La Sapienza
Università degli Studi di Roma
Dipartimento di Informatica e Sistemistica

Computer Networks II
TCP/UDP recap

Luca Becchetti
Luca.Becchetti@dis.uniroma1.it

A.A. 2009/2010

Data Link Versus Transport

- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT -> what is RTT?
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different node capacity
- Potentially different network capacity
  - need to be prepared for network congestion

Transport service

- Underlying best-effort network (Internet)
  - drops messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay
- Common end-to-end service requirements
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

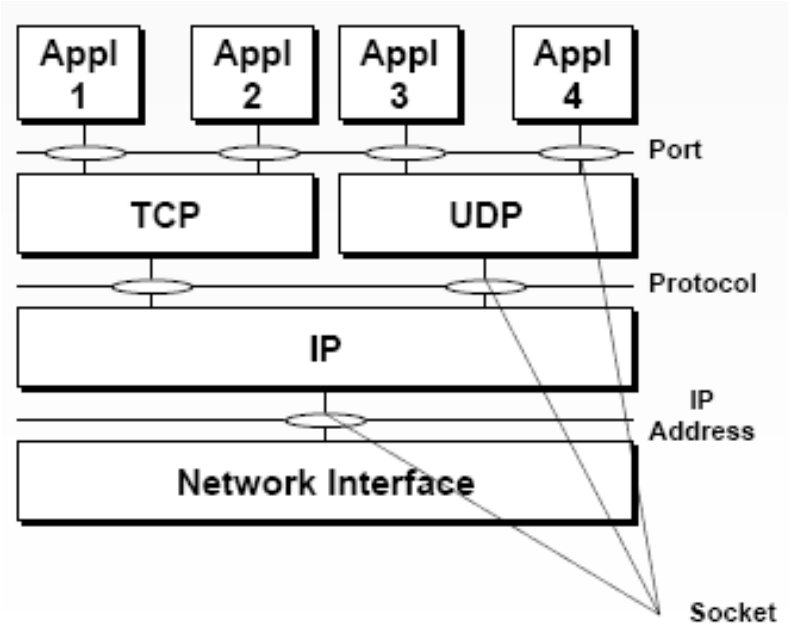
Transport service/cont.

- Provides transport service to applications, satisfying (when and to the extent possible) application requirements
- More common transport protocols:
  - User Datagram Protocol (UDP)
    - Used when no flow control or error detection/recovery required
    - Used by some real time applications having delay constraints
  - Transport Control Protocol (TCP)
    - Used when reliable service needed: flow control and error detection/recovery
  - Real-Time Transport Protocol (RTP)
    - Used for real time applications (ex., voce, video)
    - Note: works on top of UDP (so, technically, on top of another transport protocol)

Transport service - addressing

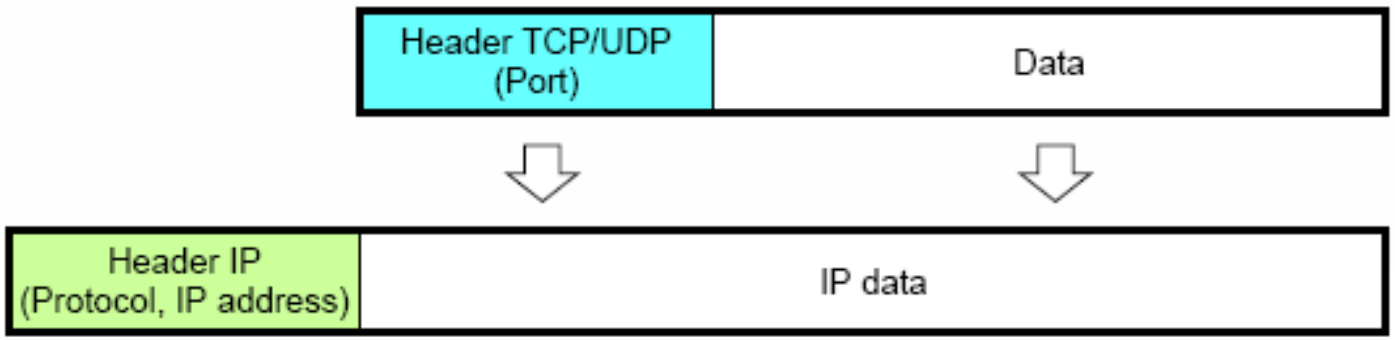
- Identifies processes that use the same transport service

- Port
  - Identifies a specific user (process) of transport layer
  - Represented as 16 bit integer
- Socket
  - Identifies interface between application and communication protocols
  - Represented by the triple (port; protocol; IP_Address)



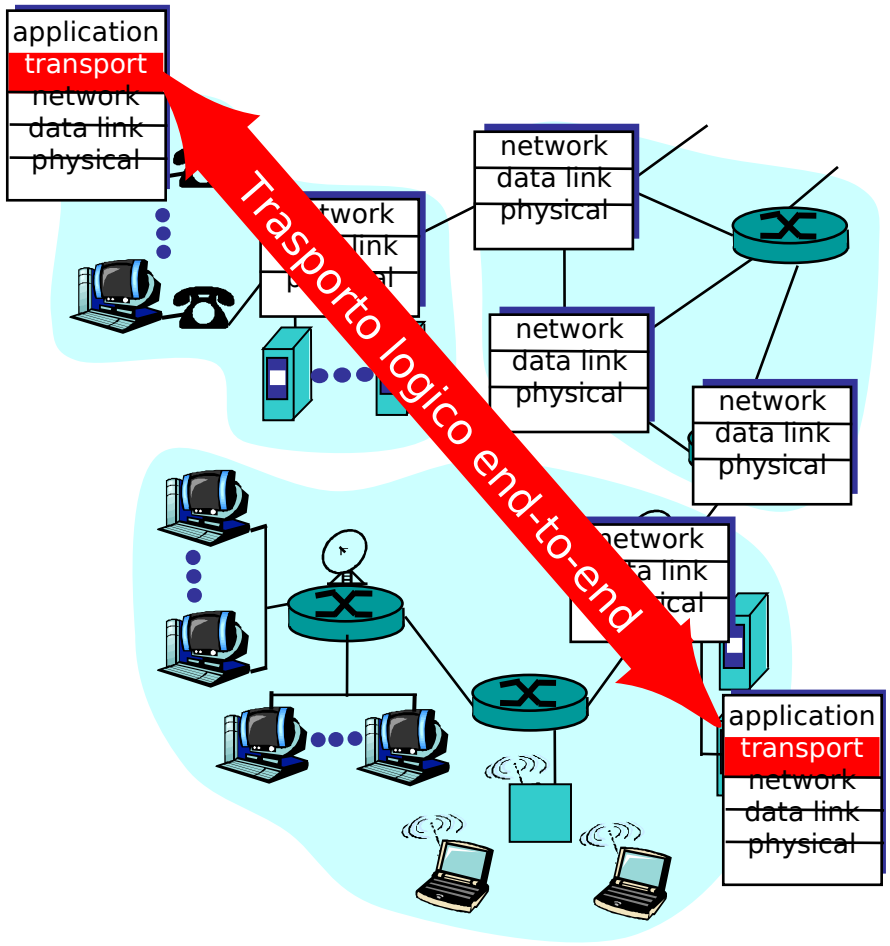
Transport service - addressing/cont.

