FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

U.PORTO

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

The Transport Layer

Redes de Computadores

2021/22

Pedro Brandão

References

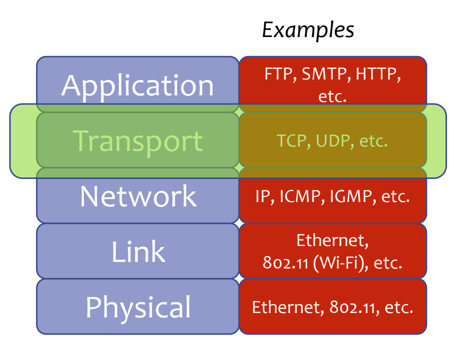
- These slides are from "Computer Networking: A Top Down Approach 5th edition. Jim Kurose, Keith Ross Addison-Wesley, April 2009"
  - With adaptations/additions by Manuel Ricardo and Pedro Brandão

Driving questions…

- *What are the services provided by the Transport Layer?*
- *What are the transport protocols in the TCP/IP stack?*
- *What are the differences between UDP and TCP?*
- *How is the connection established in TCP?*
- *What is the difference between flow control and congestion control?*
- *How does TCP implement flow control?*
- *What mechanisms does TCP adopt to prevent network congestion control?*
- *Why is it the congestion control mechanism implemented by TCP so important for the behaviour of the Internet?*

Internet protocol stack

- Application: network processes
- Transport: data transfer between processes
- Network: packet routing between source and destination
- Link: data transfer between adjacent network elements
- Physical: bits on the "wire"



Processes communicating

Process: program running within a host.

- within same host, two processes communicate using inter-process communication (defined by OS).
- processes in different hosts communicate by exchanging messages

Client process: process that actively initiates communication

Server process: process that passively waits to be contacted

Transport vs. network layer

- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

Analogy:

Mailing between companies

- processes = workers
- app messages = letters in envelopes
- hosts = buildings
- network-layer protocol = postal (CTT) service
- transport protocol = internal office mail distribution worker

RCom 21/22 - The Transport Layer

Addressing processes

- to receive messages, process must have identifier
- host device has unique 32-bit IP address
- Q: does host IP address on which process runs suffice for identifying the process?
  - A: No, many processes can be running on same host
- Identifier includes both IP address and port numbers associated with process on host.
- Example port numbers:
  - ssh server: 22
  - SMTP server: 25
    - With TLS: 465
- To connect to web server www.up.pt:
  - IP address: 104.18.6.105
  - Port: 443
    - For https, it would be 80 for http

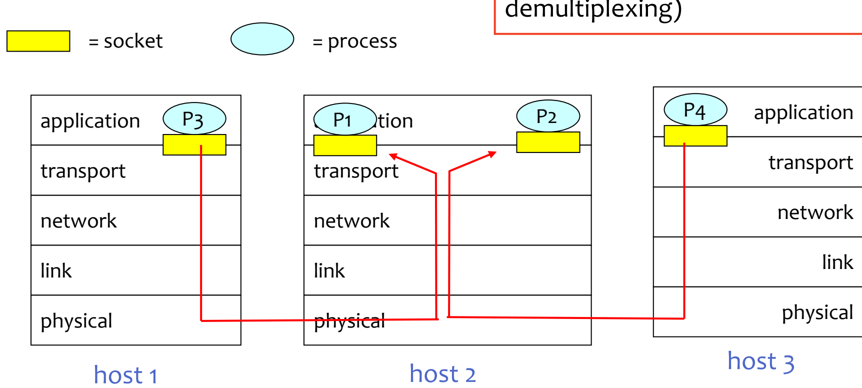
7

RCom 21/22 - The Transport Layer

Multiplexing/demultiplexing

Demultiplexing at rcv host:
delivering received segments to correct socket

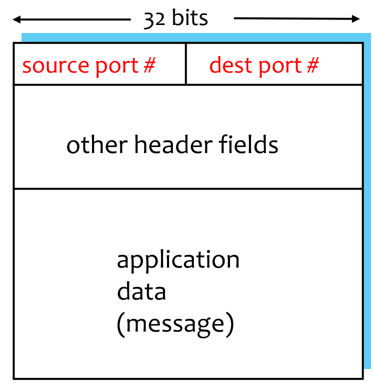
Multiplexing at send host:
gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)



