

**TC**

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE LYON

**3TC**

**PRS**

# Programmation Réseau et Système

# PRS

- **Equipe pédagogique**

- Responsable de cours: Razvan Stanica
- Intervenants TP: Oana Iova, Frédéric Le Mouel, Philippe Isorce, Mihai Popescu

- **Objectif**

- Faire le lien entre « réseaux » et « programmation »
- Comprendre le rôle et le fonctionnement de la couche transport
- Première utilisation de l'API Sockets



# PRS

## • Structure du cours

- 4h de cours (rappels TCP et API Sockets)
- 2h de TD (fonctionnement TCP)
- 8h de TP « guidé » sur l'API Sockets
- 16h de TP sur les mécanismes TCP
- 12h de projet – Implantation d'une couche transport pour un scénario donné

## • Evaluation

- 2 TPs notés (API Sockets)
- points bonus/malus pour les autres TPs
- présentation du projet
- tests du projet



# TC

INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE LYON

**3TC**

**PRS**

# Transmission Control Protocol

# TCP

- TCP = Transmission Control Protocol
  - Basic concepts already discussed in 3TC NET
- Pre-requisites for PRS
  - TCP header format
  - Connection management
  - TCP state machine
- PRS objective: TCP congestion control



# TCP

- TCP Congestion Control

- Slow Start
- Congestion Avoidance
- Fast Retransmit
- Fast Recovery
- Selective Acknowledgements

- Standardized mechanisms

- Original TCP – RFC 793
- Additional mechanisms – RFC 1122, RFC 2581, RFC 5681



# TCP

- **Flow Control**

- Manage the data rate at the transmitter in order to not overwhelm a slower receiver

- **Congestion Control**

- Manage transmission rate in order to avoid network congestion collapse

- **End-to-End Argument**

- Represents the philosophy behind TCP (and behind Internet)
- Data flow rate is controlled by end hosts
- The network does not provide any congestion or flow control support



# TCP

## • TCP Vocabulary

- Segment = the TCP payload data unit (different from “message”, “packet”, or “frame”)
- Maximum Segment Size = the size of the largest segment that can be transmitted/received. The result of a negotiation between end hosts
- Receiver Window (rwnd or awnd) = the number of segments a host can receive at a given moment. Used for flow control purposes
- Congestion Window (cwnd) = a maximum number of segments that can be transmitted by a host, decided by congestion control mechanisms





# TCP

- **Basic transmission principle**

- At any given time, a TCP host must not transmit a segment with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of  $cwnd$  and  $rwnd$

- **Important metric**

- Round-Trip Time (RTT): the time between the transmission of the segment and the reception of the ACK. RTT can vary significantly during network operation, so TCP keeps an updated estimated value



# TCP

## • FlightSize

- The amount of data that has been sent, but not yet acknowledged
- A common mistake is to consider  $\text{FlightSize} = \text{cwnd}$

## • Congestion detection

- Based on the assumption that a segment lost in the network is the result of congestion
- A sender starts a timer (based on its RTT estimate) every time it transmits a segment
- If an ACK from the destination is not received before the timeout, a loss is detected