- “Port” in TCP/UDP packet’s header
- “Protocol” and “IP_Address” in IP datagram’s header (multiplexing)



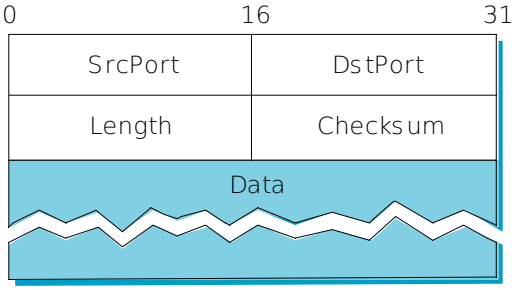
Internet transport protocol

- Reliable one-to-one transport (unicast): TCP
  - Congestion control
  - Flow control
  - Cannection oriented
- Non reliable one-to-one (unicast)/one-to-many (multicast): UDP
- Not available:
  - Real-time
  - Bandwidth guarantess



Simple Demultiplexer (UDP)

- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have *well-known* ports
  - see **`/etc/services`** on Unix
- Header format



- Optional checksum
  - pseudo header [IP header fields: source, dest, protocol, datagram length] + UDP header + data

User Datagram Protocol (UDP)

- No connection
- No error detection/recovery mechanisms
- Supports simple transaction services between applications
  - Database query
  - Address resolution [DNS]
  - Management message exchange
- May be used to develop customised “transport” protocols

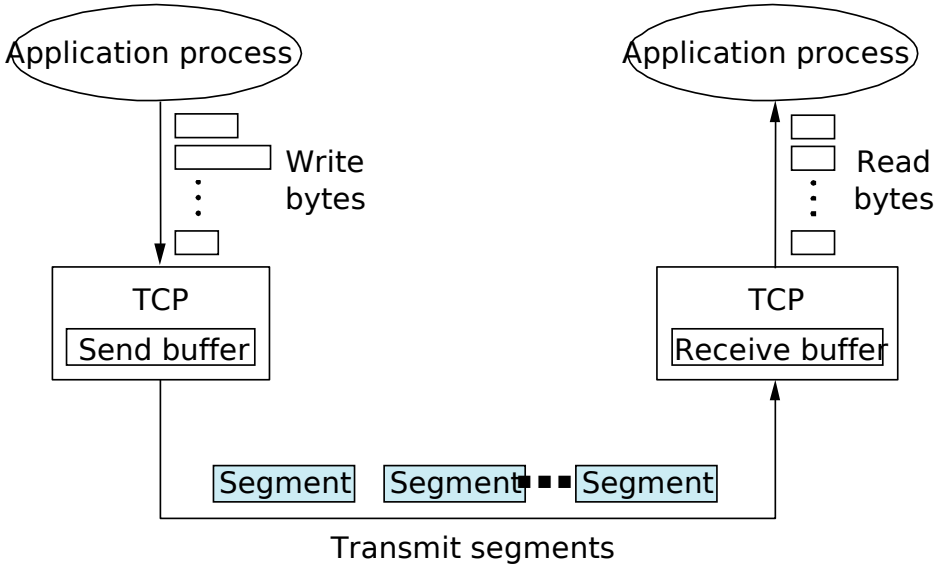
TCP Overview

- Bidirectional non structured information flow between hosts. Implements multiplexing/demultiplexing
- Connection oriented protocol
- Services provided to upper layers
  - Error detection/recovery
  - Flow control
  - Packet reordering
    - No order guarantess from IP!
  - Allows to address a specific application within a host

TCP Overview

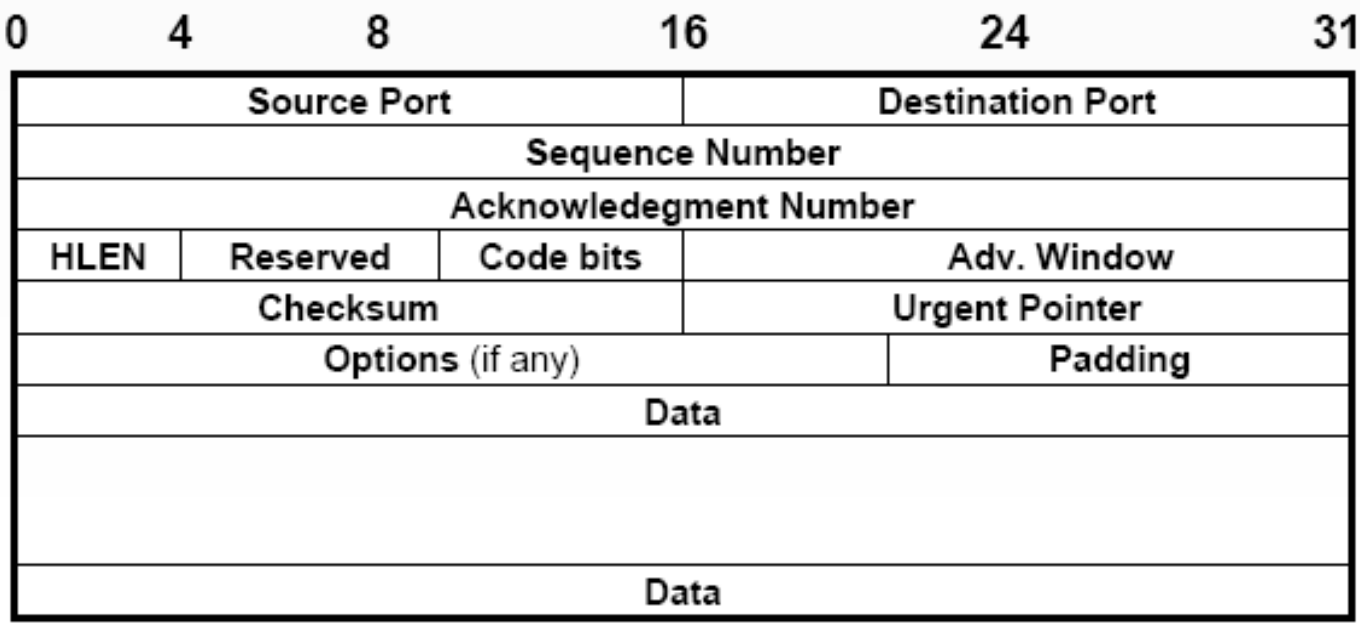
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes

- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



TCP segment format

- TCP sees data flow as a byte stream
- Byte stream partitioned into **segments**



TCP segment

- **Source Port (16 bits) and Destination Port (16 bits)**
  - Identify data source and recipient processes
- **Sequence Number (32 bits)**
  - Sequence number - transmission
  - Sequence number of first data byte in current segment with respect to beginning of TCP session
- **Acknowledgement Number (32 bit)**
  - Sequence number - receipt
  - If ACK=1: its value is sequence number of next byte expected by transmitter
  - Piggybacking acknowledgment possible -> What does it mean?