8

How demultiplexing works

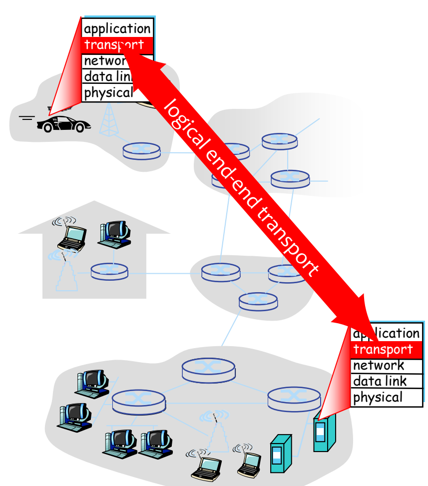
- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

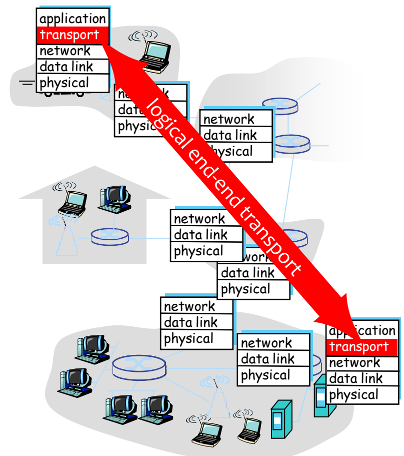
Transport services and protocols

- provide logical communication between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Internet transport protocols services

TCP service:

- connection-oriented: setup required between client and server processes
- reliable transport between sending and receiving process
- flow control: sender won’t overwhelm receiver
- congestion control: throttle sender when network overloaded
- does not provide: timing, minimum throughput guarantees, security

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security
- Q: why bother? Why is there a UDP?

UDP

RFC 768 User Datagram Protocol

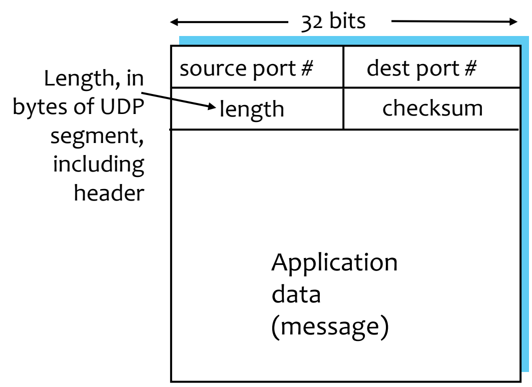
(Transport layer)

UDP - User Datagram Protocol (UDP)

- Allows applications
  - to interface directly to IP
  - with minimal additional protocol overhead
- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!

UDP segment format

- UDP header
  - Port numbers identify sending and receiving processes
  - Checksum covers header and data; optional



RCom 21/22 - The Transport Layer

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

RCom 21/22 - The Transport Layer

TCP

RFC 793 Transmission Control Protocol

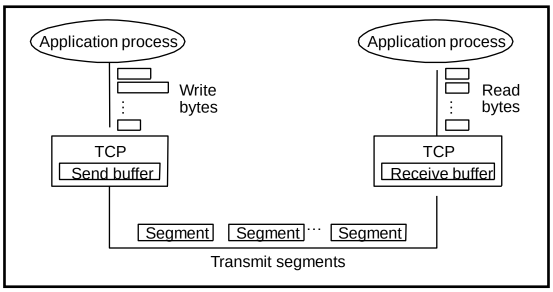
(Transport layer)