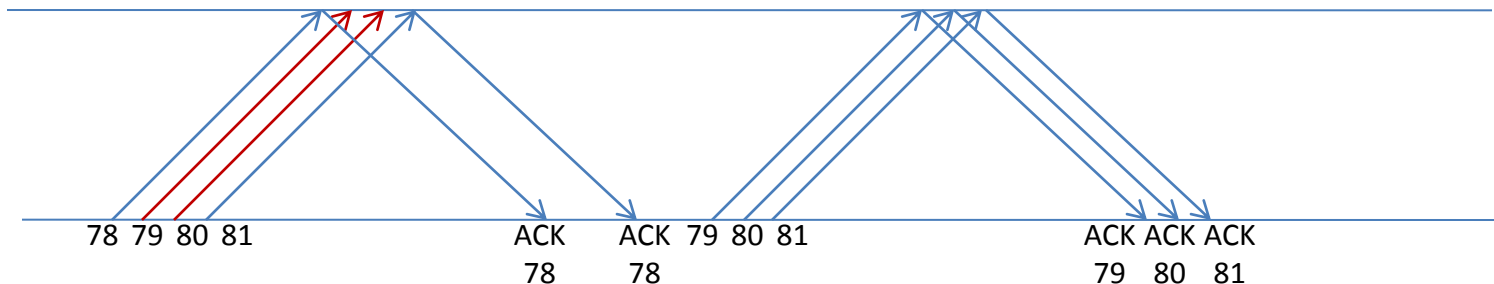




# TCP

## • Duplicate ACKs

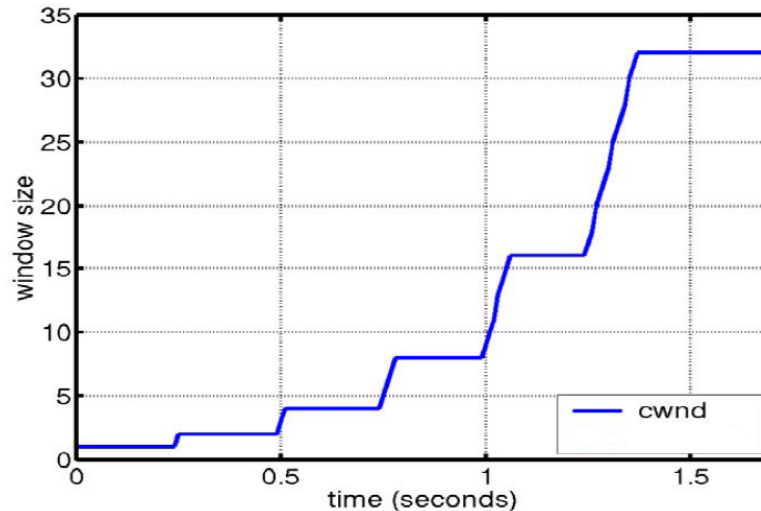
- In normal operation, a receiver is not allowed to acknowledge discontinuous segments
- The reception of an out-of-order segment results in the acknowledgement of the last contiguous segment (a duplicate ACK)



# TCP

## • Slow Start

- Motivation: the end hosts do not know the state of the network at the beginning of their connection
- Start with  $cwnd = 1$
- For every received ACK:  $cwnd = cwnd + 1$
- Practically,  $cwnd$  doubles during an RTT interval
- Despite its name, exponential increase of  $cwnd$



# TCP

- Slow Start Threshold (ssthresh)

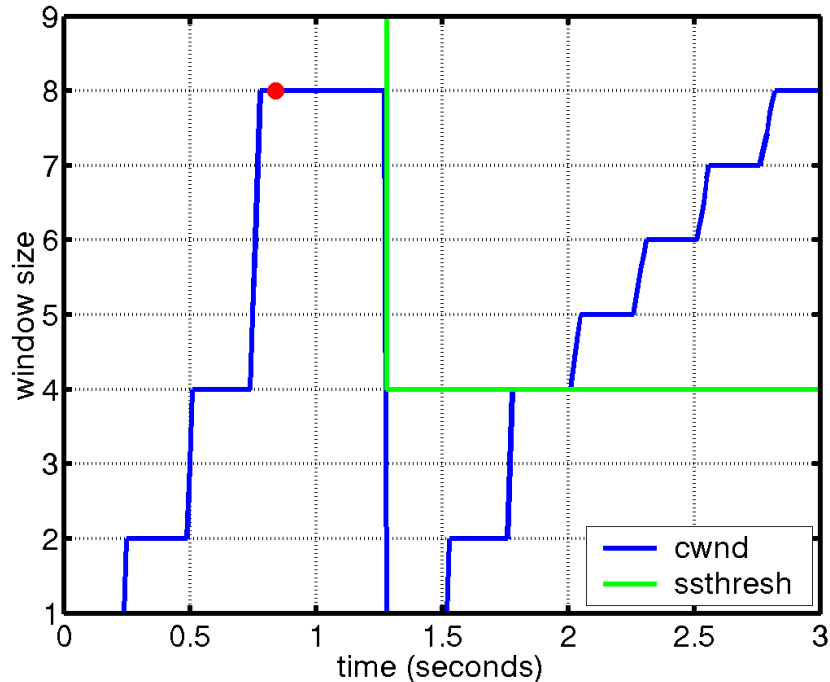
- Important TCP parameter
- Decides the moment when the host goes from Slow Start to Congestion Avoidance
- Arbitrary initial value (usually very high)
- ssthresh must follow the congestion level
- After a lost segment (detected through a timeout or duplicate ACK):  $ssthresh = FlightSize / 2$
- After the retransmission:  $cwnd = 1$