- Obs: Sequence & ACK number both refer to byte sequence and not to segment sequence

TCP segment

- **HLEN (4 bits)**
  - Number of 32 bit words contained in packet's TCP header
  - TCP header length never exceeds 60 bytes and is always multiple of 32 bits
- **Reserved (6 bits)**
  - Reserved for future use, now set to **0**
- **Advertised Window (16 bits)**
  - Windows size in bytes (byte oriented flow control)
  - Number of bytes that, starting from Ack Number's value, transmitter can receive (free receiving buffer size as segment was transmitted)
- **Checksum (16 bits)**
  - Used for error detection over entire segment + some IP header fields (pseudo header, see UDP)

TCP segment

- **Control bit (6 bits)**
  - URG
    - 1 when urgent pointer field contains meaningful value, 0 otherwise
  - ACK
    - 1 when Ack Number field contains meaningful value, 0 otherwise
  - PSH
    - 1 if data must be delivered to application, no matter what the state of receipt buffer is
  - RST
    - 1 if sender requests connection reset (“connection reset by peer”)
  - SYN
    - 1 only in first segment sent during connection set up to synchronize TCP entities [see connection set up]
  - FIN
    - 1 when source has no more data to send

TCP segment

- **Urgent Pointer (16 bits)**
  - Contains sequence number of last data byte that must urgently be delivered to receiving process
  - Typically control messages (out-of-band traffic)
- **Options (variable length)**
  - Only seldom present
  - examples:
    - End of Option List, No-operation, Maximum Segment Size (MSS)
- **Padding (variable length)**
  - Used ensure that overall header length is multiple of 32 bits

TCP addressing

- Port number can be
  - Static (Well Known port)
    - Identifiers statically associated to well known applications
      - See examples below
    - Identifiers below 256 used
  - Dynamic (Ephemeral)
    - Identifiers dynamically assigned by operating system at connection set up
    - Values strictly larger than 1023 used

Numero	Applicazione
7	Echo
21	FTP (File Transfer Protocol)
23	TELNET
25	SMTP (Simple Mail Transport Protocol

Numero	Applicazione
37	Time
53	Domain Name Server
103	X400 Mail Service
119	NNTP (USENET New Transfer Prot.)

Reliable transfer

- All packets sent reach destination in order
- Underlying network typically unreliable (e.g., IP)
- Tools
  - Sequence numbers / Acks
  - Connection
  - Timers
  - Retransmissions
  - Maximum Segment Lifetime

Sequence numbers/cont.

- 2354 bytes File to transmit
- Maximum Segment Size 500
- Initial sequence number: 181

0 1 2 2353

5 segments with sequence numbers 181, 681, 1181, 1681 and 2181 respectively are sent

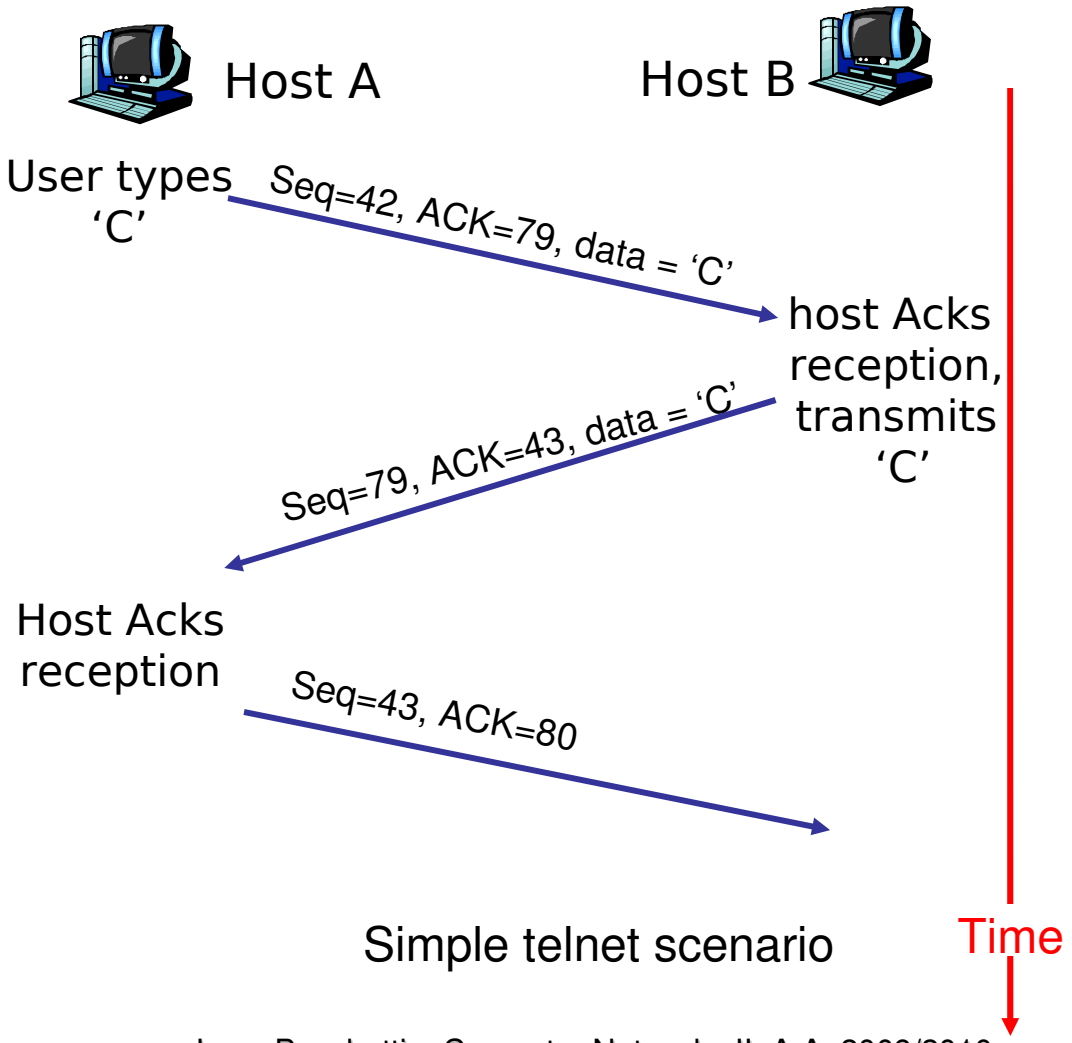
Seq. # and ACKs in TCP

Seq. number:

- Order number of first *data* byte in segment's payload

ACK:

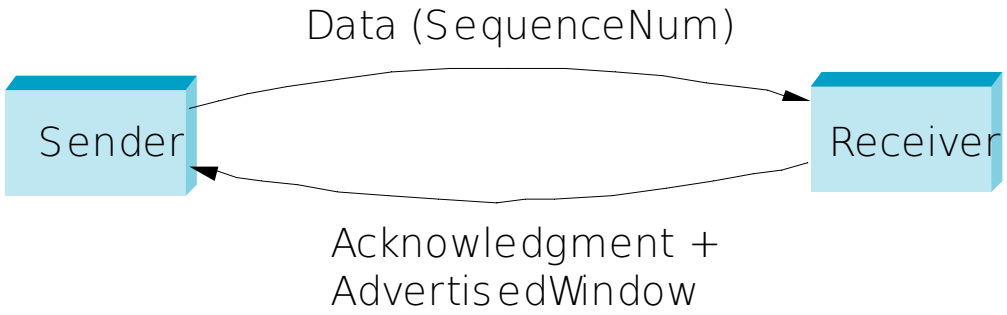
- Seq # of next *data* byte expected by sender
- Cumulative ACKs
- Q.: how to handle out-of-order segments
- A.: not specified by TCP, implementation dependent



