Computer Networks

4.Transport layer

Boni García
http://bonigarcia.github.io/
boni.garcia@urjc.es

Departamento de Teoría de la Señal y Comunicaciones y Sistemas Telemáticos y Computación
Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

2019/2020

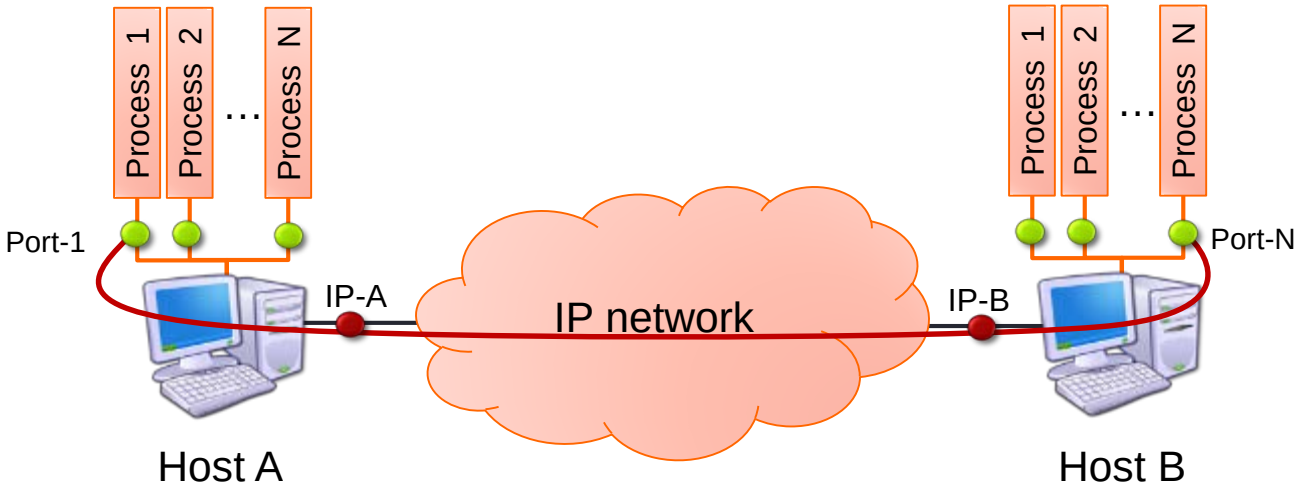
URJC Escuela Técnica Superior de Ingeniería de Telecomunicación

Table of contents

Table of contents

1. Introduction

- **Network layer** provides logical communication between remote **hosts**
- **Transport layer** provides logical communication between applications **processes** running on hosts (*multiplexing*)



1. Introduction - Ports

- We use the concept of **port** to select a given process in a host
- In the transport protocols (both TCP and UDP) we use 16 bits to address ports (i.e. there 2^{16} ports):
  - 0-1023: *Well-known ports*
    - Reserved for common Internet services (e.g. web, email…)
    - Used by system processes (i.e. required root privileged to be used)
  - 1024-49151: User or registered ports
    - Used by regular processes
  - 49152-65535: Dynamic, private, ephemeral ports
    - Used typically by client-side processes

http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

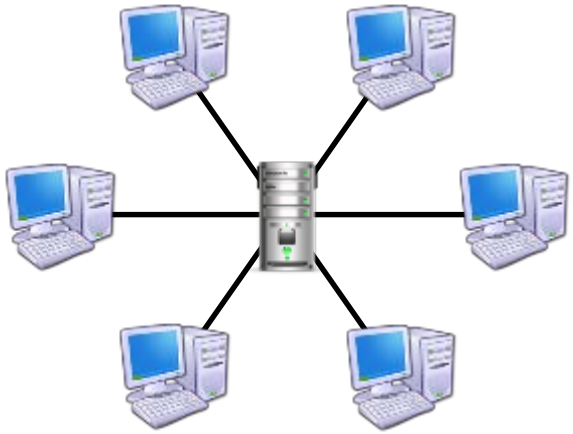
1. Introduction - Transport service

- In the TCP/IP model (Internet), there are two types of transport **protocols**. Each one provides a different type of **services** to the application layer:

1. **UDP** (User Datagram Protocol)
   - Connectionless protocol
   - Unreliable, unordered delivery (best effort)
2. **TCP** (Transmission Control Protocol)
   - Connection oriented protocol
   - Reliable, in-order delivery
   - Congestion and control flow

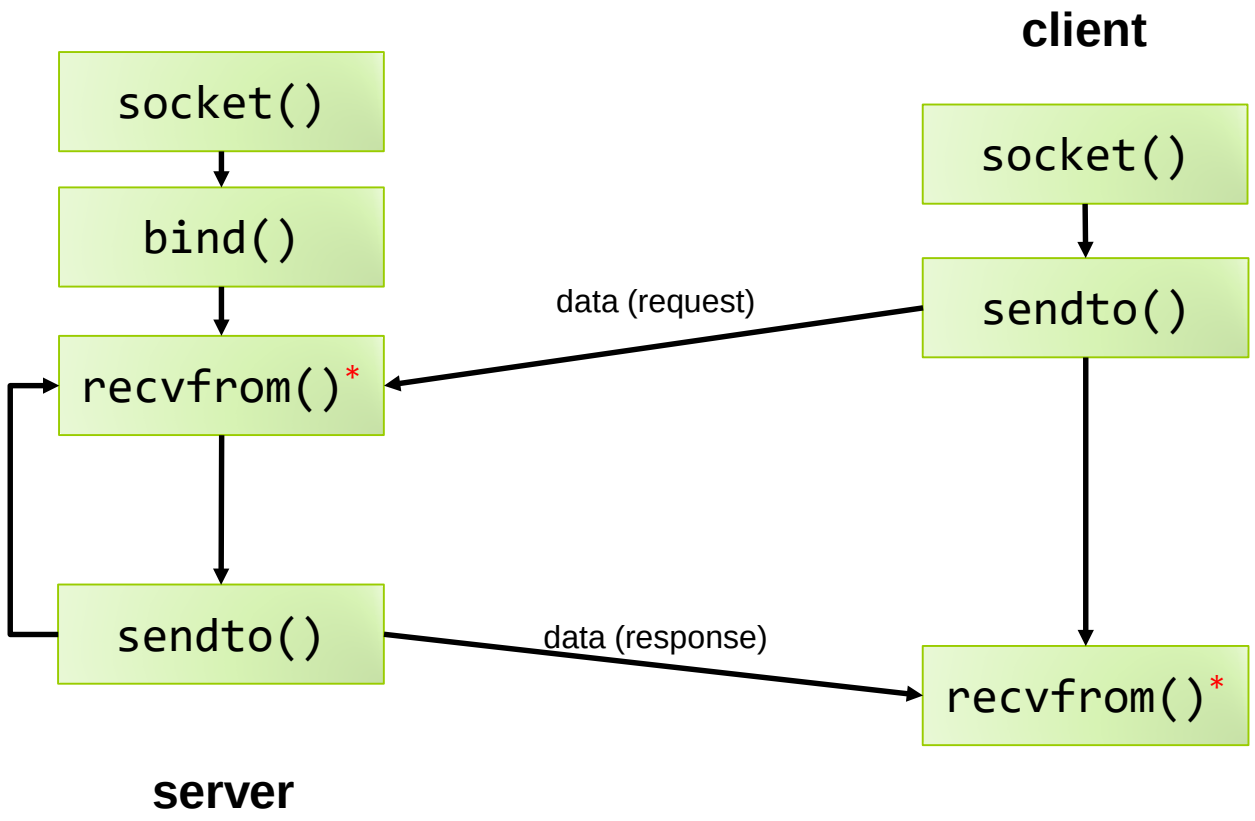
1. Introduction - Client/server model

- The most used architecture for Internet services in the **client/server** model:
  - The server component provides a service to a number of distributed clients
  - Server listen to a given **port**, which must be known by clients
  - Clients send **requests** to servers, and servers send **responses** to those requests



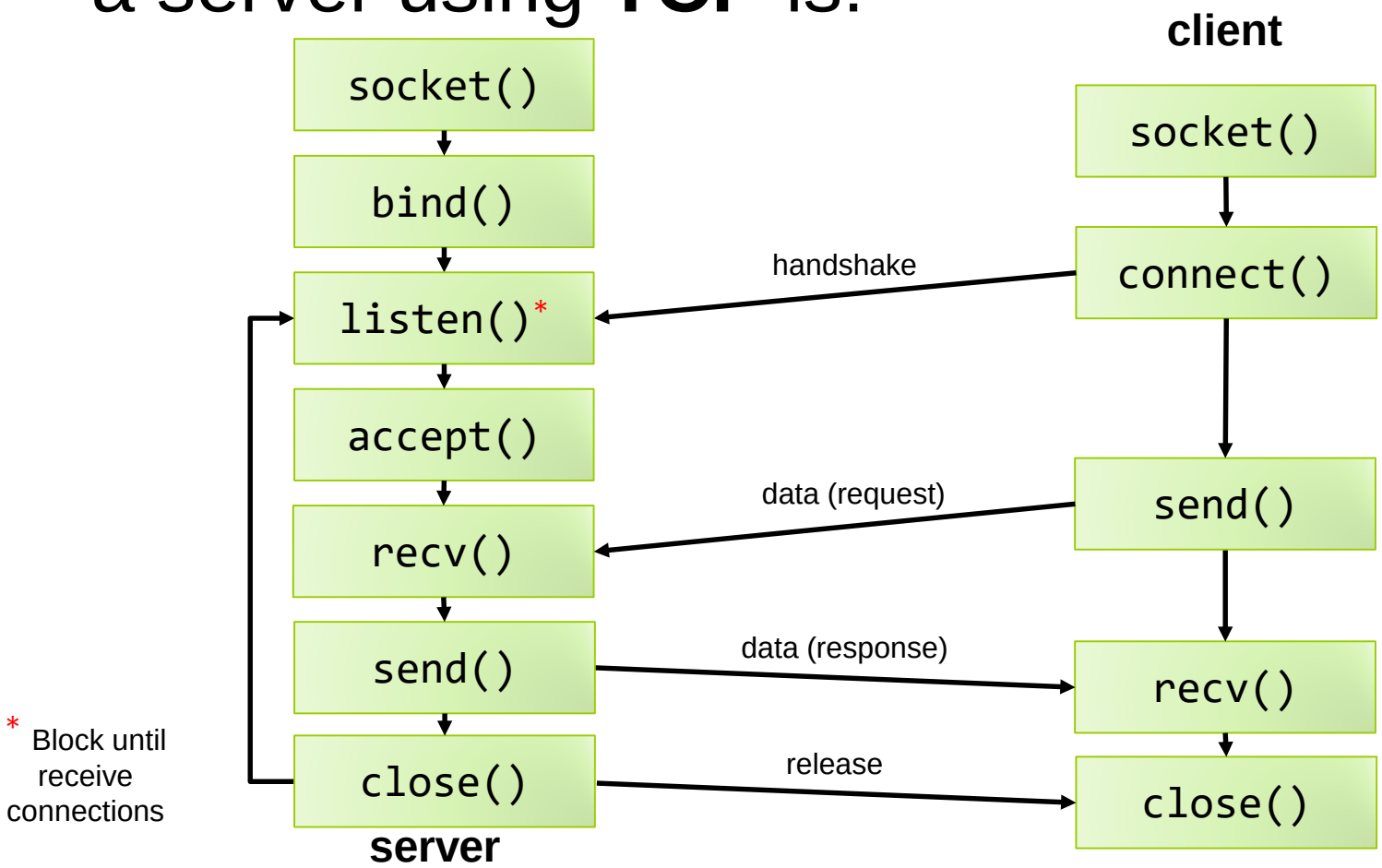
1. Introduction - Client/server model

- The sequence to communicate a client with a server using **UDP** is:



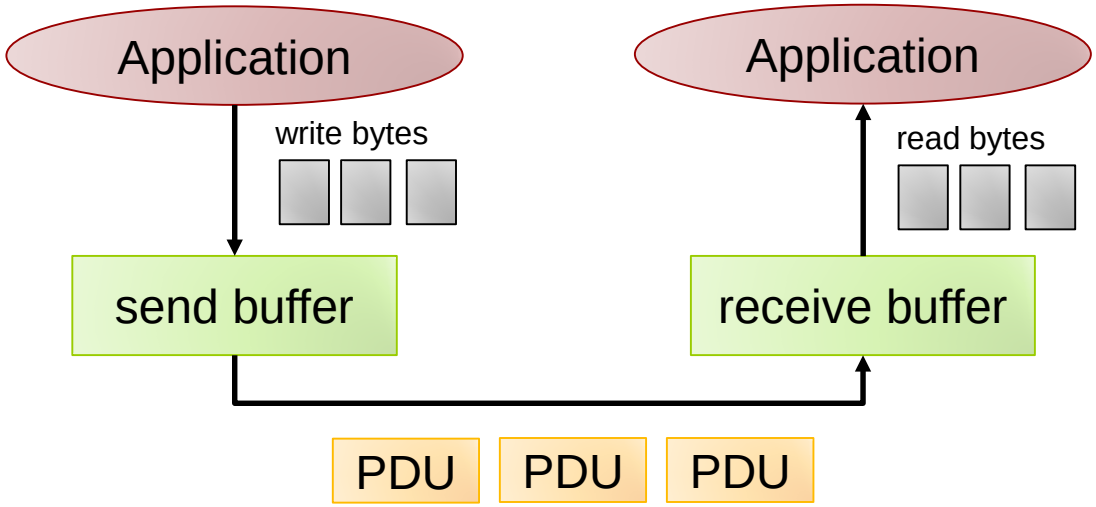
1. Introduction - Client/server model

- The sequence to communicate a client with a server using **TCP** is:



1. Introduction - Buffers

- Transport data (both in UDP and TCP) is allocated in a temporary memory called **buffer**
- Application process read and write bytes from these buffers



1. Introduction - Traffic types

- In networking, there are different types of traffic:

1. Unicast:
   - An entity sends data and other receives
2. Broadcast:
   - An entity sends data and the rest of the network receive
3. Multicast:
   - An entity sends data and a group of rest of the network receive
4. Unicast:
   - An entity sends data and other within a group receive

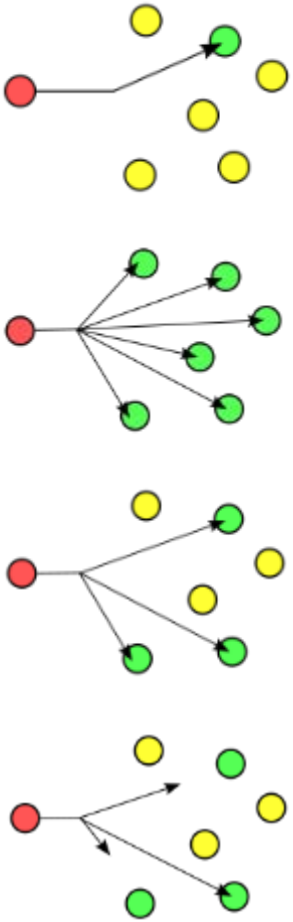


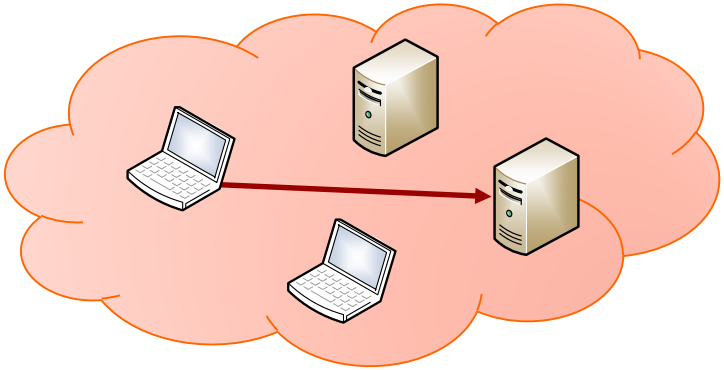
Table of contents

2. UDP - Features

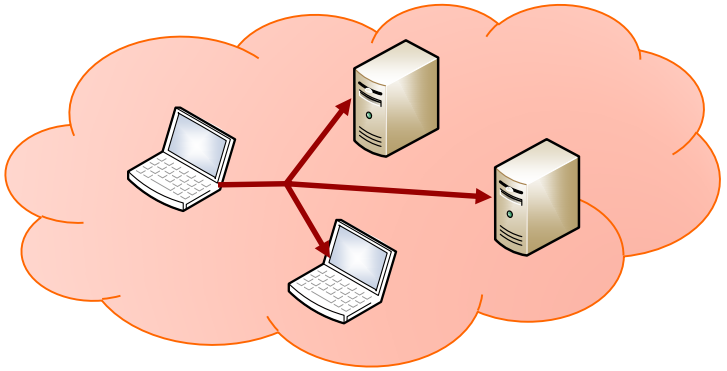
- UDP (User Datagram Protocol) is a transport protocol in the Internet model (RFC 768)
- PDUs in UDP are called **datagrams**
- UDP offers a **best effort** service to applications
  - Unreliable (datagrams can be lost)
  - Unordered delivery (datagrams can be delivered in a different order from they were sent)
- UDP is **connectionless:**
  - No handshaking between UDP sender and receiver
  - Each UDP datagram is handled independently

2. UDP - Features

- UDP is **full-duplex** (sender and receiver can communicate in both directions simultaneously)
- The UDP traffic can be **unicast** or **broadcast** :



Unicast: 1 to 1



Broadcast: 1 to all hosts

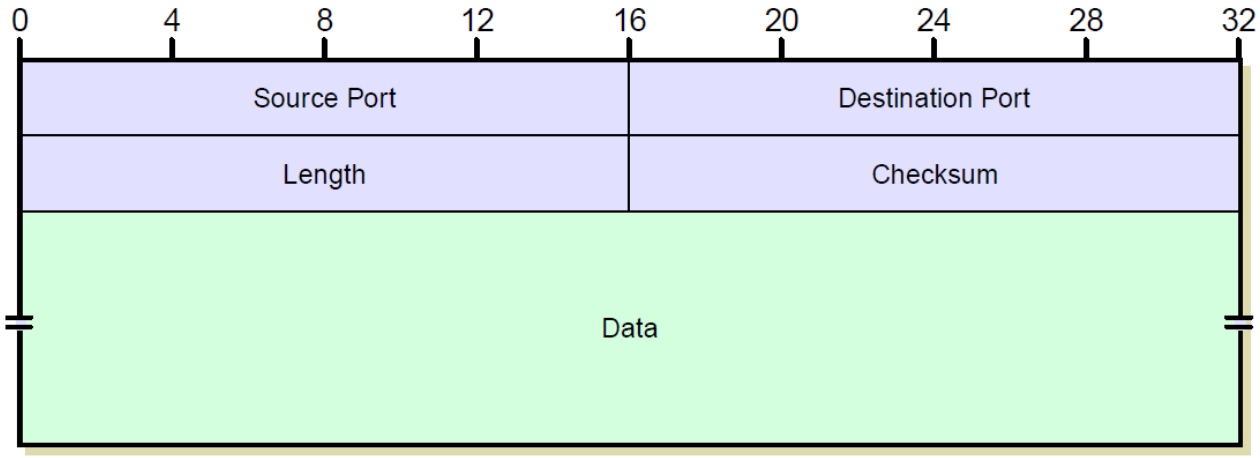
For instance, in DHCP discover

2. UDP - Use cases

- UDP is typically used by applications which requires:
  - Real-time or loss tolerance (e.g. streaming)
  - Broadcasting (since TCP is unicast)
  - Low-latency (since UDP is a very simple protocol, i.e. it is faster that TCP)
- Reliable transfer over UDP is implemented at application level (e.g. DNS)

2. UDP - Datagram format

- The format of a datagram is very simple:
  - Source and destination port (16 bits each)
  - Length (16 bits): Size in bytes of the datagram (header + data)
  - Checksum (16 bits)



2. UDP - Checksum

- Checksum is an **error-detection** method
- In UDP, is a **16-bits** word calculated by the sender of a datagram as the one's complement 16-bits sum of a *pseudo header* from IP, the UDP header, and the data
- Receiver recalculates checksum when a datagram is received. If the calculated checksum is different to the one received, the datagram is discarded

2. UDP - Checksum

- The pseudo header in IP contains: source and destination address, 8 bits to zero, protocol, and packet length
- It is used for historical reasons (TCP/UDP and IP was a single protocol at the beginning)

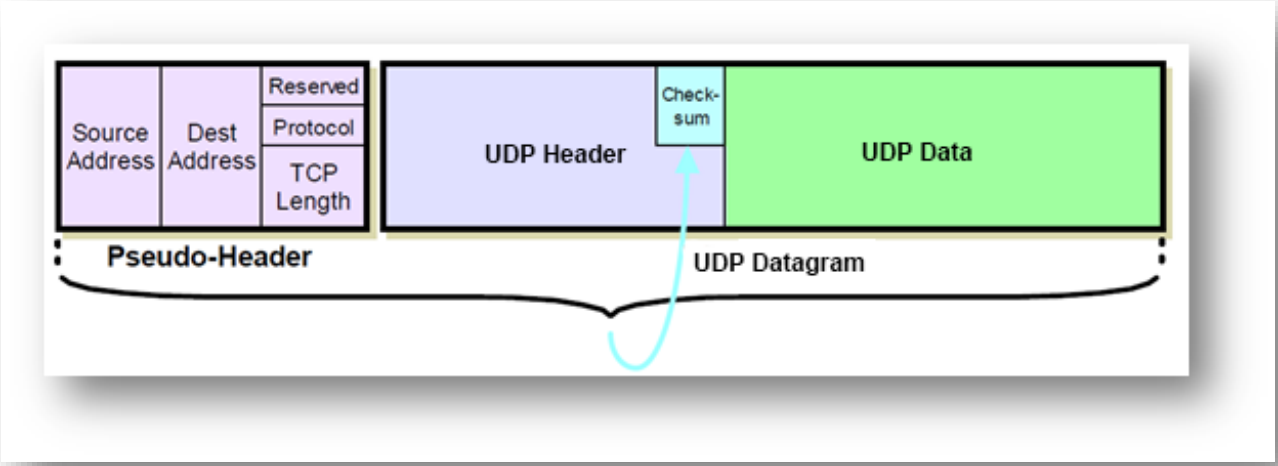


Table of contents

3. TCP - Features

- **TCP** (Transmission Control Protocol) is a transport-layer protocol defined in the RFCs 793 and 1323
- TCP is a **connection oriented** protocol providing a **reliable** and in-order delivery service to applications. Other features:
  - TCP is point to point (unicast traffic)
  - TCP is **full-duplex**
  - It provides **segmentation** (split large messages in smaller parts)
  - It provides **control** and **congestion control**

3. TCP - Segment format

- PDU in TCP is called **segment**, and its format is the following:

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

3. TCP - Segment format

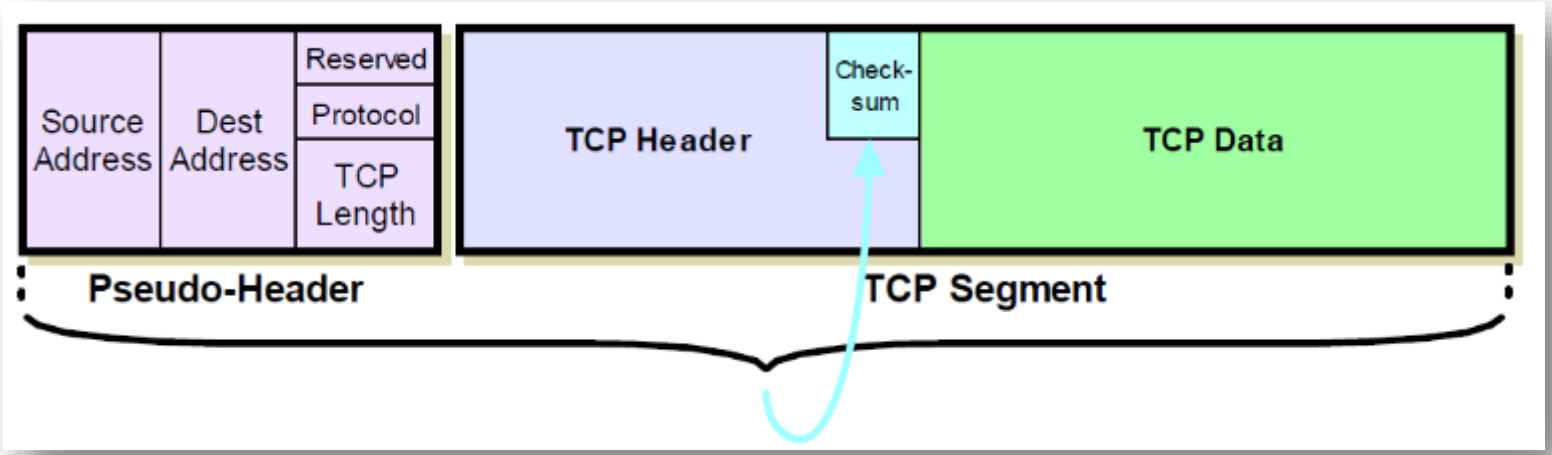
- **Source** and **destination port** (16 bits each)
- **Sequence** (*Seq*) and **acknowledge** (*Ack*) number (32 bits each): TCP is a byte-oriented protocol, which means that in order to provide reliability, in byte sent needs to be sequenced by sender and acknowledged by destination
- **Data offset** (4 bits): size of the TCP header in 32-bit (4 bytes) words. By default this field is 5 (in decimal), which means 20 bytes for the TCP header

3. TCP - Segment format

- **Flags** (9 bits): Control bits used for different purposes. The most important are:
  - **SYN** (1 bit): Do handshake (connection setup)
  - **ACK** (1 bit): Acknowledge data (used in conjunction with the *ack* number)
  - **FIN** (1 bit): Release the connection
  - **RST** (1 bit): Reset the connection (for instance, when a connection cannot be established between the port is not listening)
  - **PSH** (1 bit): Push the buffered data to the receiving application (can be used when data is sent)
  - **URG** (1 bit): Used for urgent data (theoretically used in conjunction with the field “Urgent pointer”)

3. TCP - Segment format

- **Window size** (16 bits): Size of the reception window (*rwnd*) in the control flow
- **Checksum** (16 bits): Calculated in the same way than UDP



3. TCP - Segment format

- Note about the payload (data) length:
  - TCP does not provide a field to set the length of the payload (only the length of header)
  - To calculate it, the length of the IP header and IP payload is used

- **Reserved** (3 bits): Not used, it could be used for future features (now it is sent to 000)

3. TCP - Segment format

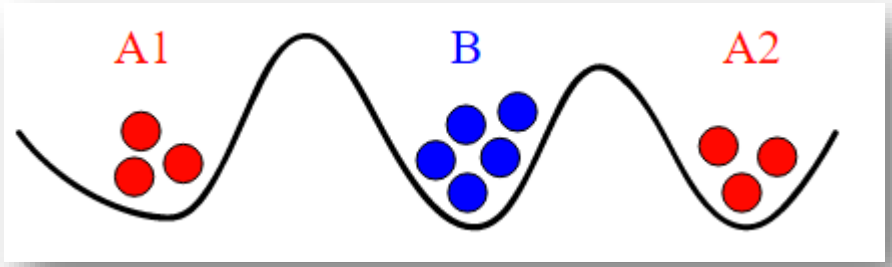
- **Urgent pointer** (16 bits): Pointer to urgent data. Due to an inconsistency between RFCs 793 and 1122, this field is not used nowadays in Internet
- **Options** (multiple of 4 bytes): Extra features for TCP, for instance:
  - Timestamps (TSval,TSecr)
  - Setup for the Maximum Segment Size (MSS)
  - …
  - To achieve options multiple of 4 bytes, padding of *no-operation option* (NOP) can be included

3. TCP - Connection handshake

- TCP is a connection oriented protocol, and so, it has three well defined stages:
  1. Connection handshake (setup)
  2. Data transfer
  3. Connection release
- Since TCP always use IP as network protocol (i.e. non reliable channel), ensuring the handshake is equivalent to the “Two Generals' Problem”

3. TCP - Connection handshake

- Two Generals' Problem:
  - Armies A1 and A2 want to attack to their enemy (B) which is placed in between
  - To success in the attack, A1 and A2 must attack at the same time
  - To coordinate the attack, A1 and A2 send messengers which cross the enemy lines
  - Messengers and can be captured, and so, acknowledgements are used (another messenger)
  - Each acknowledgement of message receipt can be also lost

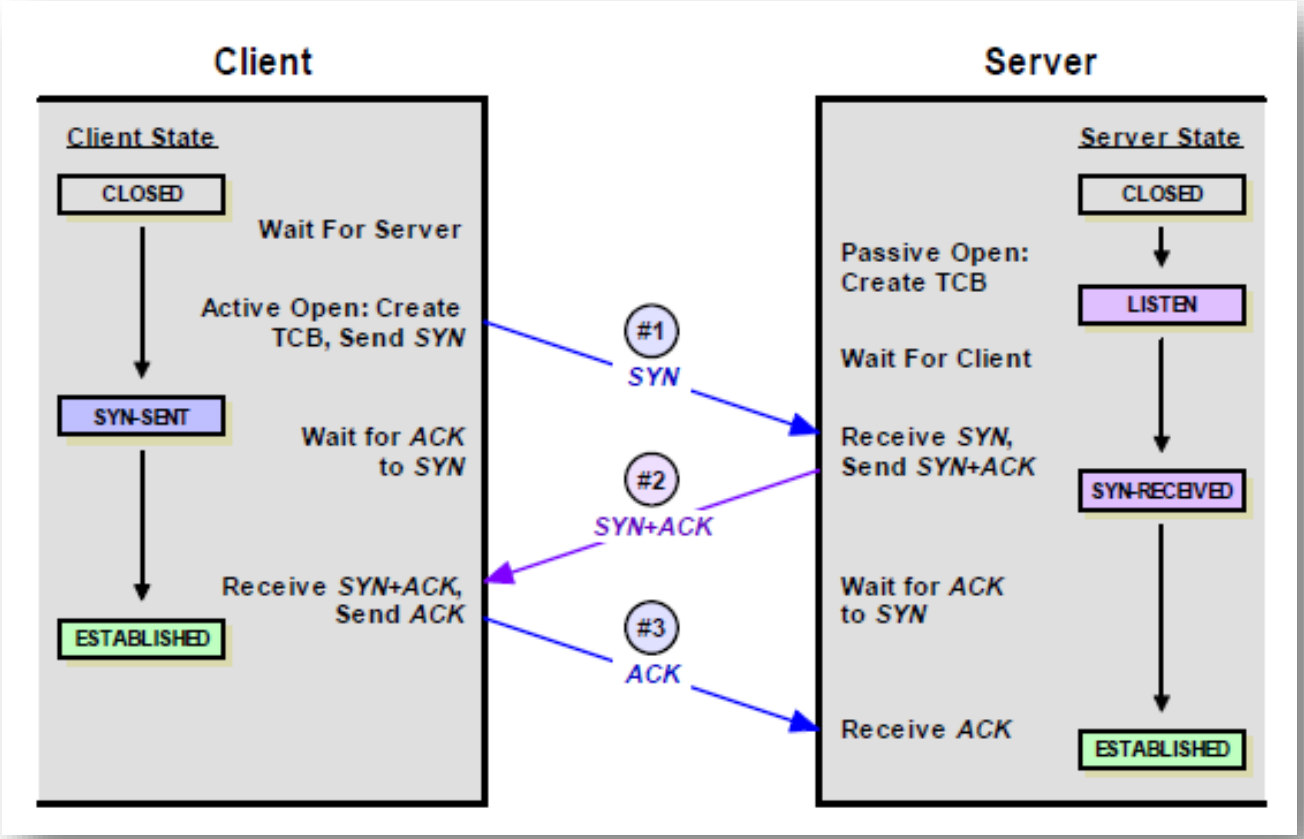


3. TCP - Connection handshake

- The problem of the two generals cannot be solved with 100% confidence (the last messenger can be captured or not)
- The same happens with TCP over IP, since IP's channels are not reliable
- For that reason, TCP uses a compromise solution: the **3-way handshake**

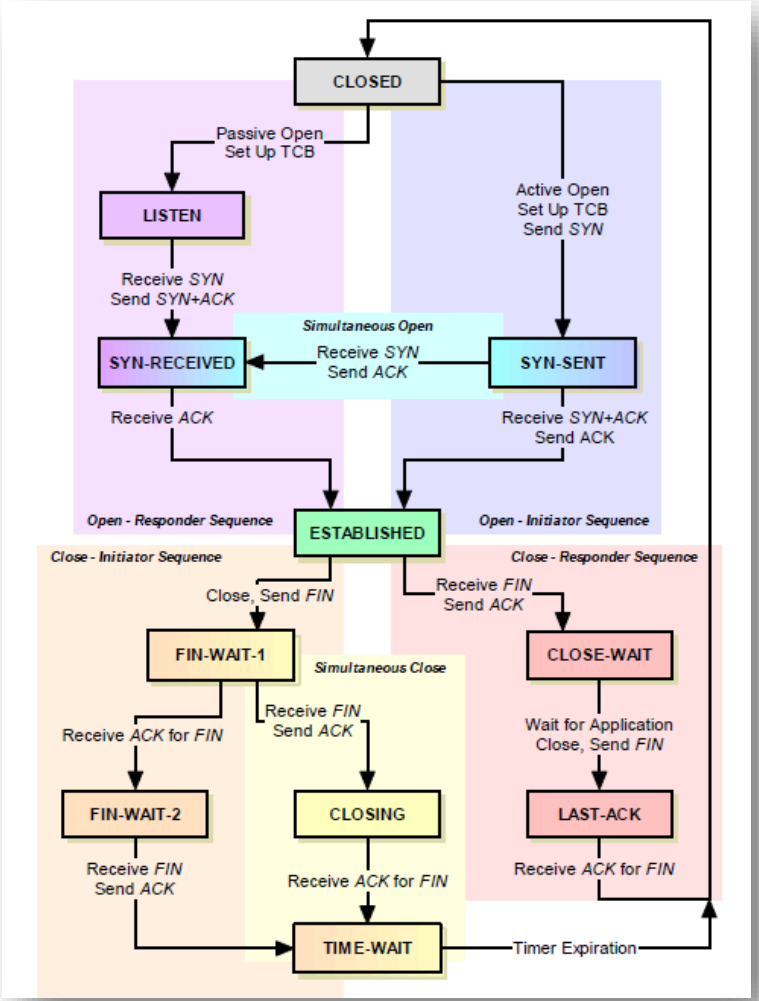
3. TCP - Connection handshake

- 3-way handshake:



3. TCP - Connection handshake

- Each TCP entity implements an FSM (Finite State Machine)
- There are timeouts between some of these states to detect error situations