17

RCom 21/22 - The Transport Layer

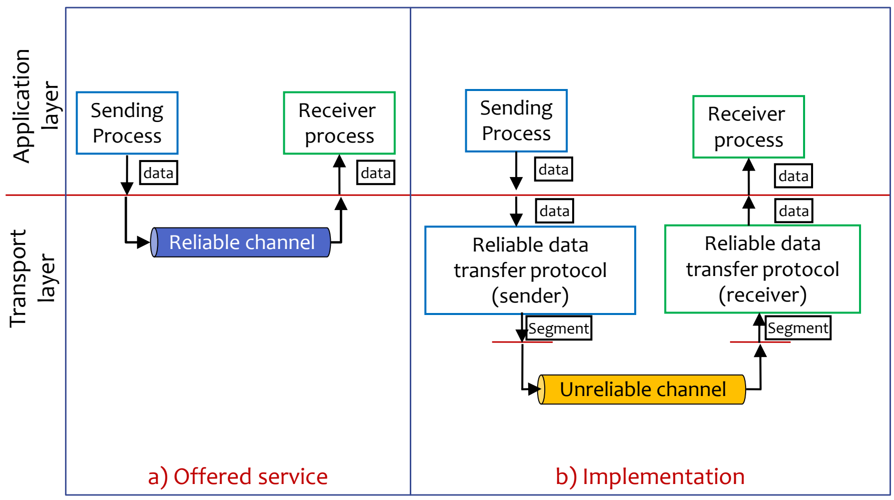
TCP – Transmission Control Protocol

- full duplex data:
  - bi-directional data flow in same connection
- connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver
  - ARQ mechanism
- Congestion control
  - Avoids network's congestion
- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*



18

Principles of reliable data transfer



Reliable transfer: problems and solutions

Packets may be lost

- Receiver confirms (ACK) reception of each segment
- If sender does not receive confirmation within a time limit ➔ re-sends segment

Packets may contain errors

- Detected using checksum
- Packets with errors are ignored ➔ handled as lost packets

Receiving duplicated segments:

- generated by the network of retransmissions by lost ACKs
- Segments are numbered
- ACK identifies the number of the last received segment
- It also solves the re-ordering problem

RCom 21/22 - The Transport Layer

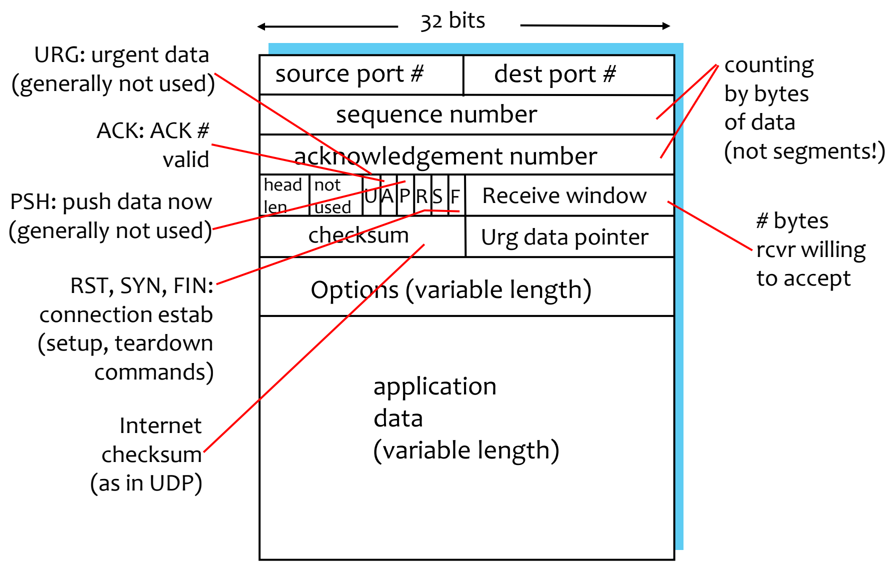
Basic TCP Operation

- Sender
  - Application data is broken in segments
  - TCP uses timer while waiting for an ACK of every segment sent
  - Un-ACKed segments are retransmitted
- Receiver
  - Errors detected using a checksum
  - Correctly received data is acknowledged
  - Segments reassembled in proper order
  - Duplicated segments discarded
- Window based flow control