Simple telnet scenario

TCP connection

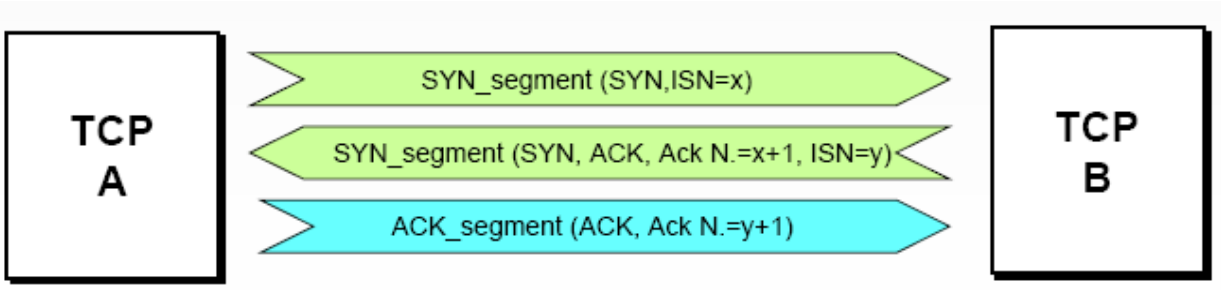
- Each connection identified by 4-tuple:
  - **(SrcPort, SrcIPAddr, DstPort, DstIPAddr)**
- Sliding window + flow control
  - **acknowledgment, SequenceNum, AdvertisedWindow**



- Order number of first *data* byte in segment's payload
- ACK:
  - # seq of next byte expected by *(this segment's)* sender
  - Cumulative ACKs

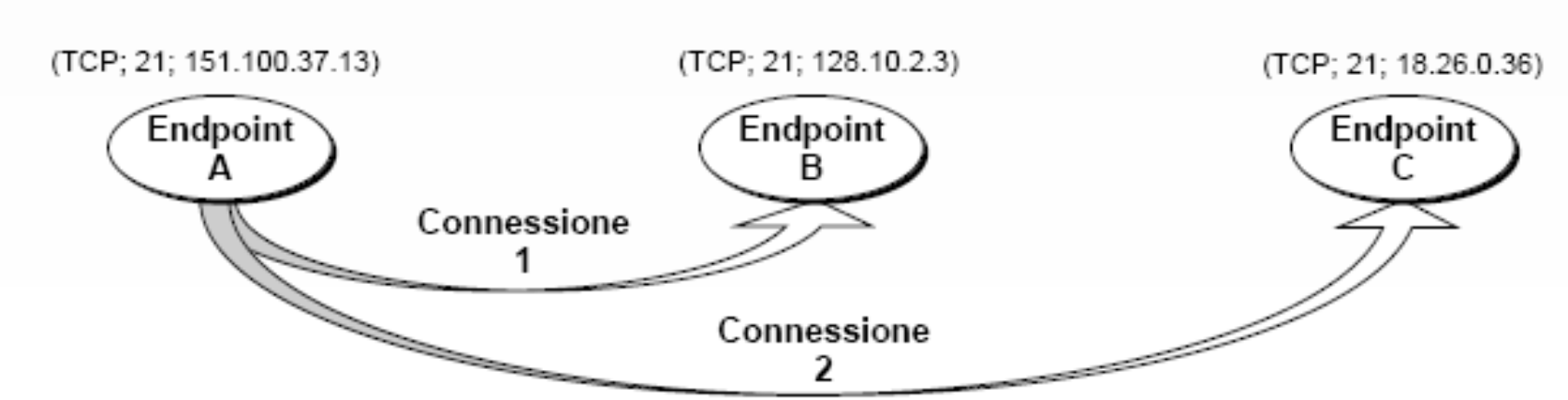
TCP connection

- Connection set-up: involved TCP entities synchronize by exchanging respective, initial sequence numbers. For any TCP entity, initial sequence number is the number starting with which all transmitted data bytes will be numbered sequentially
- Synchronization necessary to address potentially anomalous situations due to IP's unreliable transfer
  - Q.: why not **always** use **0** as initial sequence number?
- Synchronization occurs according to **"three way handshaking"** mechanism
- Each direction of the connection closed independently at connection shut down



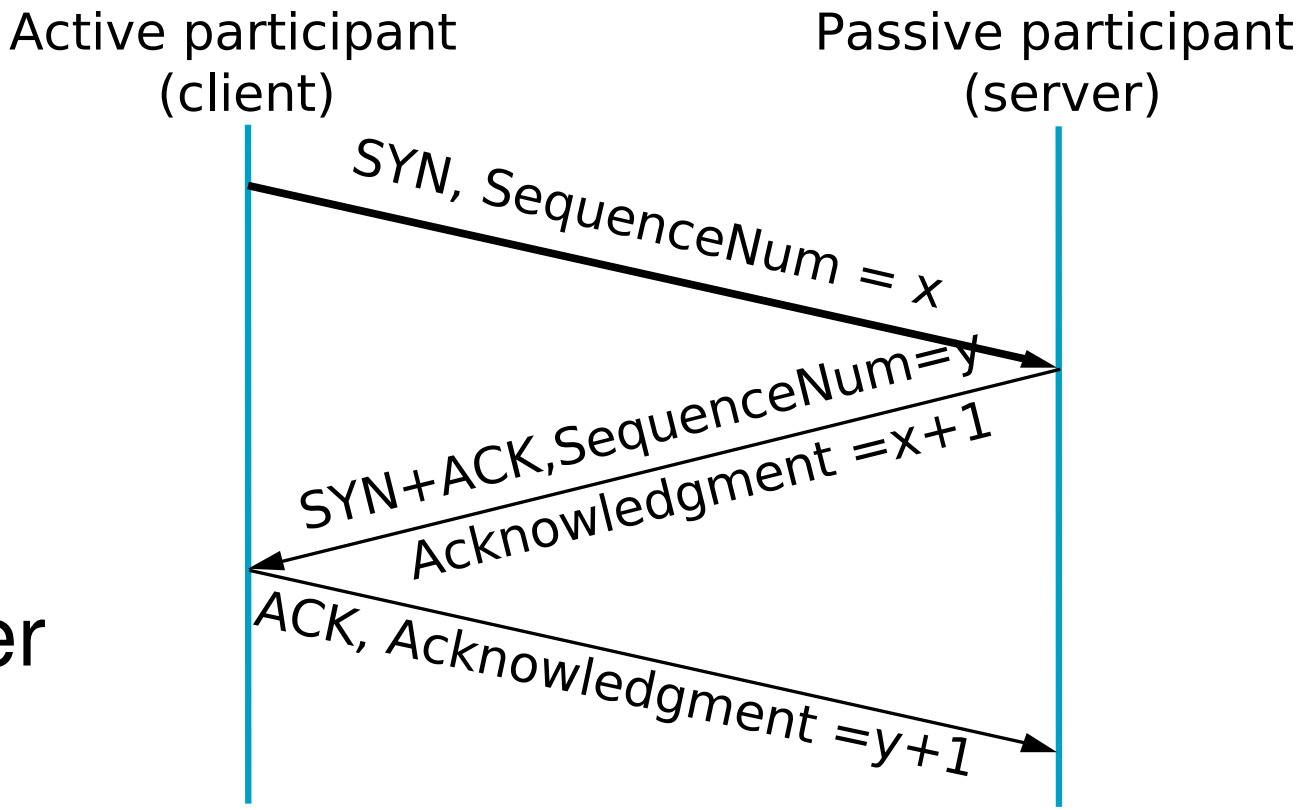
TCP connection

- A TCP connection is uniquely identified by the sockets associated to its endpoints
- One endpoint may be involved in one or more TCP connections