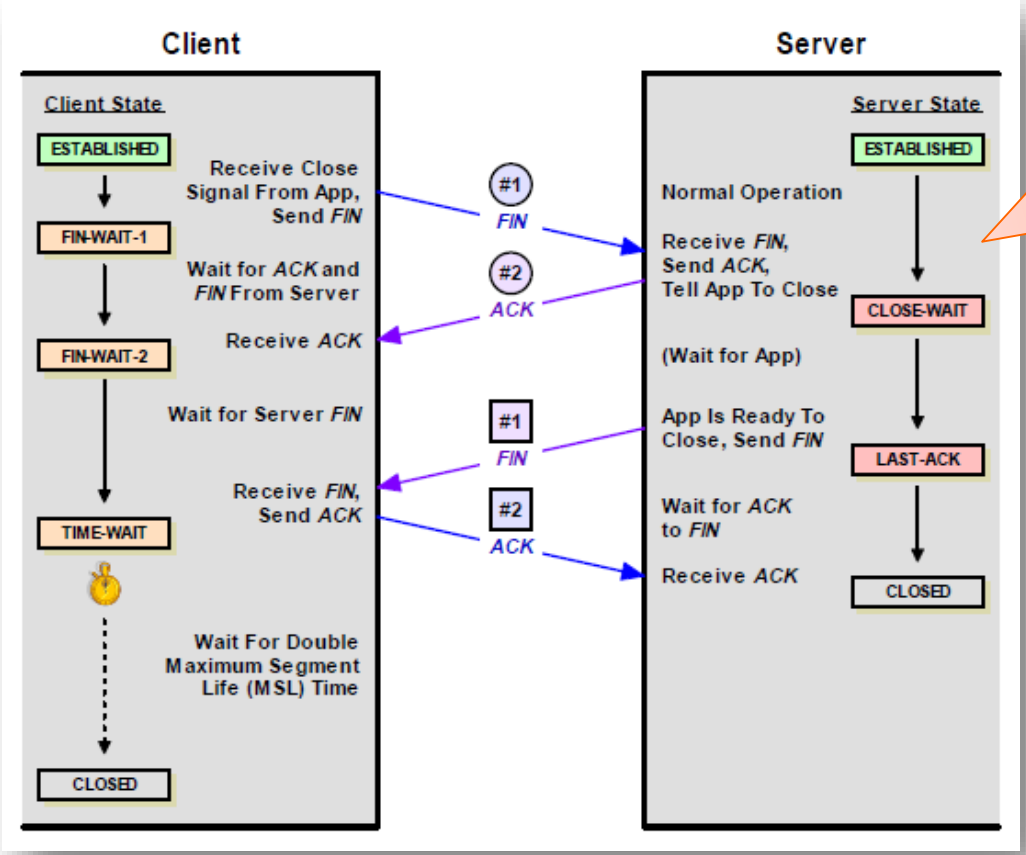


3. TCP - Connection release

- The connection release follows a **4-way handshake** (which can be done using only 3 segments if ACK and FIN from server to client is sent using only 1 segment)
  - TCP defines a wait time after the TCP is released
    - $t_{TIME\text{-}WAIT}$ = 2* MSL = 4 minutes
  - MSL (Maximum Segment Life) = 2 minutes (maximum life time for a given)
  - This time is a guard to avoid collisions with incoming connections

3. TCP - Connection release

- **4-way handshake** release:



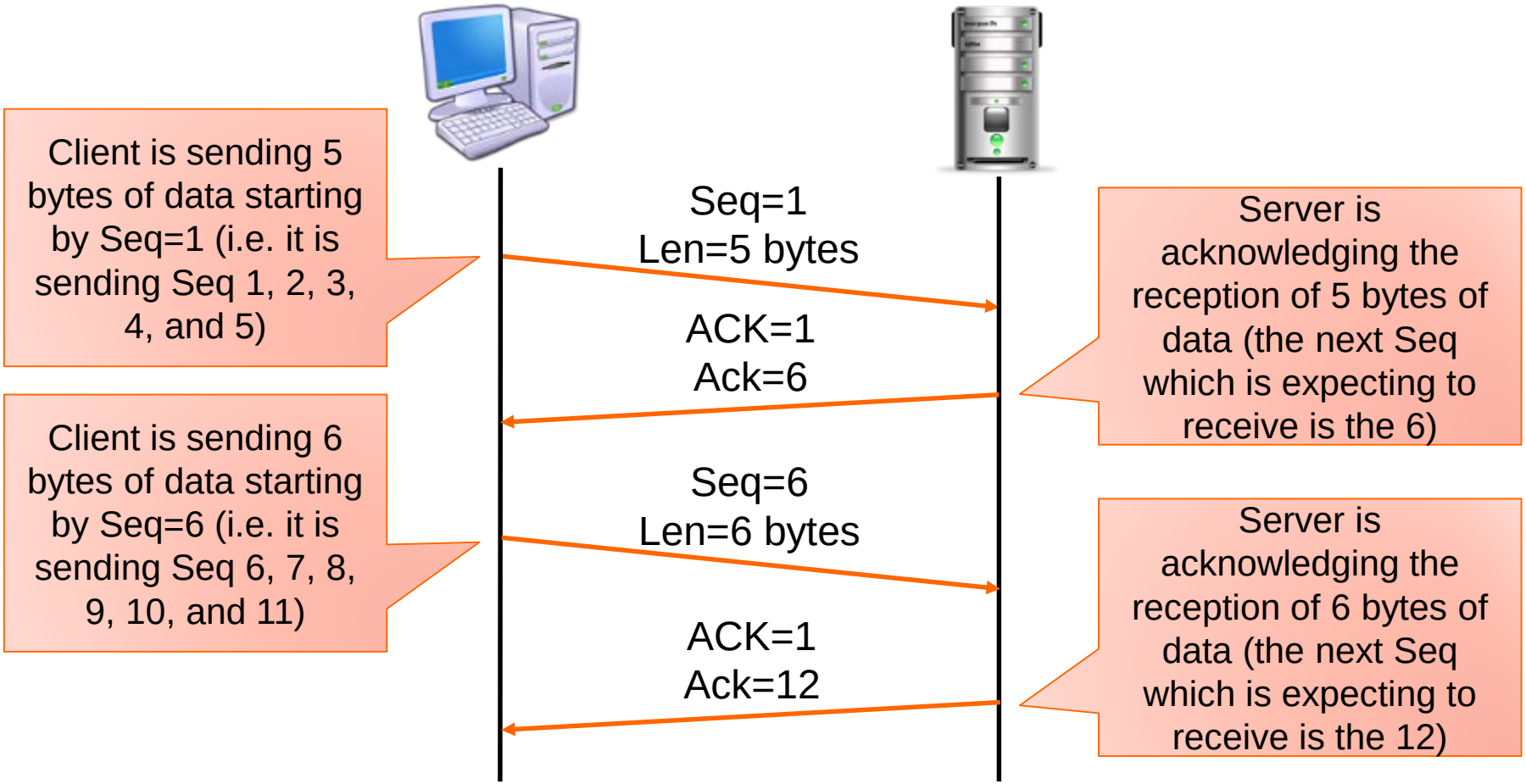
ACK and FIN from server to client can be sent using only 1 segment

3. TCP - Data transfer

- TCP is a **reliable** byte-oriented transport protocol
- A TCP entity sending data must sequence the bytes sent using the **sequence number** (*Seq*)
- These bytes must be acknowledged by the receiver using the **acknowledge number** (*Ack*) and setting the flag ACK to 1
  - The *Ack* number is the sequence number of the next byte the receiver expects to receive (receiver acknowledges the receipt of all data bytes but not including byte number *Ack*)

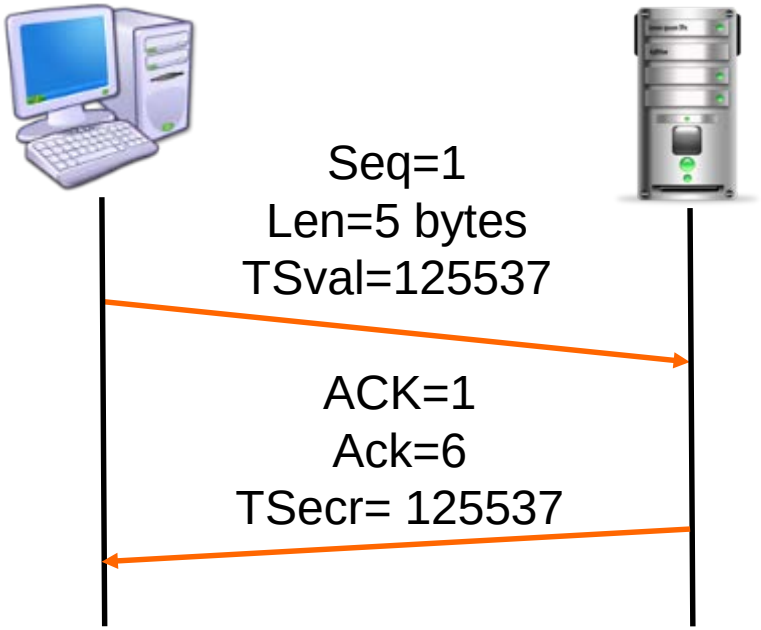
3. TCP - Data transfer

- Example: data sent and received correctly:



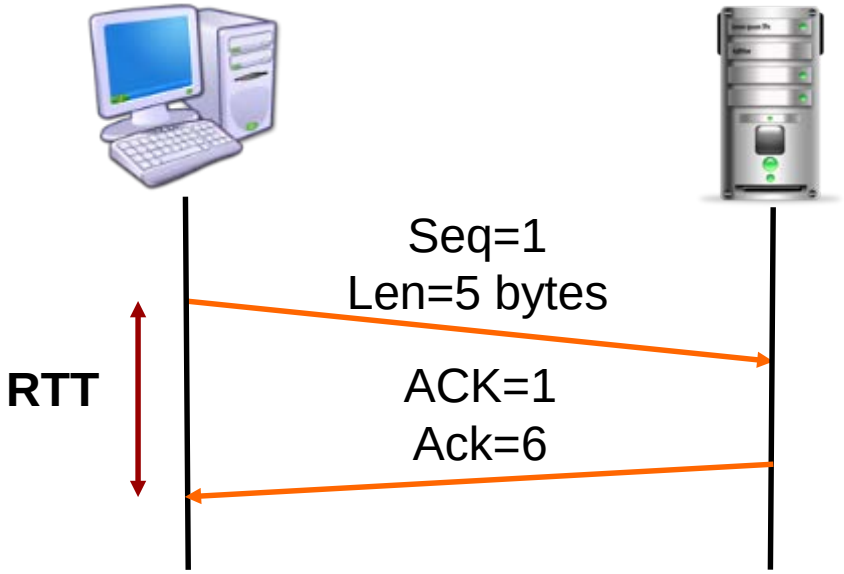
3. TCP - Data transfer

- To match the data sent (Seq number) and its acknowledgement (Ack number), TCP uses the timestamp option:
  - TSval: timestamp value
  - TSecr: timestamp echo reply (match acknowledged TSval)



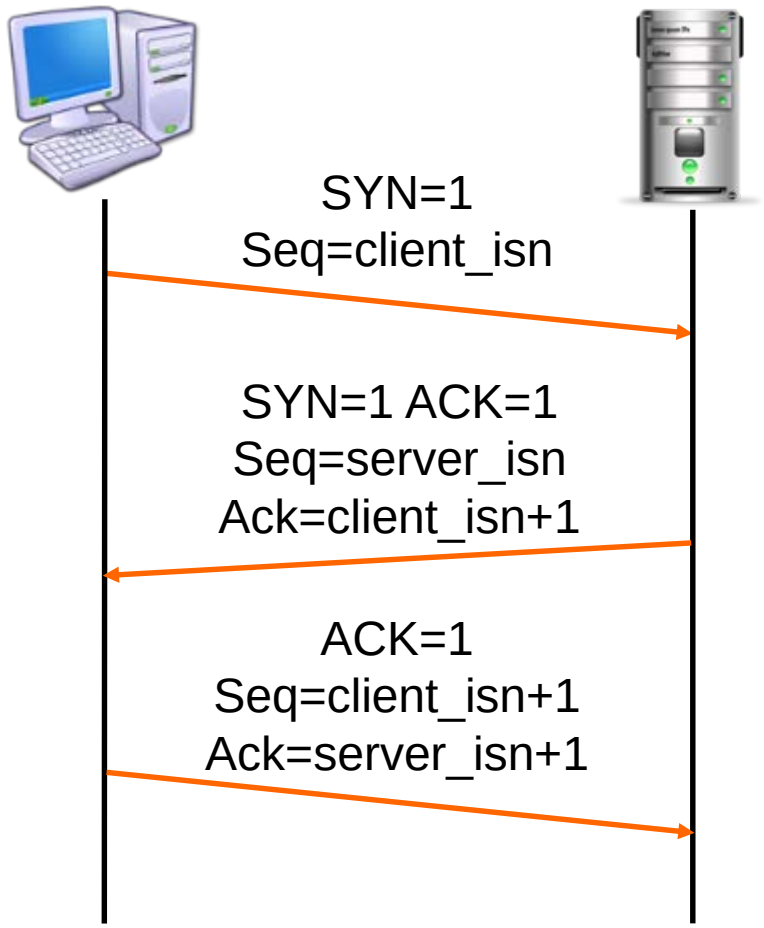
3. TCP - Data transfer

- For each segment with data and acknowledged, we can calculate its **RTT** (Round Trip Time)