# TCP

- Congestion Avoidance
  - The TCP host enters in this mode when  $cwnd > ssthresh$
  - $cwnd = cwnd + 1/cwnd$
  - For each RTT:  $cwnd = cwnd + 1$



# TCP

## • Congestion Avoidance

- Motivation: the exponential increase of Slow Start is too aggressive
- Once a congestion has been detected, the transmitter tries to avoid reaching the congested state once again
- A static approach can miss the opportunity of an increased throughput
- The slow cwnd increase can delay the next congestion, while still testing for transmission opportunities



# TCP

## • Fast Retransmit

- A timeout is a clear indication of network congestion, but can be very long
- A duplicate ACK can have different reasons: congestion, segments following different paths, re-ordered ACKs
- Considering a segment lost after the first duplicate ACK is too aggressive
- TCP considers a segment lost after 3 duplicate ACKs (that means 4 consecutive ACKs of the same segment)



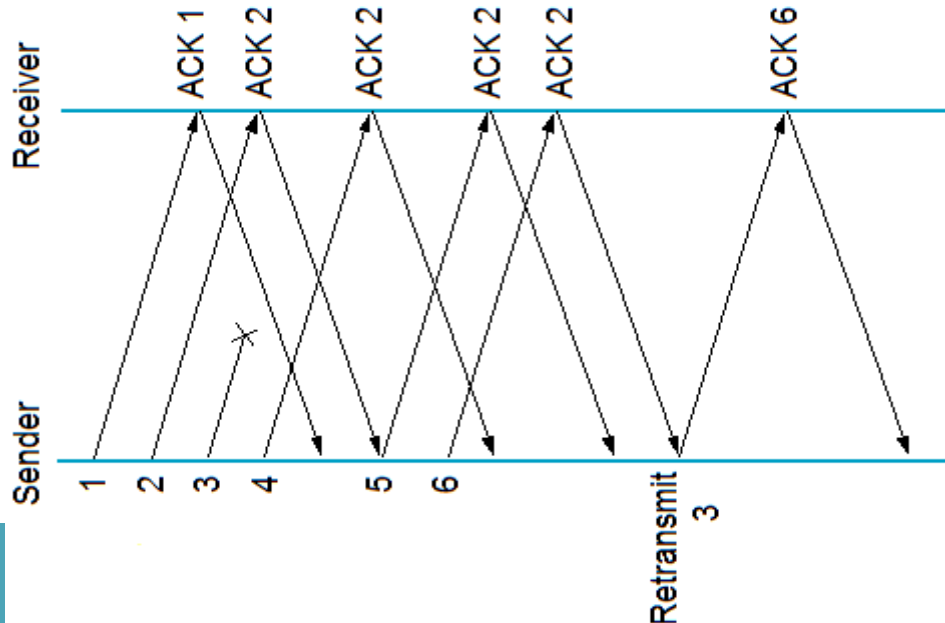




# TCP

## • Fast Retransmit

- The usual operation mode
- Retransmit lost message
- Calculate FlightSize=  $\min(\text{rwnd}, \text{cwnd})$
- $\text{ssthresh} = \text{FlightSize}/2$
- Enter Slow Start:  $\text{cwnd} = 1$



# TCP

- Fast Retransmit

- This mechanism generally eliminates half of the TCP timeouts
- This yields roughly a 20% increase in throughput
- It does not work when the transmission window is too small to allow the reception of three duplicate ACKs



# TCP

- **Fast Recovery**

- The reception of duplicate ACKs also means that network connectivity exists, despite a lost segment
- Entering Slow Start is not optimal in this case, as the congested state might have disappeared
- The mechanism allows for higher throughput in case of moderate congestion
- Complement of Fast Retransmit