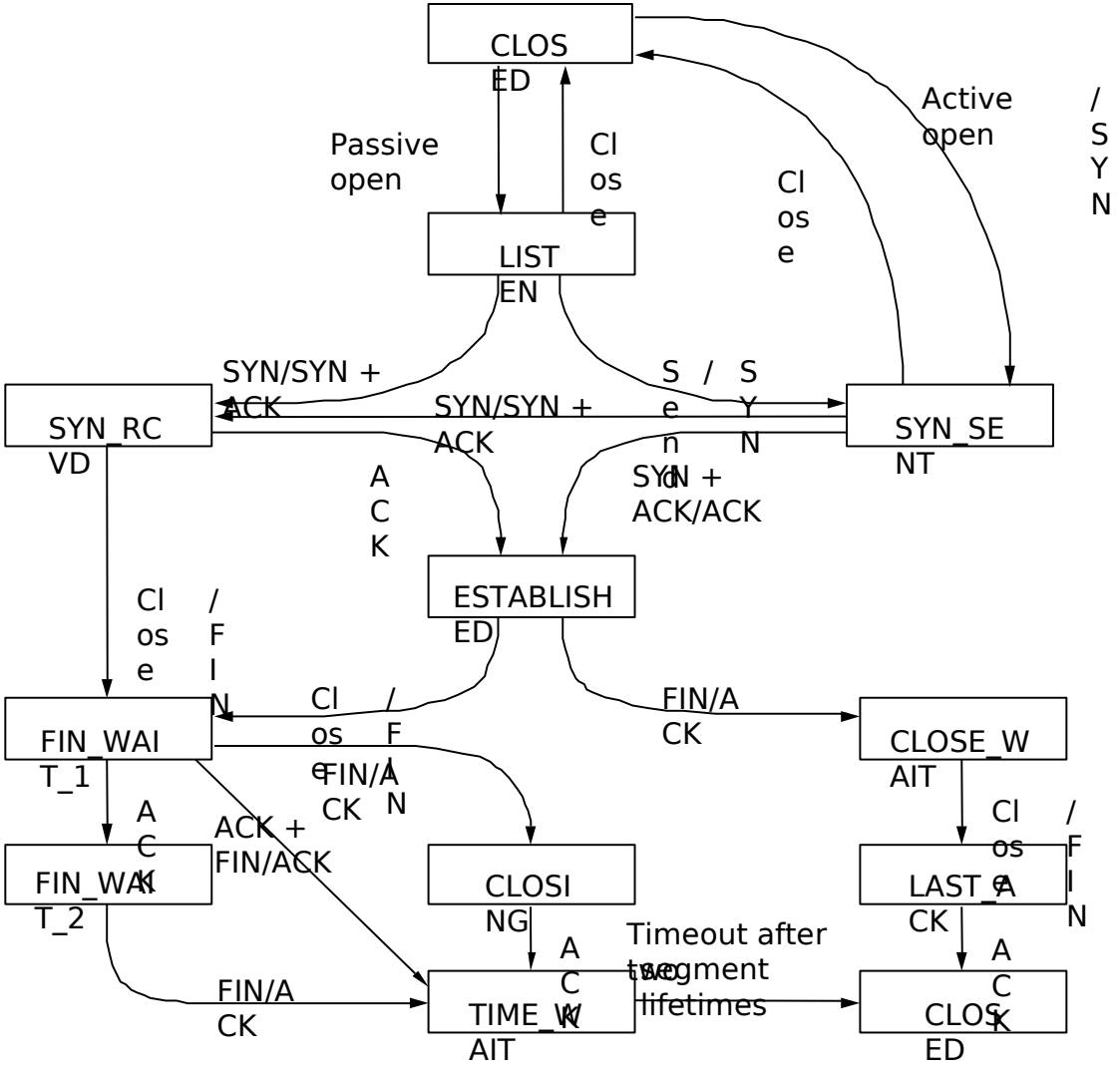


Connection set up

- Client server model

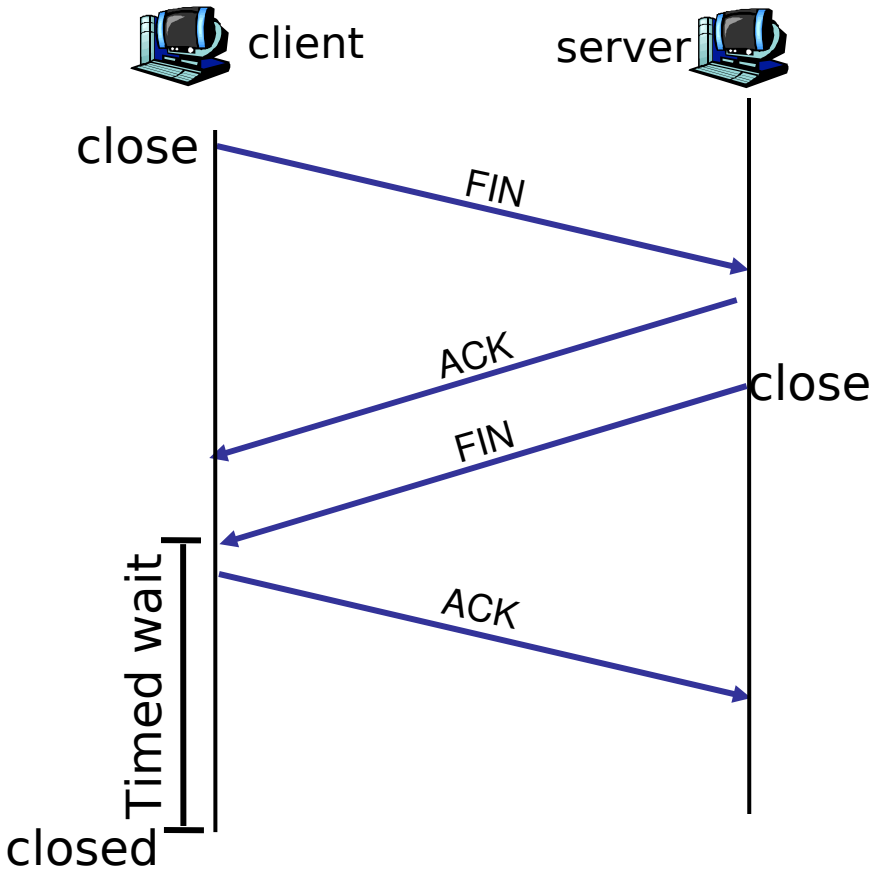


State Transition Diagram



Connection close down (e.g., client starts)

- Step 1: client transmits TCP FIN control segment to server
- Step 2: server receives FIN, replies with ACK. Shuts down connection, also sends FIN.
- Step 3: client receives FIN, replies with ACK.
  - Enters "time wait" state, ACKs received FIN segments
- step 4: server gets ACK. Connection closed



Error recovery

- TCP only uses positive ACKs
- Sender retransmits if ACK not received before timer expires (Timeout)
- Timeout set up crucially affects TCP's performance
  - If timeout value too low: segments delayed by congestion can be considered lost and retransmitted -> efficiency loss (bandwidth waste)
  - If timeout too large: reaction to true segment loss too slow -> efficiency loss (delays in data transfer)