21

RCom 21/22 - The Transport Layer

TCP segment structure



22

TCP Header

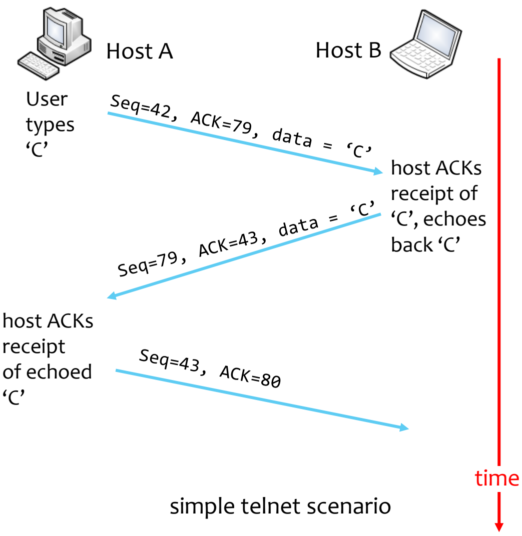
- Ports number are the same as for UDP
- 32 bit SeqNumber uniquely identifies the application data contained in the TCP segment
  - SeqNumber is in bytes
  - It identifies the first byte of data
- 32 bit AckNumber is used for piggybacking ACKs
  - AckNumber indicates the next byte the receiver is expecting
  - Implicit ACK for all of the bytes up to that point
- Window size
  - Used for flow control (ARQ) and congestion control
    - Sender cannot have more than a window of bytes in the network
  - Specified in bytes
    - Window scaling used to increase the window size in high speed networks
- Checksum covers the header and data

Sequence Numbers in TCP

- TCP regards data as a byte-stream
  - each byte in stream is numbered sequentially
- TCP breaks byte stream into segments
  - size limited by the Maximum Segment Size (MSS)
- Each packet has a sequence number
  - sequence number of the 1st byte of data transported by the segment
- TCP connection is duplex
  - data in each direction has different sequence numbers

TCP seq. #'s and ACKs

- Seq. #'s:
  - byte stream "number" of first byte in segment's data
- ACKs:
  - seq # of next byte expected from other side
  - cumulative ACK

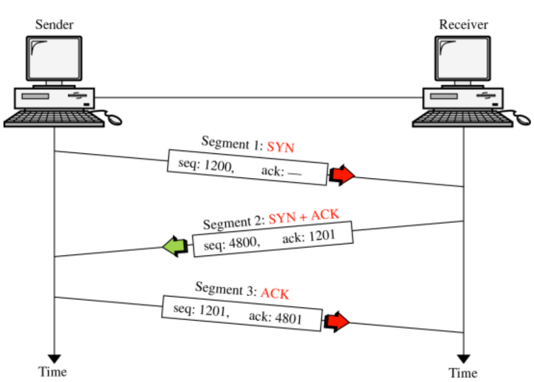


simple telnet scenario

Seq Nr and Acks

Seq. is only increased with data in the segment.
So, if host A would send another segment:
`Seq=43, data ='C'`
i.e., no increment in the Seq number.
See fig. 7 of page 31 of RFC 793. And next slide.