# TCP

## • Fast Recovery

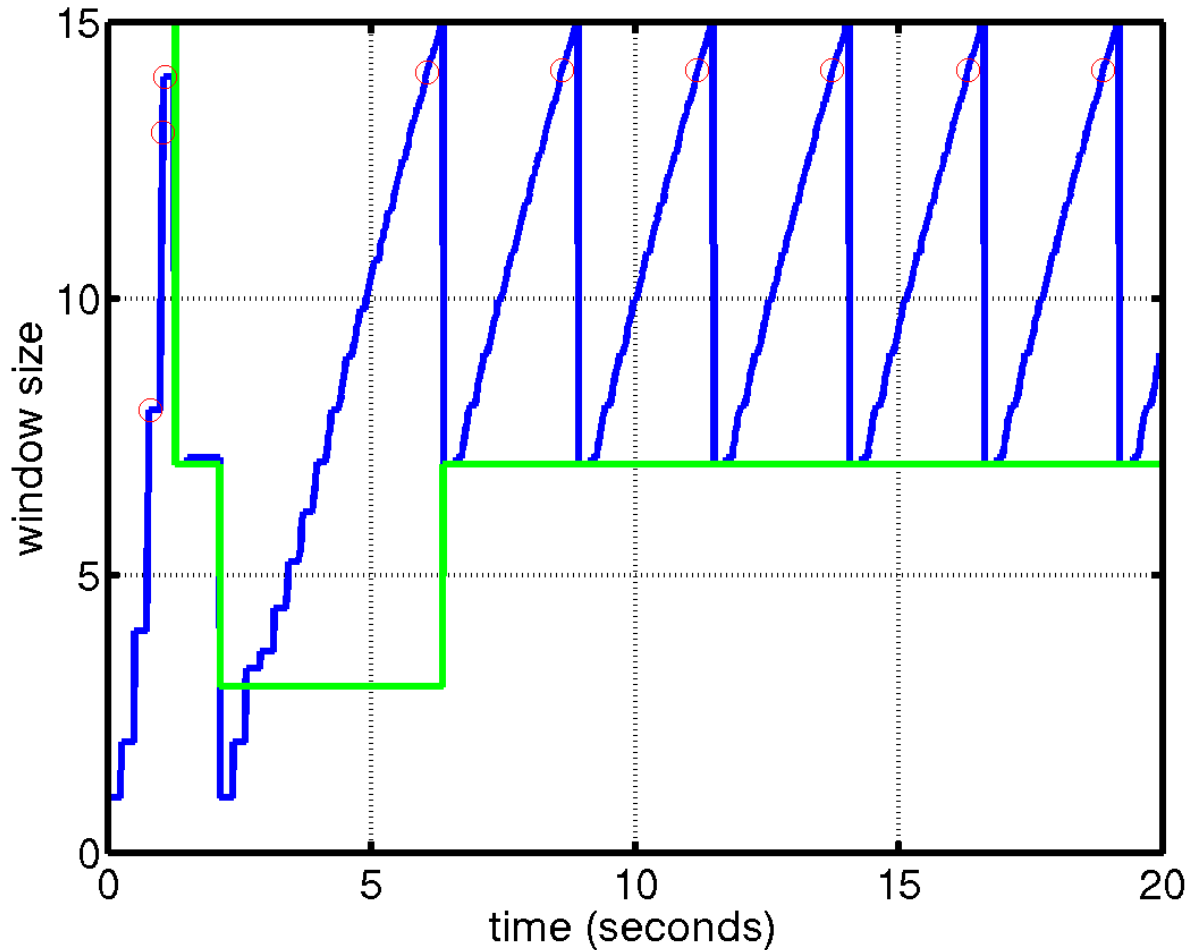
- Mode entered after 3 duplicate ACKs
- As usual, set  $ssthresh = FlightSize/2$
- Retransmit lost packet
- Window inflation:  $cwnd = ssthresh + ndup$  (number of duplicate ACKs received)
- This allows the transmission of new segments
- Window deflation: after the reception of the missing ACK (one RTT later)
- Skip Slow Start, enter Congestion Avoidance