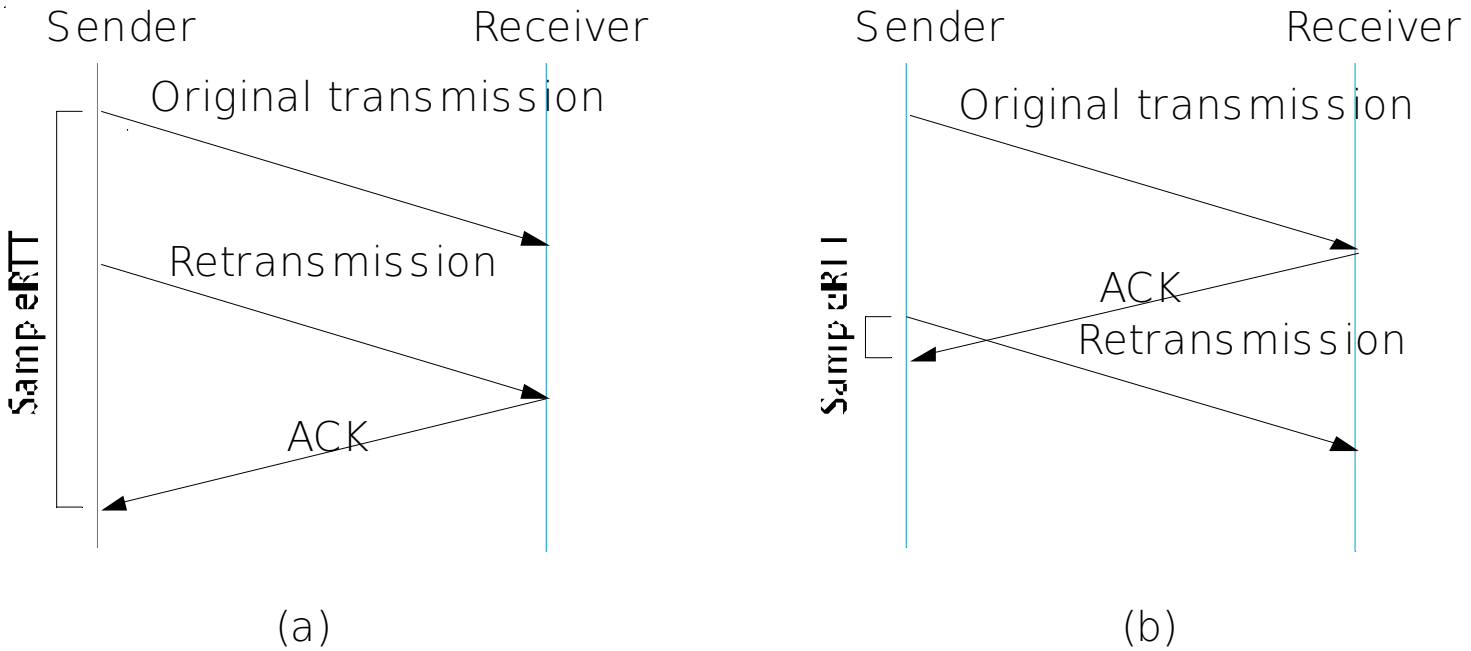
Error recovery

- **Retransmission TimeOut (RTO)** dynamically and adaptively updated
- TCP maintains dynamic estimation of Round Trip Time (RTT)
  - RTT = latency between segment transmission and corresponding ACK receipt
- RTO is larger than average estimated RTT
- RTT estimation affected by following error sources:
  - ACK transmission may not immediately follow segment receipt
  - On a retransmission, impossible to decide whether ACK refers to original transmission or retransmission
  - Congestion in the network may change dramatically over time

Adaptive Retransmission (Original Algorithm)

- Measure **SampleRTT** for each segment / ACK pair
- Compute weighted average of RTT
  - **EstRTT** = α x **EstRTT** + $(1-\alpha)$ x **SampleRTT**
  - α between 0.8 and 0.9
- Exponential Weighted Moving Average -> why ?
- Set timeout based on **EstRTT**
  - **TimeOut = 2** x **EstRTT**

Karn/Partridge Algorithm

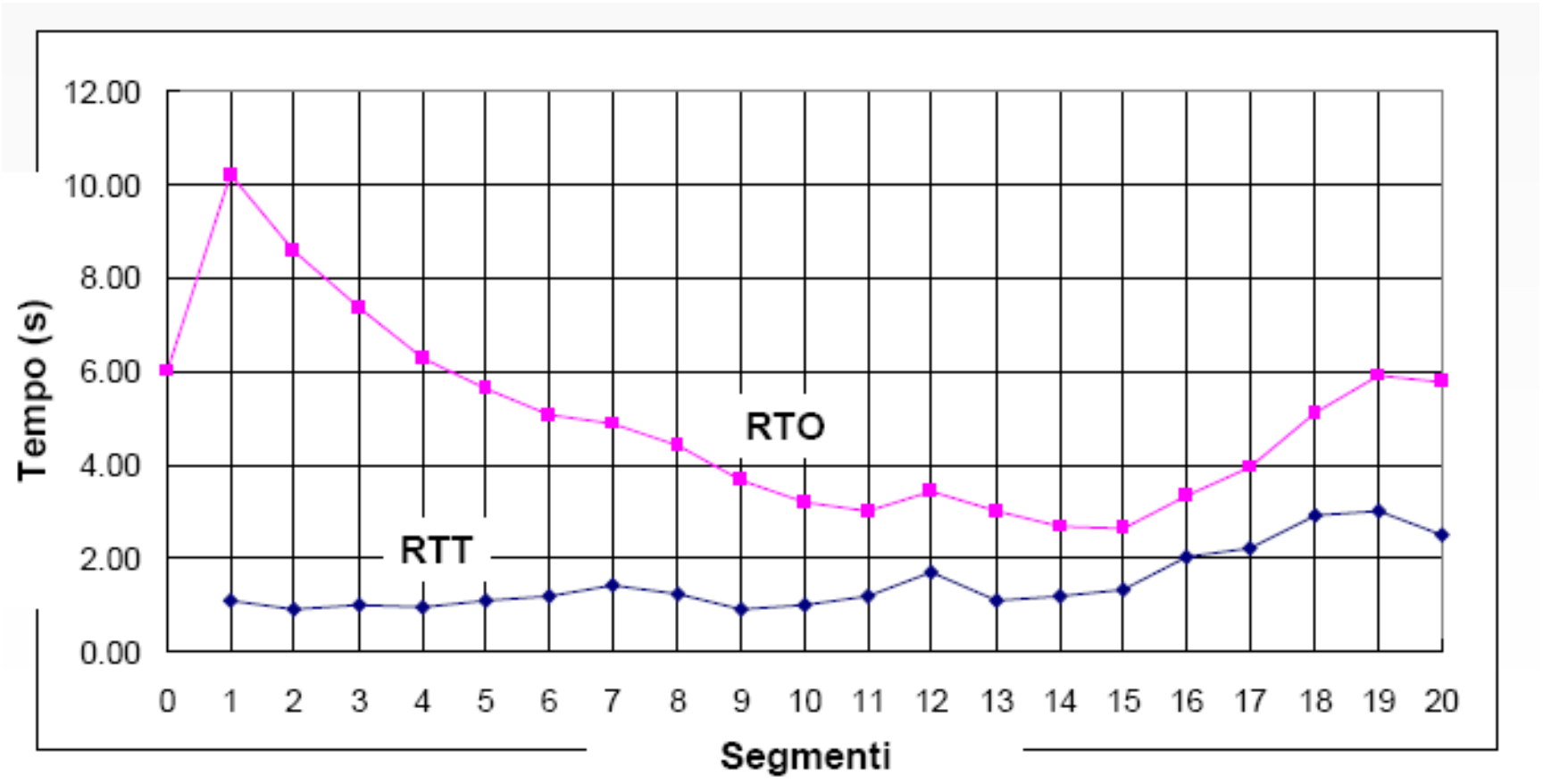


- Do not sample RTT when retransmitting
- Double timeout after each retransmission