TCP: Connection establishment



- Necessary to exchange 3 segments to establish a TCP connection
- Flags SYN e SYN+ACK
- Define initial sequence numbers

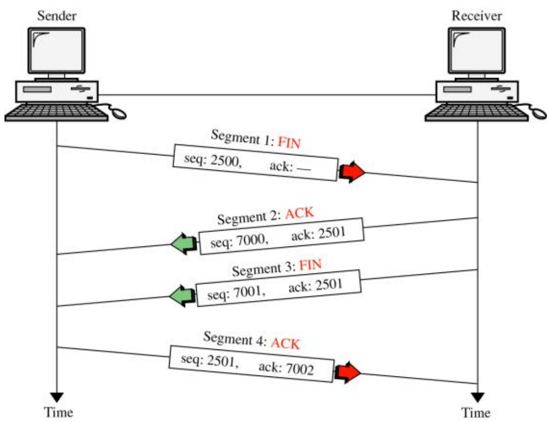
Seq Nr and Syn

Seq. is only increased with data in the segment, **except** for control segments SYN and FIN.
Therefore, in segment 3 the seq is increased.
It enables to ACK the FIN and SYN.
See page 26 of RFC 793. And next slide for the FIN scenario.

RCom 21/22 - The Transport Layer

U.PORTO

TCP: Connection close



- Necessary to exchange 4 segments to close a TCP connection
- Bidirectional flow ➔ 2 closing operations (for each direction)

27

RCom 21/22 - The Transport Layer

U.PORTO

TCP Connection Management FSM

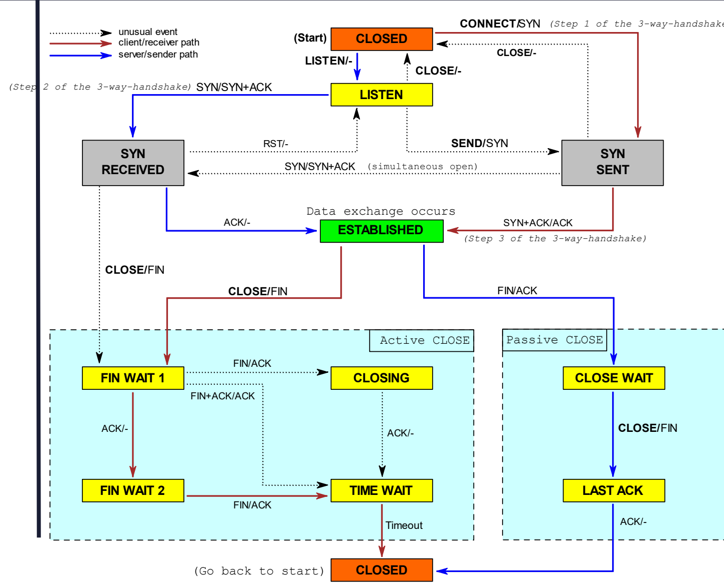
Image from wikiCommons, Sergiodc2, Marty Pauley, Scil100

The heavy solid line is the normal path for a client.

The heavy dashed line is the normal path for a server.

The light lines are unusual events.

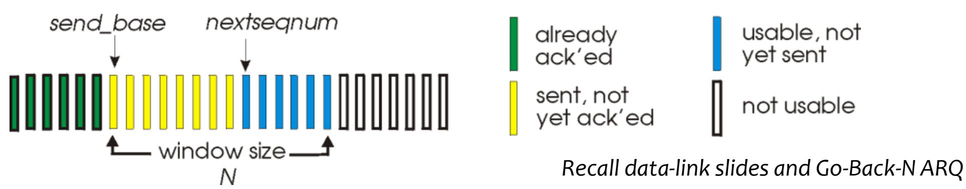
Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.