# TCP

- Typical TCP Saw-tooth Pattern



# TCP

## • Selective Acknowledgements

- The receiver can only acknowledge contiguous segments
- No ACK for segments correctly received after a lost segment
- The sender has no feed-back regarding correctly received segments: retransmit or not?
- Ideally, the sender should retransmit only the missing segments
- With SACK, the receiver provides this feed-back to the sender



# TCP

- Delayed ACK

- RFC 1122
- Reduce overhead by combining multiple ACKs in one segment
- Delay an ACK by up to 500 ms
- For a stream of full-sized incoming segments, an ACK is sent every second segment
- Can be interesting for piggy-backing: data and ACK in the same segment



# TCP

- Nagle's algorithm
  - RFC 896
  - Small packet problem
  - Combine small outgoing data and send one single segment
  - If segment with un-received ACK, keep buffering output data until a full size segment can be sent
  - Poor interaction with delayed ACKs





# TCP

## • TCP Versions

- TCP Tahoe: Slow Start, Congestion Avoidance, Fast Retransmit
- TCP Reno: Fast Recovery
- TCP New Reno: Modified Fast Recovery (window inflation)
- Many other proposals exist: Vegas, Hybla, BIC, Westwood, ...



# TCP

- TCP Cubic

- Current state of the art
- Window size no longer controlled by received ACKs
- cwnd computed as a cubic function of time since the last congestion
- Three phases:
  - aggressive increase until ssthresh (similar to slow start)
  - slow probing for higher window
  - aggressive probing for higher window



# TCP

- **Beyond the End-to-End Argument**

- The way routers decide to drop packets impacts the functioning of TCP
- Advanced techniques can be implemented inside the network

- **Random Early Detection**

- RED manages router queues and drops packets based on a queue threshold
- Once the queue is over the threshold, the router drops packets with a certain facility
- Only the affected TCP senders will enter Slow Start or Congestion Avoidance, slowing the network down before the actual congestion



# TCP

## • Explicit Congestion Notification

- ECN is based on a queue threshold parameter, just as RED
- As opposed to RED, ECN only marks packets instead of dropping them
- Routers mark 2 bits in the IP header (Type of Service field) to signal whether congestion is occurring
- Through cross-layer mechanisms, TCP can learn this information and reduce the congestion window
- ECN avoids packet drops and reduces the delay created by retransmissions



# TCP

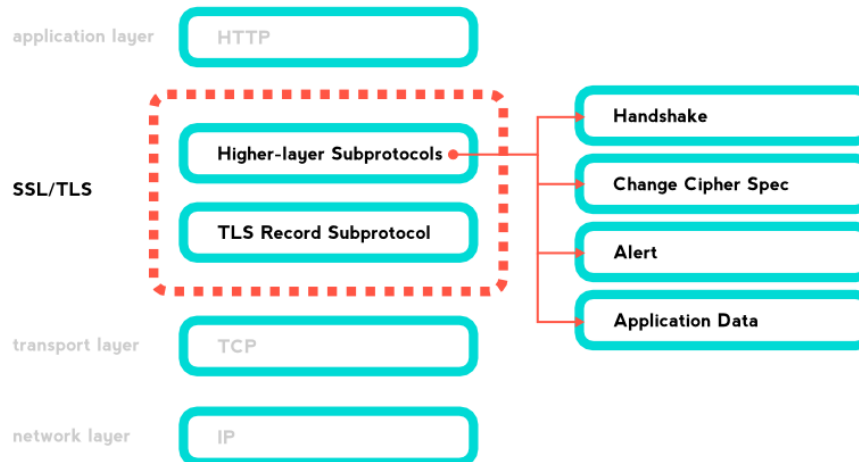
- QUIC – Quick UDP Internet Connections
  - User space implementation of a transport protocol
  - Released by Google in 2013
  - From January 2017, implemented in the Chrome browser and the Google Search and You Tube applications
  - Currently transports between 5% and 10% of the Internet traffic
  - 5% reduction in search time
  - 15% reduction in video rebuffering



# TCP

## • QUIC – Motivations

- Classic functioning: HTTP/2 – TLS – TCP
- Transport Layer Security (TLS) – extra overhead
- Data transmitted after 2xRTT