Jacobson/ Karels Algorithm

- New Calculations for average RTT
- **Diff** = **SampleRTT** - **EstRTT**
- **EstRTT = EstRTT + (δ x Diff)**
  - **Decreases if Diff < 0**
- **Dev = Dev + δ (|Diff| - Dev)**
  - where δ is a factor between 0 and 1
- Consider variance when setting timeout value
- **TimeOut** = μ x **EstRTT** + ϕ x **Dev**
  - where μ = 1 and ϕ = 4
- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control

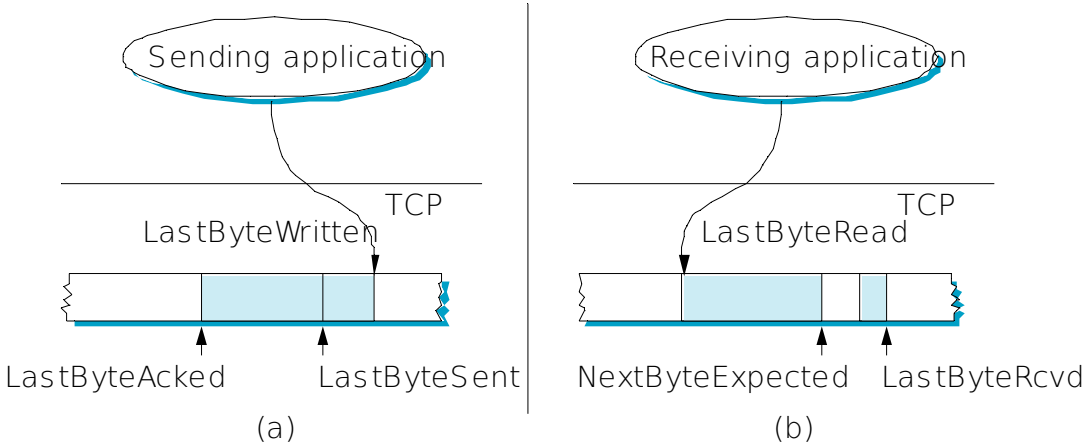
RTO calculation example



Flow control

- Constrain data transmission rate of sender
  - Necessary in Internet, due to presence of hosts with greatly varying capabilities
- TCP uses window-based flow control
  - Variable width sliding window
  - credit allocation scheme
- Basic data unit in flow control: byte
- Data bytes numbered progressively starting with initial sequence number chosen during 3-way handshaking
- Receipt of an acknowledgment (ACK Number=X and Window=W) means:
  - All bytes up to (X-1)-th are acknowledged
  - Transmitter may further transmit W bytes, i.e., all those with numbers between X and X+W-1

Sliding Window Revisited



- Sending side
  - **LastByteAcked <= LastByteSent**
  - **LastByteSent <= LastByteWritten**
  - buffer bytes between **LastByteAcked** and **LastByteWritten**
- Receiving side
  - **LastByteRead < NextByteExpected**
  - **NextByteExpected <= LastByteRcvd +1**
  - buffer bytes between **LastByteRead** and **LastByteRcvd**

Flow Control

- Send buffer size: **MaxSendBuffer**
- Receive buffer size: **MaxRcvBuffer**
- Receiving side
  - **LastByteRcvd** - **LastByteRead** < = **MaxRcvBuffer**
  - **AdvertisedWindow** = **MaxRcvBuffer** - ((**NextByteExpected - 1)** - **LastByteRead**)
- Sending side
  - **LastByteSent** - **LastByteAcked** < = **AdvertisedWindow**
  - **EffectiveWindow** = **AdvertisedWindow** - (**LastByteSent** - **LastByteAcked**)
  - **LastByteWritten** - **LastByteAcked** < = **MaxSendBuffer**
  - block sender if (**LastByteWritten** - **LastByteAcked**) + *y* > **MaxSenderBuffer**
- Always send ACK in response to arriving data segment
- Persist when **AdvertisedWindow = 0**

Congestion control

- Alleviate conditions of network overload by reducing offerred traffic [indirect, cooperative approach]
- Difficulties:
- IP provides no mechanisms for congestion detection/control
- TCP is end-to-end protocol
  - Congestion often occurs at intermediate routers
  - TCP may detect/control congestion only indirectly
- No network assistance in congestion control
  - Network does not cooperate with hosts for congestion control
- TCP entities possess only approximateknowledge of network state
  - network delays
- TCP entities compete on available network resources

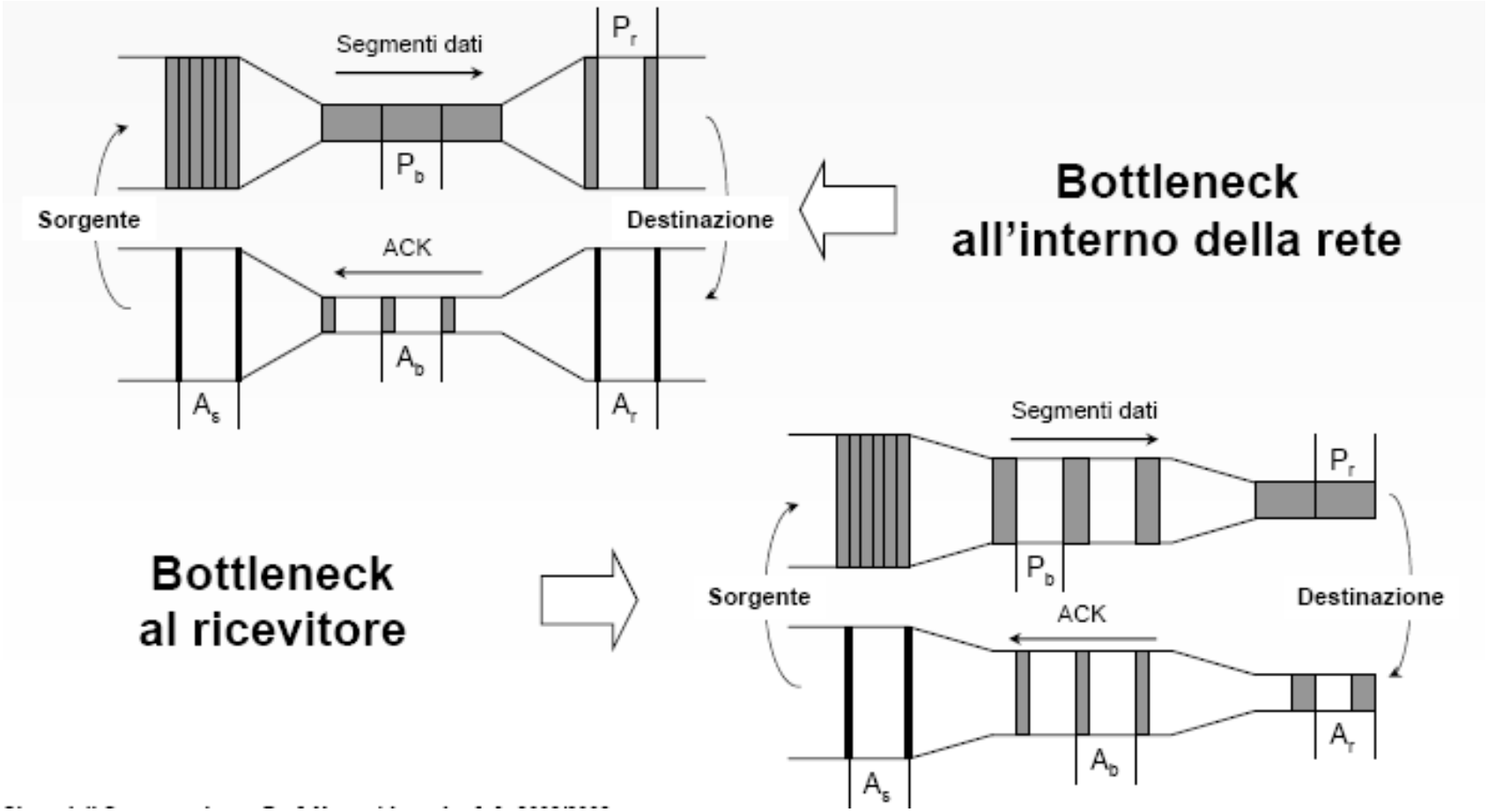
Congestion control/cont.