28

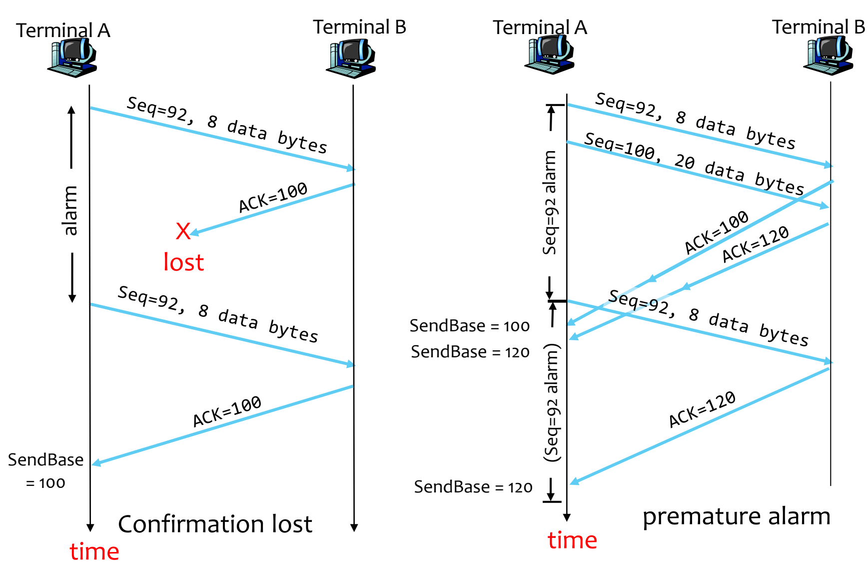
Retransmissions in TCP – A variation of Go-Back-N

- Sliding window
  - Ack contains a single sequence number
  - acknowledges all bytes with a lower sequence number
  - duplicate ACKs sent when out-of-order packet received
- Sender retransmits a single packet at a time
  - optimistic assumption → only one packet is lost
- Error control based on byte sequences, not packets

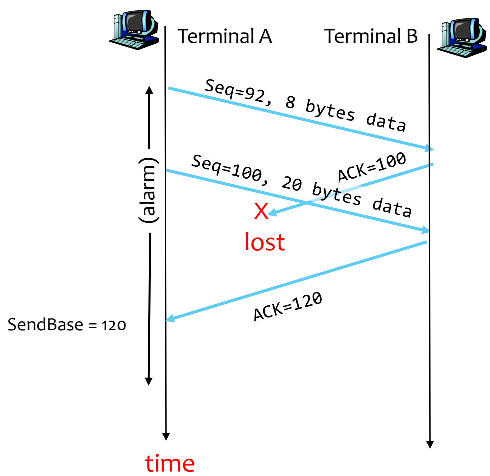


Recall data-link slides and Go-Back-N ARQ

TCP: retransmission scenario



TCP: retransmission scenarios



Cumulative confirmation

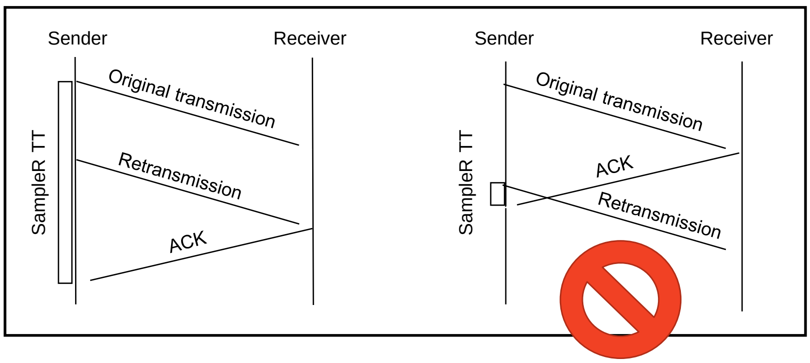
Adaptive Retransmission

- RFC 6298 - Computing TCP's Retransmission Timer
- RTT → Round Trip Time
- `sampleRTT` measured for each segment/ACK pair
- Moving average (smooth RTT) of SRTT
  - $SRTT_{new} = (1-\alpha) \times SRTT_{old} + \alpha \times sampleRTT$
    - $\alpha = 1/8$
- Variation of **RTT**
  - $RTTVAR_{new} = (1 - \beta) * RTTVAR_{old} + \beta * |SRTT_{old} - sampleRTT|$
    - $\beta = 1/4$
- RTO = SRTT + max (G, K*RTTVAR)
  - clock granularity of G seconds
  - K = 4
  - RTO - retransmission timeout

RCom 21/22 - The Transport Layer

Karn/Partridge Algorithm

- `sampleRTT` not measured in retransmission
- Timeout doubled for each retransmission