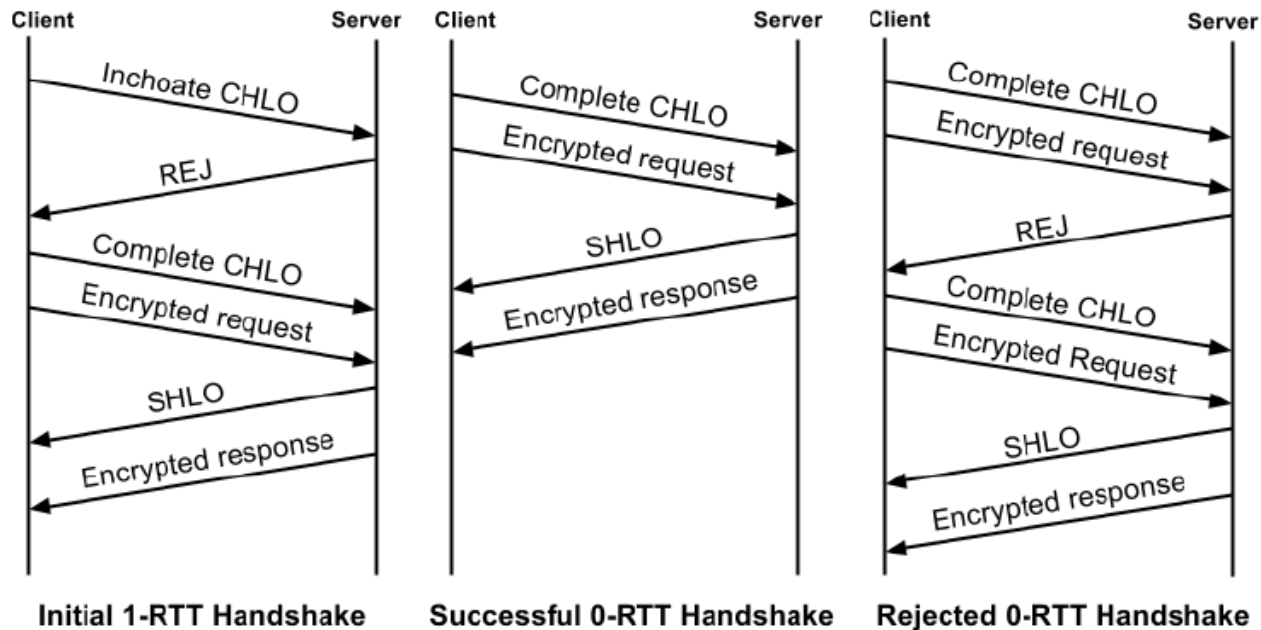




# TCP

## • QUIC – Principles

- Save the context of already known servers/clients
- Results in 0-RTT connection in 85% of the cases



# TCP

## • QUIC – Principles

- Multiple streams transmitted over the same connection (similar mechanisms in TCP)
- Streams are controlled both independently (flow controlled) and per connection
- A large part of the transport layer information is encrypted
- Unique identifier, even for retransmissions, easing RTT estimation
- TCP congestion control mechanisms





# TCP

## • QUIC – Problems

- Not always better performance than TCP (e.g. when a lot of packets are delivered in disorder)
- 2x CPU consumption compared with TCP
- TCP unfriendly



# TCP

- Implementation

- At the transport layer, an active application is identified by the 5-tuple: (protocol, @IP<sub>source</sub>, Port<sub>source</sub>, @IP<sub>dest</sub>, Port<sub>dest</sub>)

- A client needs to know @IP<sub>server</sub> and Port<sub>server</sub> in order to send a connection request

- The Port<sub>client</sub> can be allocated dynamically by the operating system



# TCP

- Socket

- Application programming interface (API) for communication between processes
- When processes are run on different machines, a socket becomes the basis of network communications
- Support for both TCP and UDP
- Bidirectional communication using functions such as *read()/write()* or *send()/recv()*
- A socket is represented as a file handler in Unix systems



# TCP

- Socket API

- A series of libraries

- `sys/socket.h` – core socket functions and data structures
    - `netinet/in.h` – IP, TCP and UDP data structures
    - `sys/un.h` – data structures for local communications
    - `arpa/inet.h` – functions for manipulating IP addresses
    - `netdb.h` – functions for translating protocol and host names