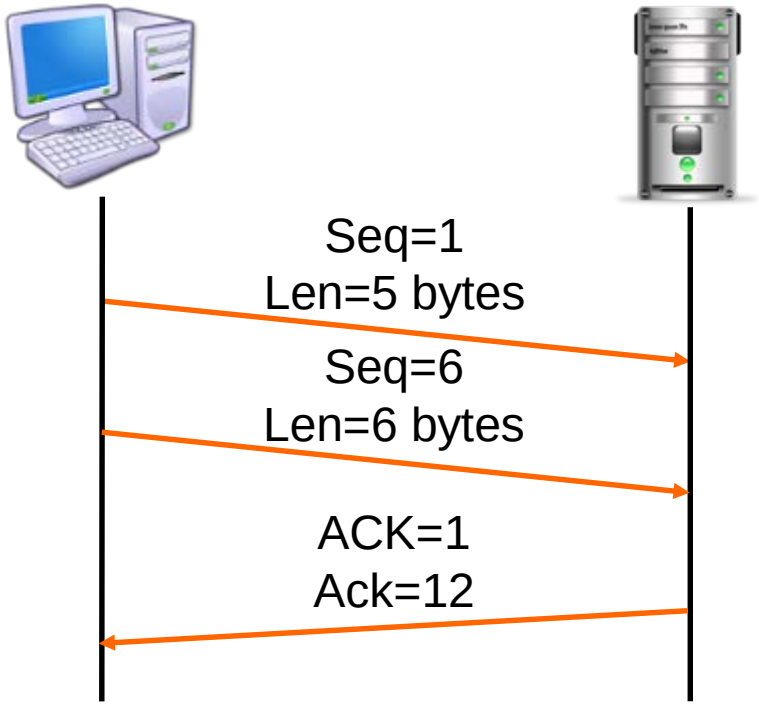
3. TCP - Data transfer

- The Initial Sequence Number (**ISN**) is established in the connection handshake (SYN=1)
- This number is calculated randomly:
  - To avoid mixing different segments of different connections
  - To avoid attacks for prediction of TCP sequence (e.g. Mitnick attack)



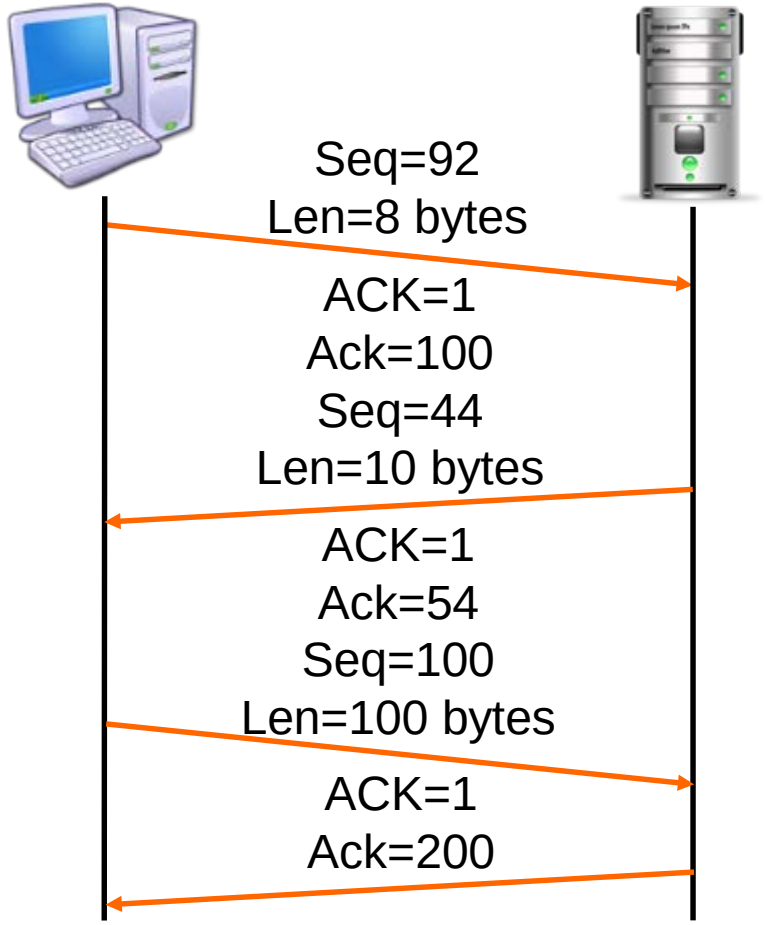
3. TCP - Data transfer

- The most common acknowledgment schema in TCP is **cumulative ACK** (a single acknowledgement is used to response to a number of segments previously received)



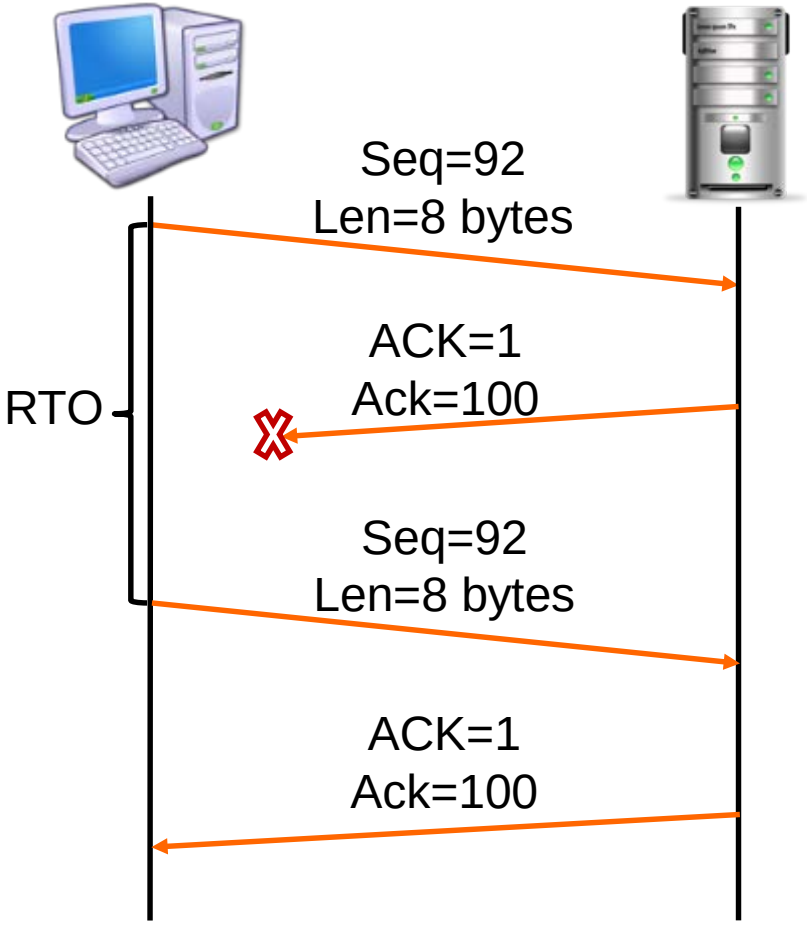
3. TCP - Data transfer

- Data can be sent and acknowledged in both directions of the connection simultaneously (this is known as ***piggybacking***)



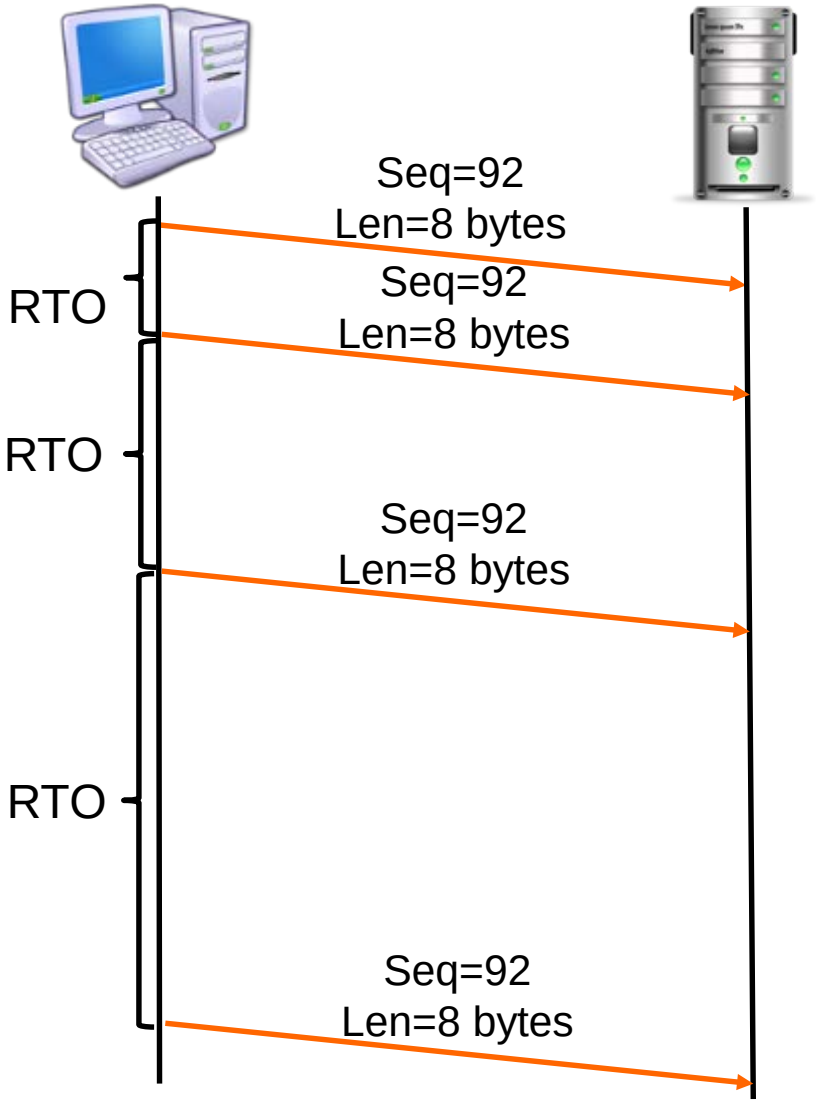
3. TCP - Retransmissions

- Segment loss are detected using timeouts
- RTO (Retransmission Timeout) is calculated as a function of the estimated RTT



3. TCP - Retransmissions

- RTO is doubled (plus a safety margin) each time a retransmission is performed
- This mechanism is known as **exponential backoff**



3. TCP - Maximum segment size

- Maximum Segment Size (**MSS**) is the value used to define the maximum segment that will be used during a connection
- The objective of using MSS is to avoid IP fragmentation
- If not established during handshake (with options), MSS is calculated using the value of MTU (Maximum Transfer Unit) of link layer

```
MSS = MTU – TCP_header_length – IP_header_length
```

