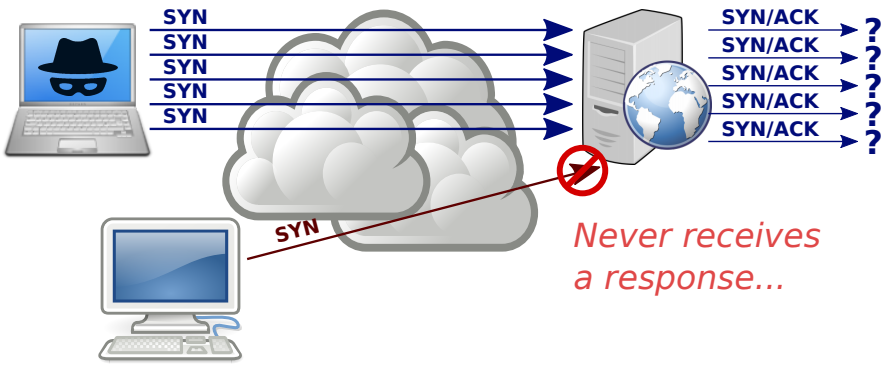
Defenses:

[10] RFC with all the details about the data structure used to keep the information for each connection: `https://datatracker.ietf.org/doc/html/rfc2140`

SYN Flooding

Issue: SYN received → store all the info (i.e., *state*[10]) for processing segments.

Attacker's goal: fill victim's SYN queue so that it cannot accept other connections.



Features:

- Protocol state attack:
  - targets TCP state control mechanism;
  - low rate ⇒ no need to flood the network.
- Source address can be spoofed.

Defenses:

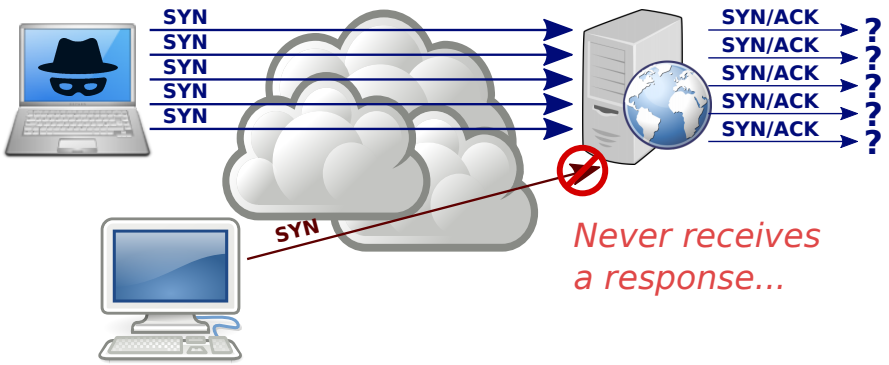
- Increase the size of the queue

[10] RFC with all the details about the data structure used to keep the information for each connection: `https://datatracker.ietf.org/doc/html/rfc2140`

SYN Flooding

Issue: SYN received → store all the info (i.e., *state*[10]) for processing segments.

Attacker's goal: fill victim's SYN queue so that it cannot accept other connections.



Features:

- Protocol state attack:
  - targets TCP state control mechanism;
  - low rate ⇒ no need to flood the network.
- Source address can be spoofed.

Defenses:

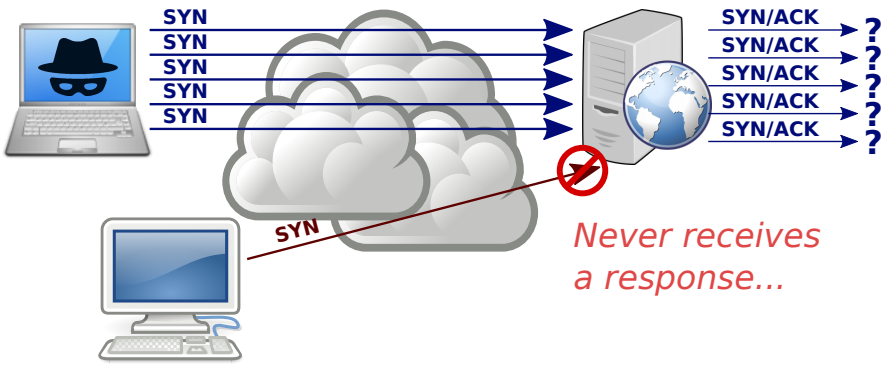
- Increase the size of the queue ⇒ the attacker sends more packets.