# TCP

- Socket API

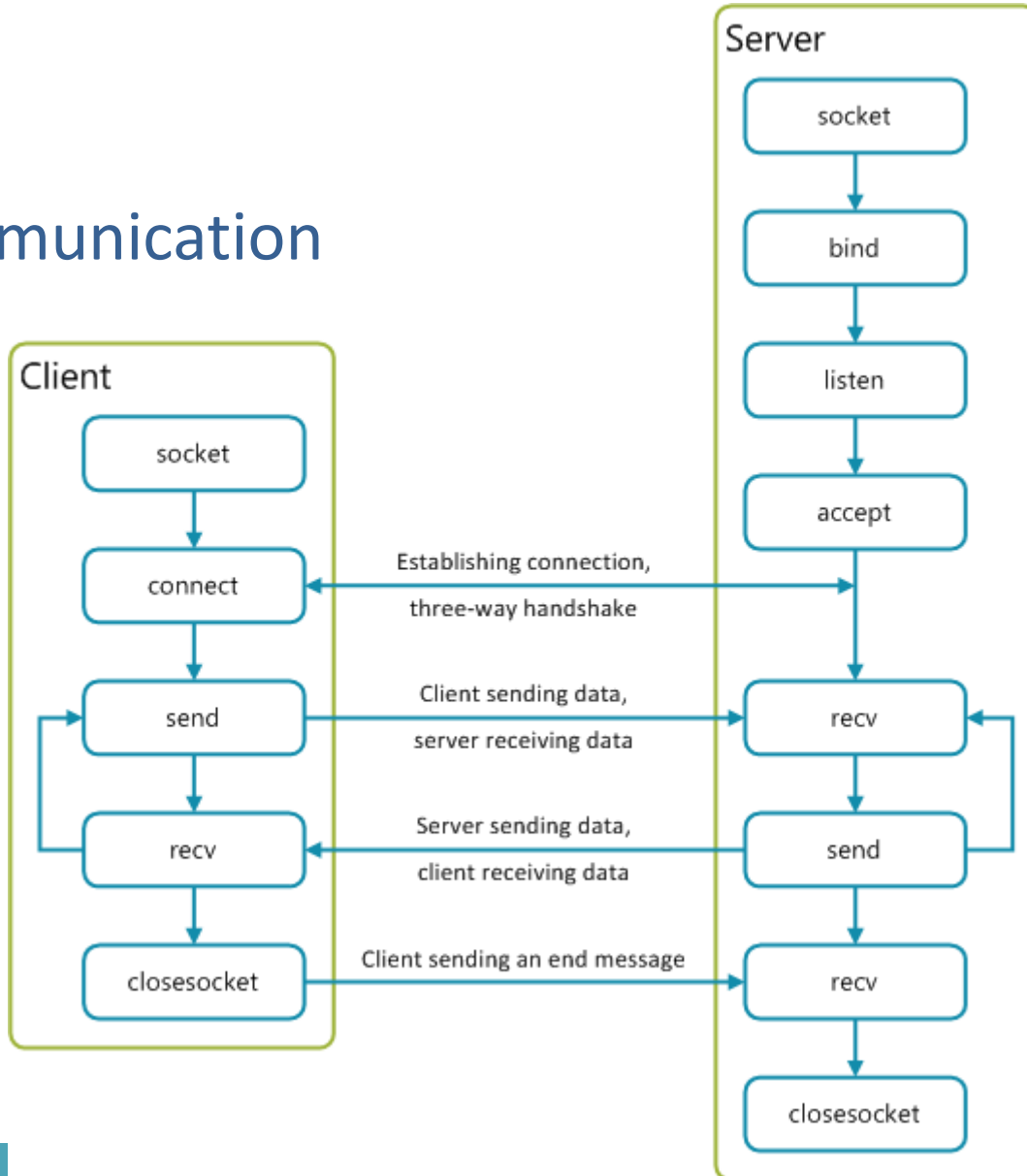
- A series of functions

- socket()
- bind()
- listen()
- connect()
- accept()
- send() / write()
- recv() / read()
- close()
- select()
- poll()



# TCP

- TCP communication



# TCP

- UDP communication