3. TCP - Maximum segment size

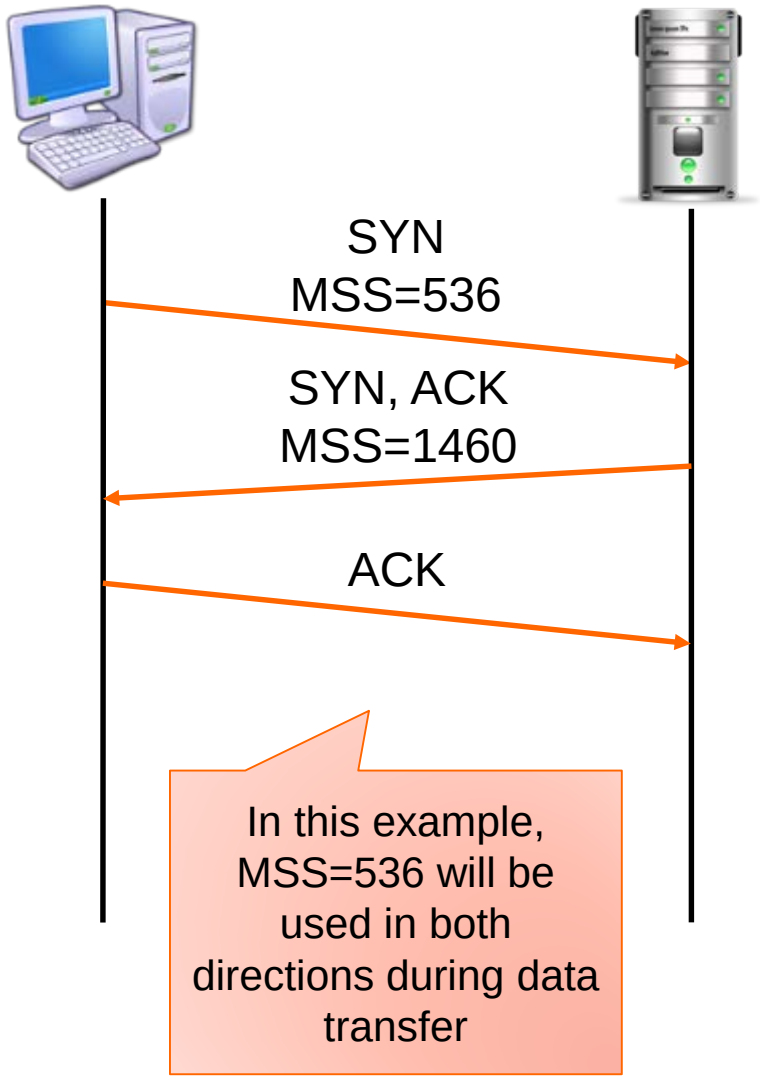
- Maximum Segment Size (**MSS**) is the value used to specify the largest amount of bytes of the TCP payload in a single segment
  - The objective of using MSS is to avoid IP fragmentation
- If not established during handshake (with options), MSS is calculated using the value of MTU (Maximum Transfer Unit) of link layer

```
MSS = MTU – default_TCP_header_length – default_IP_header_length
```

- A TCP sender adjusts the TCP payload length according to the MSS value less the length of IP or TCP options (RFC 6691)

3. TCP - Maximum segment size

- MSS can be negotiated during connection handshake using options
- When both client and server propose a MSS value in the handshake, the **smaller** value will be used during the data transfer



3. TCP - Maximum segment size

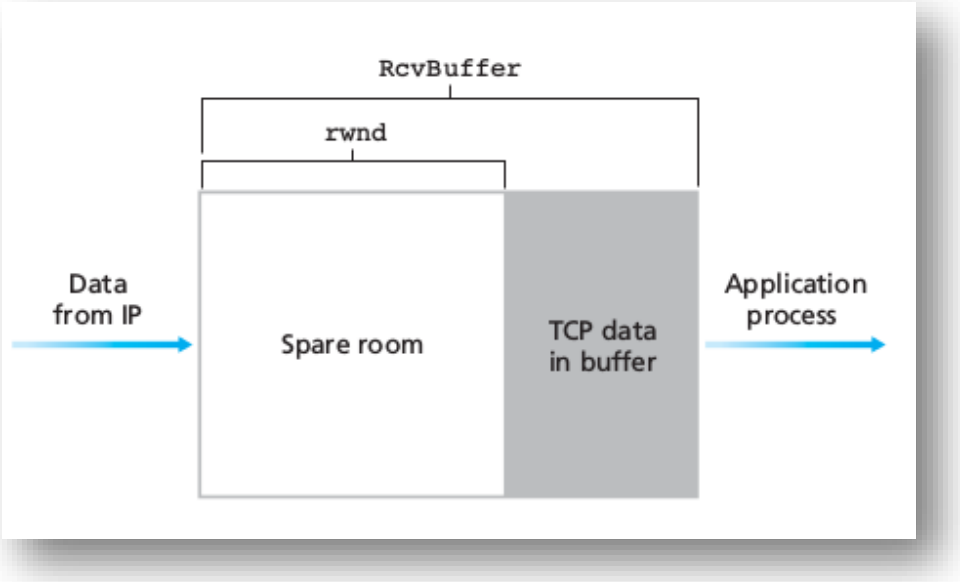
- Fragmentation may occur if an intermediate router supports a MTU smaller than the negotiated MSS
- To avoid this, **Path MTU Discovery** is used:
  - It works by setting the Don't Fragment (DF) flag in the IP headers of outgoing packets
  - Any intermediate router whose MTU is smaller than the packet will drop it, sending back an ICMP error message (*Fragmentation Needed*, Type 3, Code 4) together with **MTU of the next hop**
  - This value is used to calculate a new MSS

3. TCP - Flow control

- TCP **flow control** is a mechanism which aims to ensure that a sender is not overwhelming a receiver by sending packets faster than it can consume
- TCP uses a **sliding window** for implementing flow control
  - Each entity defines a variable called reception window (*rwnd*)
  - This value is sent in each TCP segment (*Win*) and means the maximum number of bytes which an entity is able to receive

3. TCP - Flow control

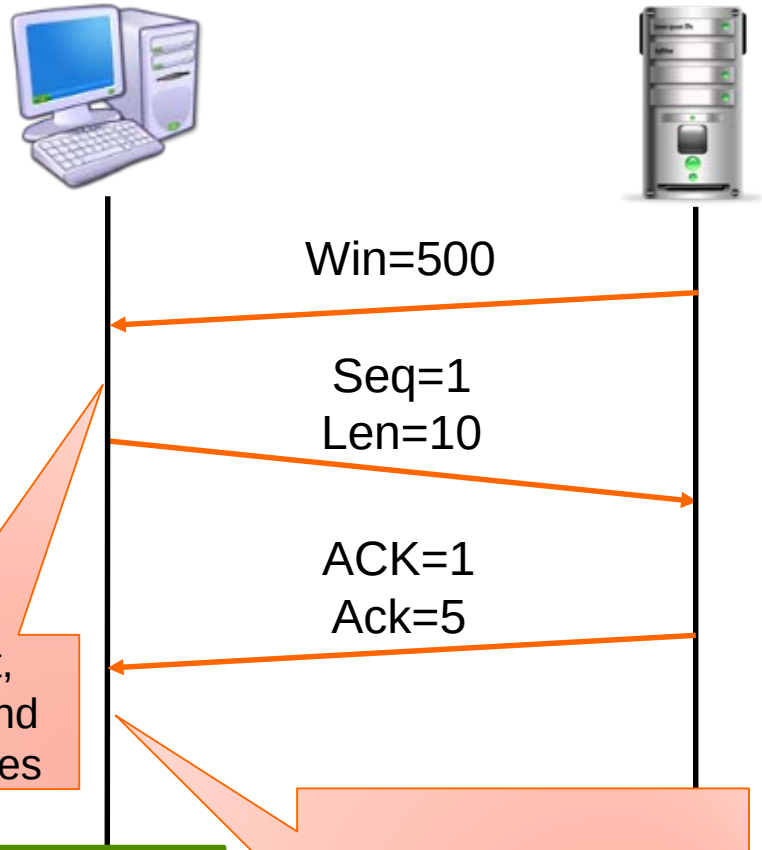
- It is called sliding window because:
  - The window closes when bytes are acknowledged
  - The window opens when application process bytes




```
rwnd = RcvBuffer - [LastByteRcvd - LastByteRead]
```

3. TCP - Flow control

- A TCP entity can send new bytes depending of the advertised reception window (*Win*) less the number of bytes sent but not acknowledged




```
new_data_length = win – bytes_not_ack =
win – (last_seq_number – last_ack_number) =
win – [(last_seq + last_len – 1) – (last_ ack - 1)] =
win – (last_seq + last_len – last_ack)
```

3. TCP - Flow control

- A receiver TCP entity can reduce the *rwnd* value to zero bytes if it is unable to pass data to app layer: it is said to have closed the window
- In this case the sender cannot send further data segments until the received re-opens the window
- The sender may, however, send window probe segments to check the status of the window (*TCP Keep Alive*)
- These special window probes are sent periodically, with data length equal to zero bytes, until the receiver is able to accept data again

3. TCP - Congestion control

- Congestion control is a mechanisms aimed to prevent **network congestion**
- In TCP, this mechanism is implemented using a the congestion window (*cwnd*), which is a value used by the source to limit how much data it is allowed to have in transit at a given time

$$\texttt{LastByteSent - LastByteAcked} \leq \texttt{min\{cwnd, rwnd\}}$$

Number of bytes sent and not acknowledged

Supposing rwnd is big (no reception problem, normal case), the data flow is determined only by the value of cwnd

3. TCP - Congestion control

- The value of *cwnd* varies following an algorithm made up by 3 steps (RFC 2581)
  1. Slow start
  2. Congestion avoidance
  3. Congestion detection

3. TCP - Congestion control

1. Slow start
   - At the beginning, segments are sent with few data
   - With each ACK, *cwnd* doubles its value
2. Congestion avoidance
   - Once *cwnd* reaches a given threshold (SST), the behavior of *cwnd* is linear
3. Congestion detection
   - At any time, congestion can be detected (e.g. timeout in the reception of an ACK)
   - In this case, SST is reduced and the algorithm starts over again