- When congestion occurs, window-based flow control indirectly protects network (besides receiver)
  - If network congested -> less ACKs received -> Less segments transmitted
  - Adaptive timeout estimation reduces the frequency of retransmissions under congestion, which would otherwise further worsen it
- TCP's end-to-end flow control adapts transmission rate to available byte rate at bottleneck (self-clocking property)

Congestion control/cont.

- Network bottlenecks
  - *logical*, due to one or more congested routers
  - *physical*, due to unsufficient available bandwidth in one or more physical connections (e.g., obsolete, low bandwidth cable)
  - *Receiver dependent*, caused by a receiver of low computational/memory capabilities
- TCP congestion control
  - Cannot determine causes of bottleneck
  - Cannot decide better answer to congestion
- TCP uses RTT estimation as a measure of congestion level
  - Timeout expiry is considered a congestion symptom

Congestion control



Congestion control/cont.

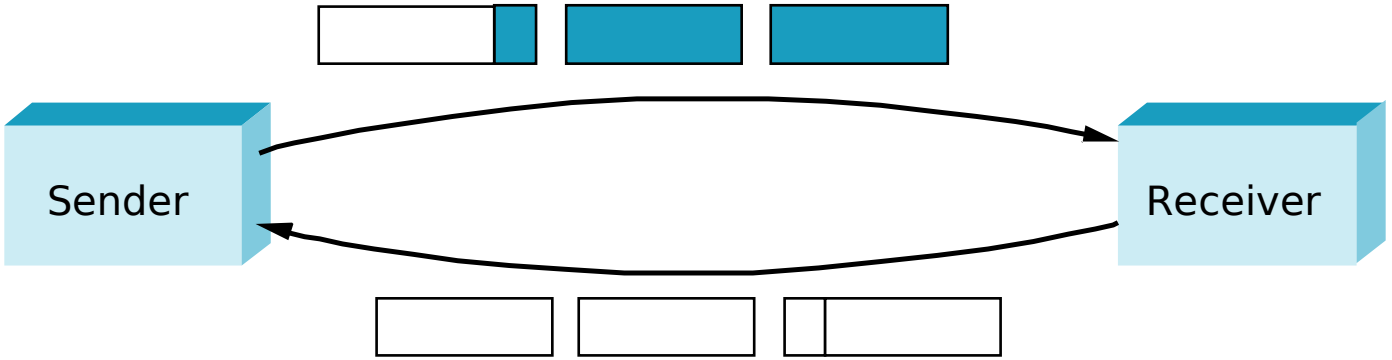
- Additional mechanisms
- different TCP implementations, all implementing (at least) slow start and congestion avoidance
  - Berkeley
  - Tahoe
  - Reno

	Meccanismo	TCP Berkeley	TCP Tahoe	TCP Reno
Window	Slow Start	◆	◆	◆
	Congestion Avoidance	◆	◆	◆
	Fast retransmit		◆	◆
	Fast recovery			◆

- Fast retransmit: retransmit upon receipt of duplicate ACKs
- Fast recovery: do not restart congestion window to 1 upon packet loss

Silly Window Syndrome

- Slow sender could transmit several TCP segment with few data [high overhead]
- Slow receiver could trigger “little” TCP segment sending once its buffer frees space
- How aggressively does sender exploit open window?



- Receiver-side solutions [Clark solution]
  - after advertising zero window, wait for space equal to a maximum segment size (MSS)
  - delayed acknowledgements
- Sender side solution [Nagle’ algorithm]

Nagle's Algorithm

- How long does sender delay sending data?
  - too long: hurts interactive applications
  - too short: poor network utilization
  - strategies: timer-based vs self-clocking
- When application generates additional data
  - if fills a max segment (and window open): send it
  - else
    - if there is unack'ed data in transit: buffer it until ACK arrives
    - else: send it

MSL Wrap Around

- 32-bit **`SequenceNum`**

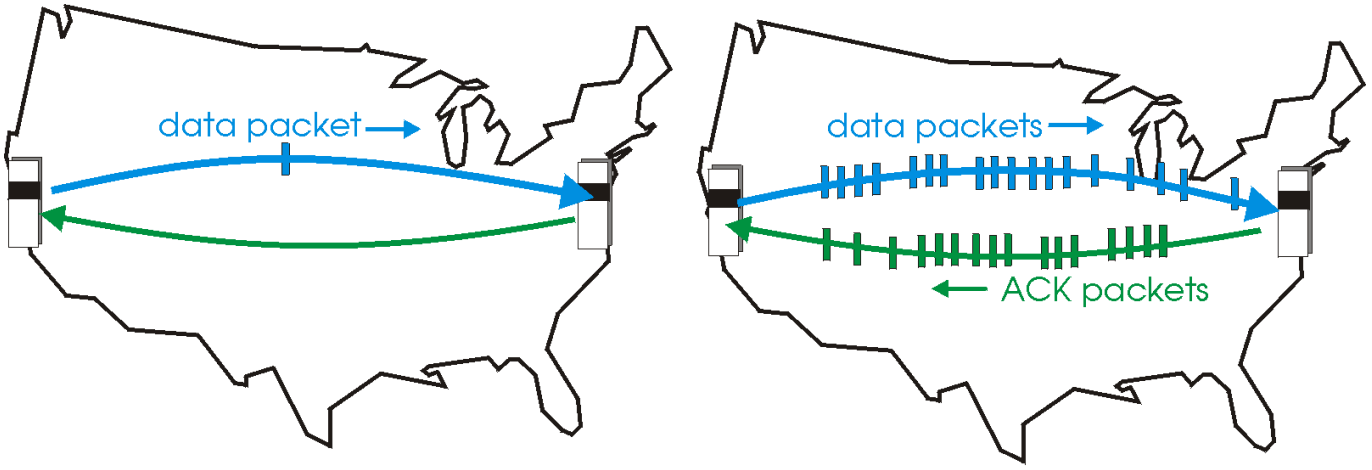
Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

Keeping the Pipe Full/1

- Delay-bandwidth product:

  B x RTT

- B is bandwidth



(a) a stop-and-wait protocol in operation

(b) a pipelined protocol in operation

Keeping the Pipe Full/2

- 16-bit **`AdvertisedWindow`**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

assuming 100ms RTT

References

- Kurose's and Ross' textbook
  - Chap.3, in particular 4.1 - 4.5
- Peterson's and Davie's textbook
  - Chap. 5, in particular 5.1 e 5.2
- TCP/IP guide
  - http://www.tcpipguide.com/free/index.htm