[10] RFC with all the details about the data structure used to keep the information for each connection: `https://datatracker.ietf.org/doc/html/rfc2140`

SYN Flooding

Issue: SYN received → store all the info (i.e., *state*[10]) for processing segments.

Attacker's goal: fill victim's SYN queue so that it cannot accept other connections.



Features:

- Protocol state attack:
  - targets TCP state control mechanism;
  - low rate ⇒ no need to flood the network.
- Source address can be spoofed.

Defenses:

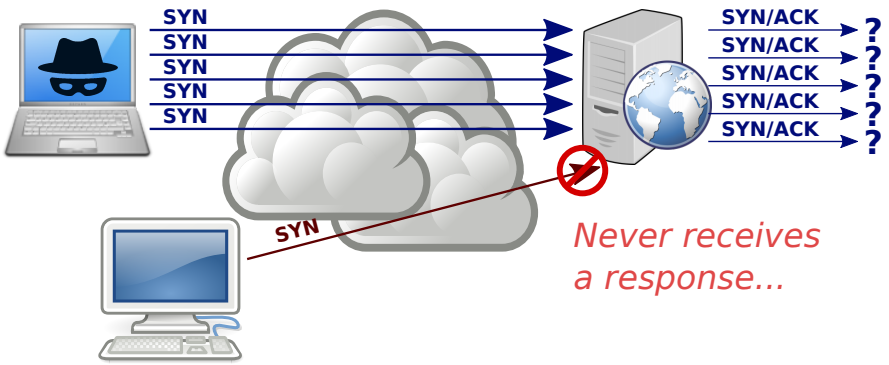
- Increase the size of the queue ⇒ the attacker sends more packets.
- Decrease timeout

[10] RFC with all the details about the data structure used to keep the information for each connection: `https://datatracker.ietf.org/doc/html/rfc2140`

SYN Flooding

Issue: SYN received → store all the info (i.e., *state*[10]) for processing segments.

Attacker's goal: fill victim's SYN queue so that it cannot accept other connections.



Features:

- Protocol state attack:
  - targets TCP state control mechanism;
  - low rate ⇒ no need to flood the network.
- Source address can be spoofed.

Defenses:

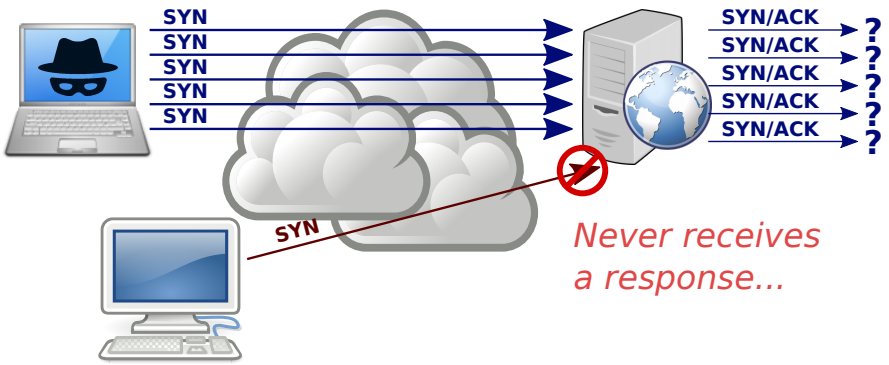
- Increase the size of the queue ⇒ the attacker sends more packets.
- Decrease timeout ⇒ the attacker increases the rate.

[10] RFC with all the details about the data structure used to keep the information for each connection: `https://datatracker.ietf.org/doc/html/rfc2140`

SYN Flooding

Issue: SYN received → store all the info (i.e., *state*[10]) for processing segments.

Attacker's goal: fill victim's SYN queue so that it cannot accept other connections.



Features:

- Protocol state attack:
  - targets TCP state control mechanism;
  - low rate ⇒ no need to flood the network.
- Source address can be spoofed.

~~Defenses:~~ **Poor defenses:**

- Increase the size of the queue ⇒ the attacker sends more packets.
- Decrease timeout ⇒ the attacker increases the rate.

[10] RFC with all the details about the data structure used to keep the information for each connection: `https://datatracker.ietf.org/doc/html/rfc2140`