3. TCP - Congestion control

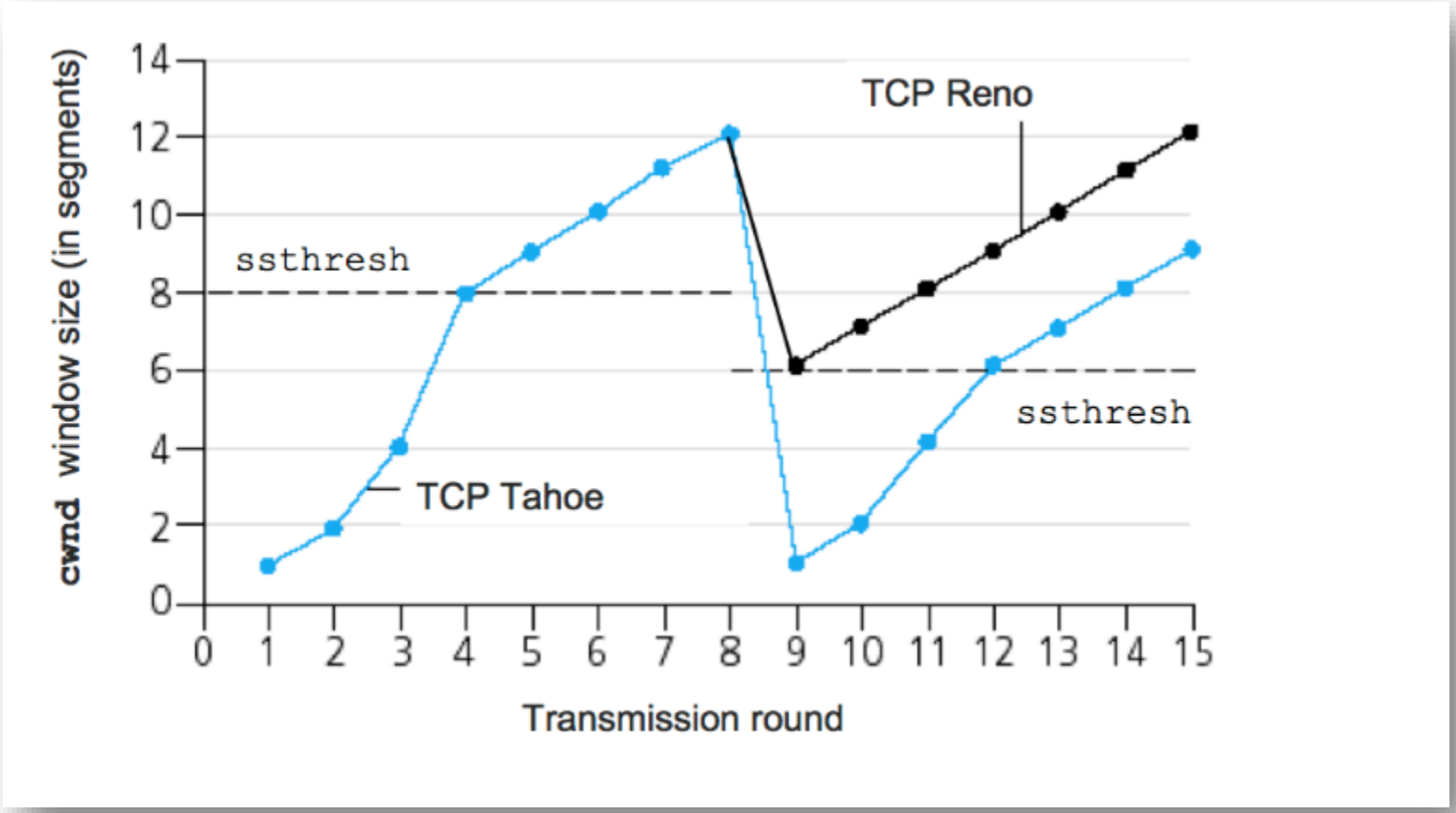


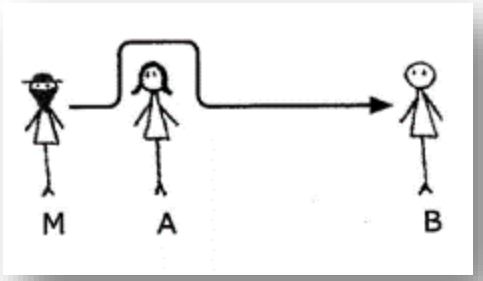
Table of contents

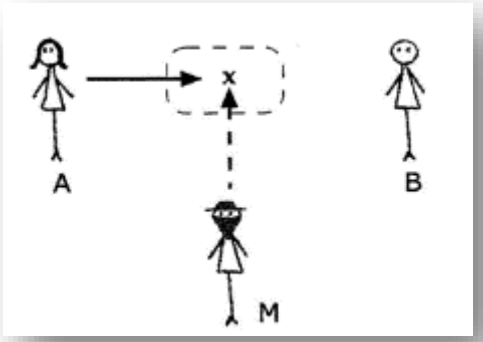
4. Security - Concepts

- **Security** in computer networks are a set of techniques that aim to minimize vulnerabilities
  - The objective is to ensure that the cost of breaking the security is greater than the resource that is intended to protect
  - There is no total security: for any protection, you can always find an element capable of breaking it
- An **attack** is an unauthorized actions to a given resource by a malicious subject to destroy, modify or steal sensitive data

4. Security - Concepts

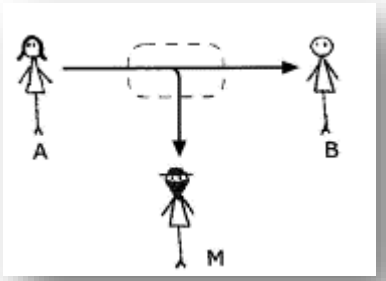
- Attack types:
  - **Spoofing** attack is a situation in which a malicious user successfully identifies as another
  - **Denial-of-service** (DoS) is an attack in which a malicious user seeks to make a host or service unavailable to its intended users

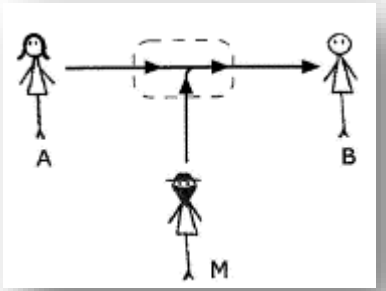




4. Security - Concepts

- Attack types:
  - **Man-in-the-middle** is an attack where the malicious user secretly relays between two parties. It can be:
    - Replay: the malicious user simply copy of communications between two remote entities
    - Modification: the attacker can modify and/or block selected traffic before relaying to the remote entity





4. Security - Concepts

- A security **hacker** is someone who explores methods for breaching defenses and exploiting weaknesses in a computer system or network
  - The term hacker can also refer to any skilled computer programmer (see *How to become a hacker* by Eric S. Raymond)
- A **cracker** on the other side carry out unauthorized access to computers in order to commit another crime (e.g. stealing, etc.)

4. Security - Services

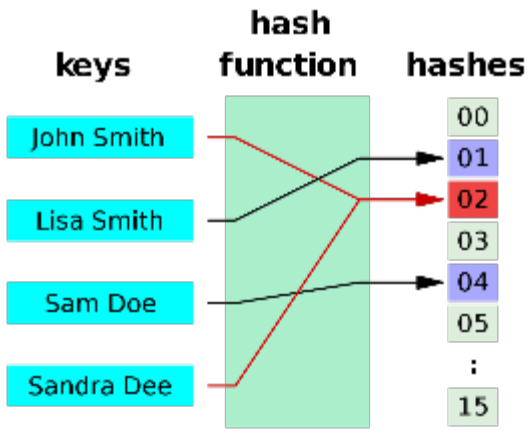
- A **security service** protects users' communications against certain attacks
  - **Confidentiality** is the property, that information is not made available to unauthorized users
  - **Authentication** is the act of verifying a claim of identity
  - **Authorization** determines what resources or actions are permitted to given users
  - **Integrity** means maintaining and assuring the accuracy and completeness of data

4. Security - Services

- **Authentication** is achieved with:
  - Something you know (e.g. passwords)
  - Something you have (e.g. Id, mobile)
  - Something you are (e.g. fingerprints)
- **Authorization** is typically achieved with grants, roles, tokens
  - AAA (Authentication, Authorization and Accounting)

4. Security - Services

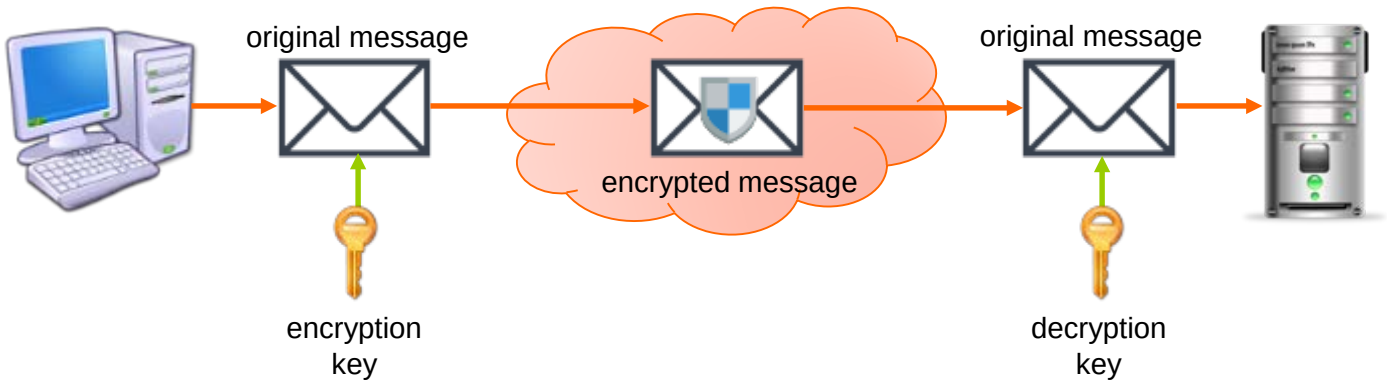
- **Integrity** can be done using **hash functions**
  - A hash function is a method that can be used to map data of arbitrary size to fixed-size values
  - Hash function should minimize duplication of output values (collisions)



Examples of hash functions:
- SHA (Secure Hash Algorithm)
- MD5 (Message-Digest Algorithm 5)
- DSA (Digital Signature Algorithm)

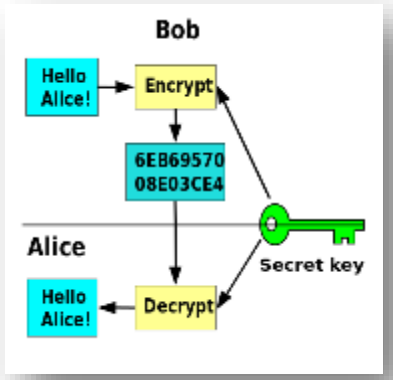
4. Security - Cryptography

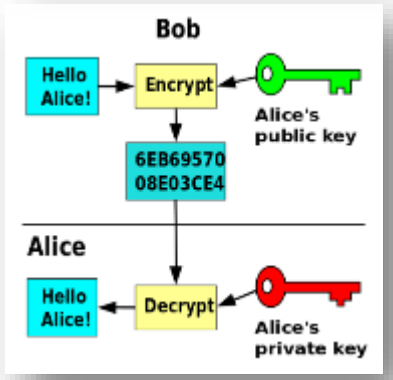
- **Confidentiality** is achieved using **cryptography** (prevent third parties from reading private messages)
- A **cryptosystem** is a suite of cryptographic algorithms for achieving confidentiality (encryption)



4. Security - Cryptography

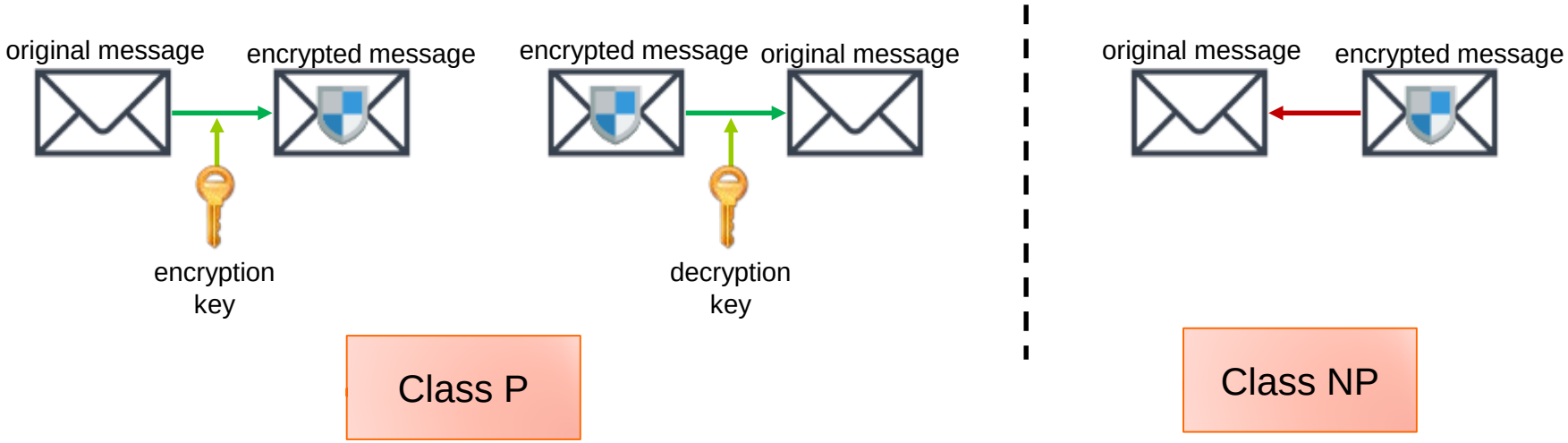
- Cryptosystems types:
  - **Symmetric-key** systems use the same cryptographic keys for both encryption of plaintext and decryption
  - **Public-key** (or **asymmetric**) systems use pairs of keys: public keys which may be disseminated widely, and private keys which are known only to the owner





4. Security - Cryptography

- Public-key cryptosystems use algorithms which can be solved can be solved by a deterministic Turing machine (P complexity) in a way, but non-deterministic (NP complexity) in the order way



4. Security - Cryptography

- For example, public-key algorithm use **prime factorization**, which is finding which prime numbers multiply together to make the original number
  - When the numbers are sufficiently large, no efficient, prime factorization algorithm is known
  - The public key consists of a product of two large primes used to encrypt a message
  - The private key (prime factors) are used to decrypt messages

4. Security - Cryptography

- Cryptosystems examples:
  - Public-key (or asymmetric):
    - RSA (Rivest, Shamir y Adleman)
    - Diffie-Hellman
    - ElGamal
    - Elliptic-curve cryptography (ECC)
  - Symmetric-key:
    - AES (Advanced Encryption Standard)
    - ES (Data Encryption Standard)
    - IDEA (International Data Encryption Algorithm)
    - 3DES
    - RC2, RC4, RC5
    - Blowfish

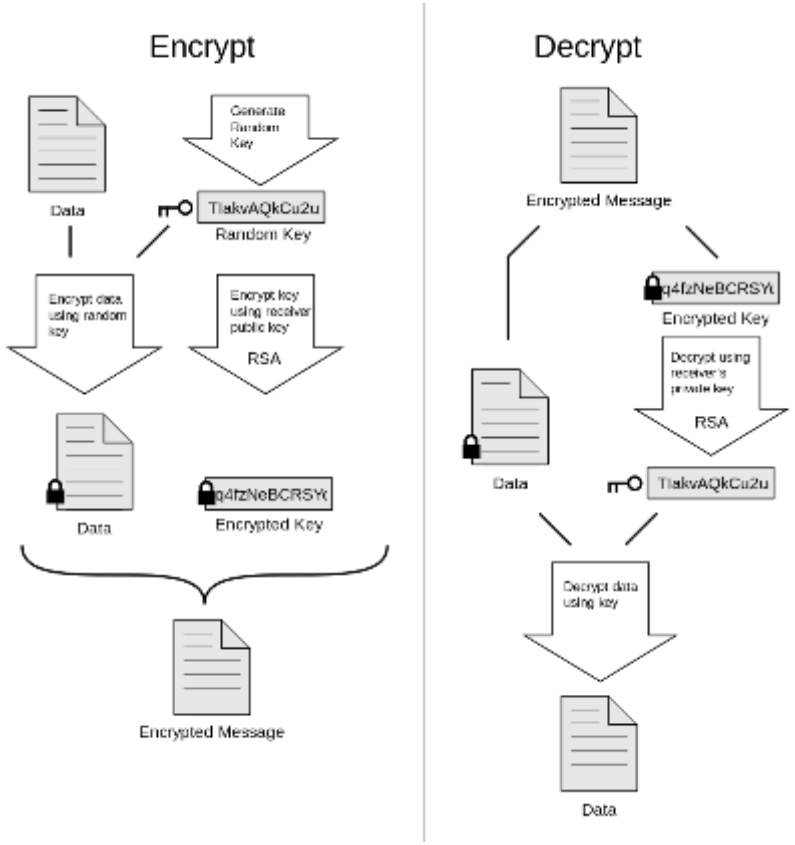
4. Security - Cryptography

- How can we make a public key public?

1. Through a set of hosts and procedures called Public Key Infrastructure (PKI) intended to provide security services
   - In this model there are trusted entities called Certification Authority (CA) that issue digital certificates
   - The digital certificate is a piece of information that associates an entity with its public keys
2. Through public key servers
   - The public keys of the users are stored on a key server
   - For example: keys.gnupg.net, pool.sks-keyservers.net, pgp.mit.edu

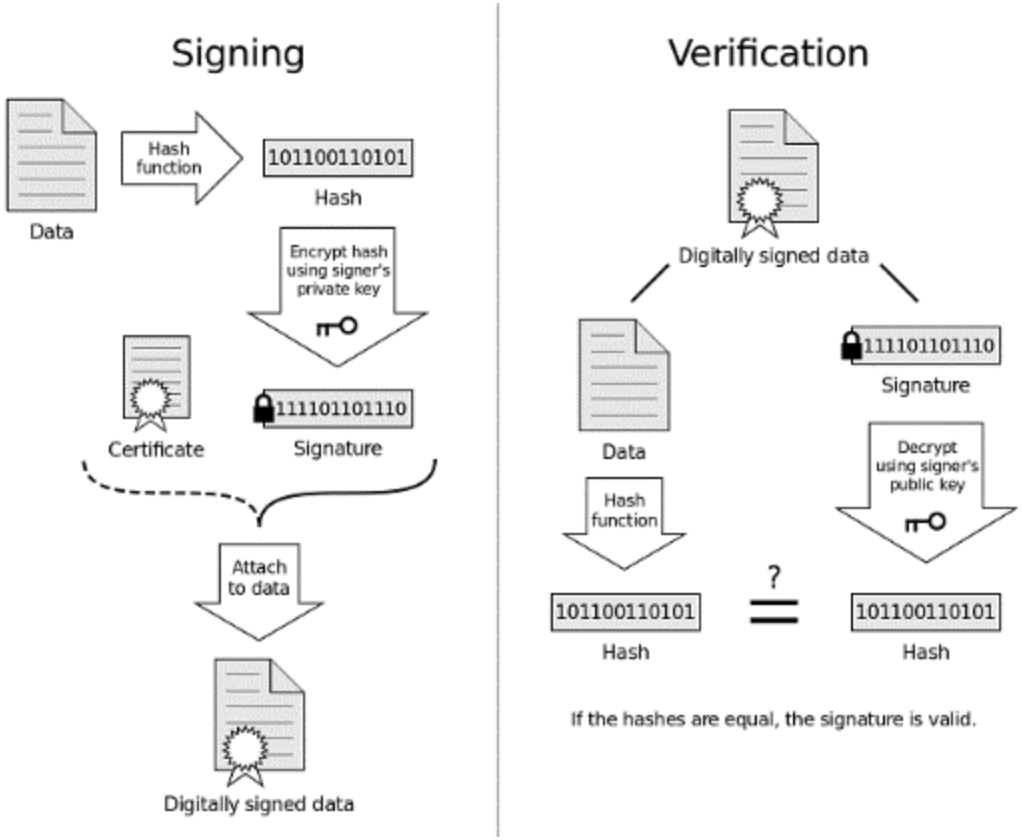
4. Security - Cryptography

- Example #1: PGP (Pretty Good Privacy)
  - End-to-end encryption for emails



4. Security - Cryptography

- Example #2: Digital Signature
  - Verifying the authenticity of digital messages



4. Security - TLS

- **Transport Layer Security** (TLS) is a cryptographic protocols designed to provide security over a computer network:
  - Confidentiality: data is encrypted at TCP level
  - Authentication: entities can be authenticated (typically servers)
  - Integrity: a hash function is used to assure data consistency

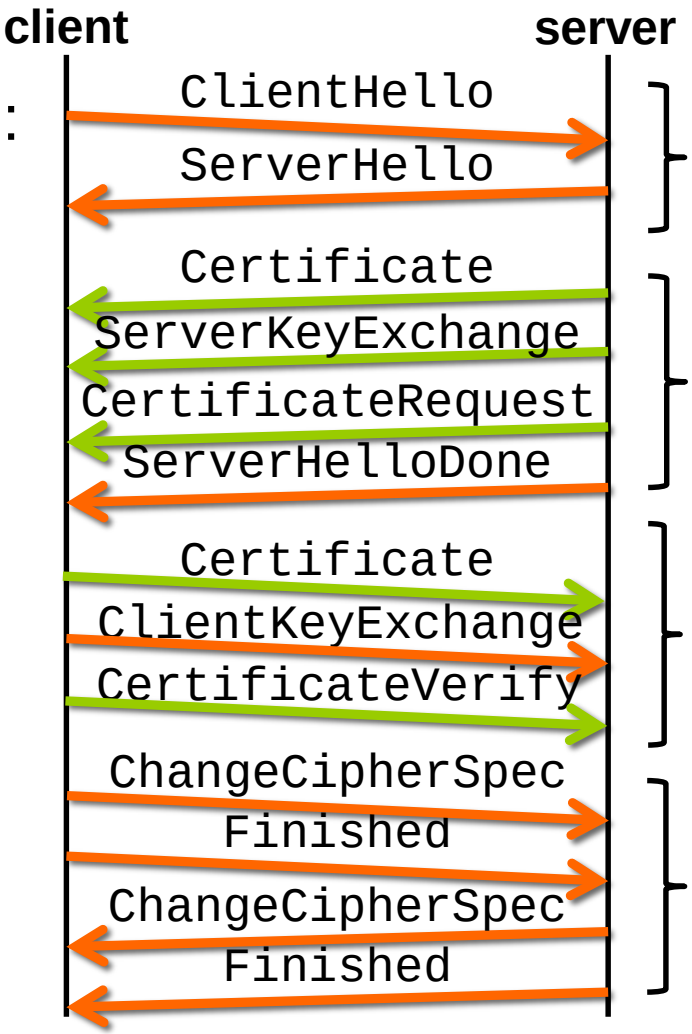
4. Security - TLS

- TLS is the successor of the deprecated Secure Sockets Layer (SSL)
- There is a variant of TLS that works on UDP called DTLS (Datagram Transport Layer Security)

Version	Year
SSL 2.0	1995
SSL 3.0	1996
TLS 1.0	1999
TLS 1.1	2006
TLS 1.2	2008
TLS 1.3	2018

4. Security - TLS

- TLS handshake:

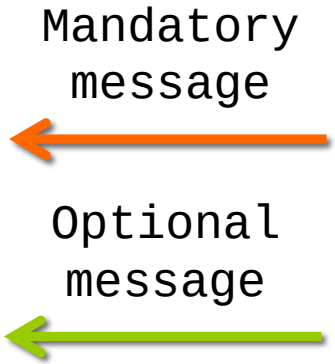


Step 1. The client informs the encryption algorithms it supports. The server chooses one possible

Step 2. Optionally, the server sends or requests a certificate

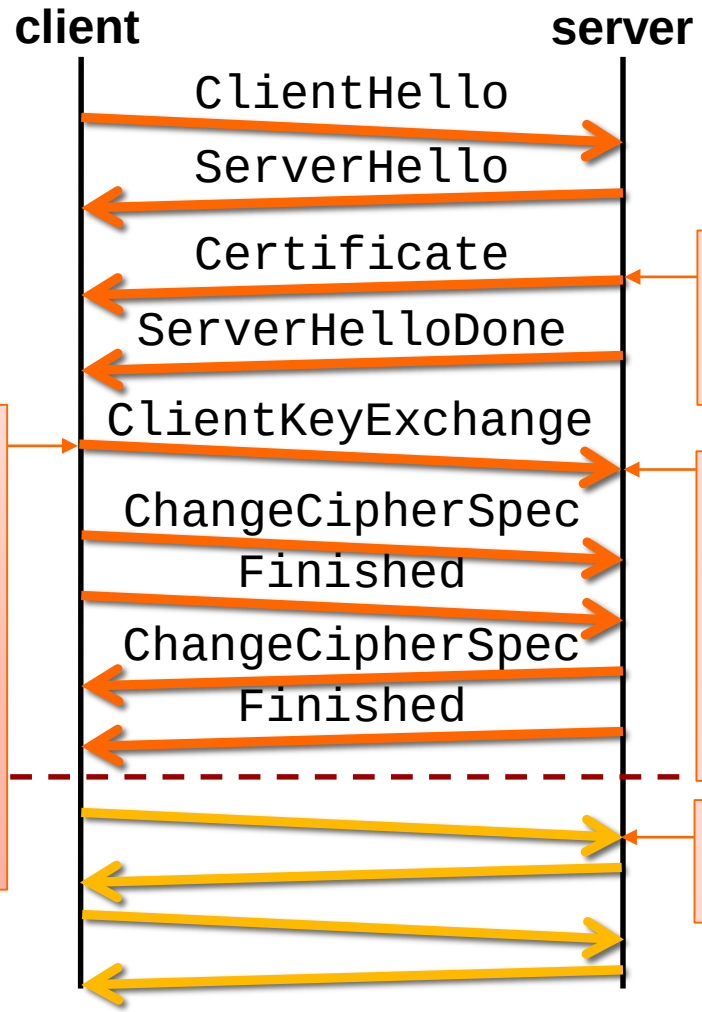
Step 3. Optionally, the client sends certificate or verifies the server certificate. The client sends a pre-master key that will be used to generate the exchange key (MS, Master Secret)

Step 4. Finish handshake and start symmetric encryption using MS



4. Security - TLS

- Example: HTTPS



When sending the certificate the server is sending its public key to the client

The client generates a pre-master key that will be used to generate the MS master key with which all session data will be encrypted. This key is sent encrypted with the server's public key, obtained from the certificate

The server obtains the pre-master key by deciphering the message sent by the client, using its private key. The MS is obtained from the pre-master

Data exchange through symmetric encryption

Table of contents

5. Takeaways

- **Transport layer** provides logical communication between applications **processes** running on hosts
- A **port** is a number used to identify an entity at transport level
- There are 2 transport protocols in Internet:

1. **UDP** (User Datagram Protocol)
   - Connectionless protocol
   - Unreliable, unordered delivery (best effort)
2. **TCP** (Transmission Control Protocol)
   - Connection oriented protocol
   - Reliable, in-order delivery
   - Congestion and control flow