33

RCom 21/22 - The Transport Layer

Selective ACK

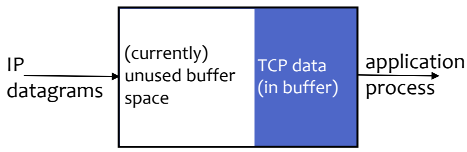
- Option for selective ACKs (SACK) also widely deployed
- Selective acknowledgement (SACK)
  - adds a bitmask of packets received
  - implemented as a TCP option
- When to retransmit?
  - packets may experience different delays
  - still need to deal with reordering
  - wait for out of order by 3 packets

34

RCom 21/22 - The Transport Layer

TCP Flow Control

- receive side of TCP connection has a receive buffer:



- app process may be slow at reading from buffer

- speed-matching service: matching send rate to receiving application's drain rate
- Receiver informs of buffer free space
- Sender limits data in transit to that window

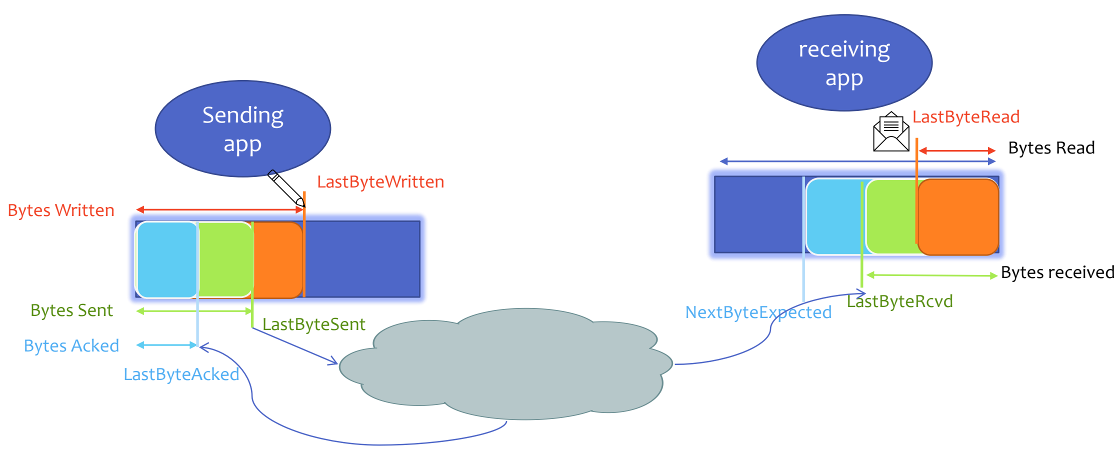
flow control

sender won't overflow receiver's buffer by transmitting too much, too fast

35

RCom 21/22 - The Transport Layer

TCP Buffers and sliding window



36

RCom 21/22 - The Transport Layer

U.PORTO

Sliding Window

- Sender
  - `LastByteAcked < = LastByteSent`
  - `LastByteSent < = LastByteWritten`
  - Buffers bytes between `LastByteAcked` and `LastByteWritten`
- Receiver
  - `LastByteRead <  NextByteExpected`
  - `NextByteExpected < = LastByteRcvd +1`
  - Buffers bytes between `LastByteRead` e `LastByteRcvd`

37

RCom 21/22 - The Transport Layer

U.PORTO

Flow Control

- Buffer length
  - Sender → `MaxSendBuffer`
  - Receiver → `MaxRcvBuffer`
- Receiver
  - `LastByteRcvd - LastByteRead < = MaxRcvBuffer`
  - `AdvertisedWindow = MaxRcvBuffer -(LastByteRcvd - LastByteRead)`
- Sender
  - `LastByteWritten - LastByteAcked < = MaxSendBuffer`
  - `LastByteSent - LastByteAcked < = AdvertisedWindow`
  - `EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)`
- Sending application blocks if it needs to write y bytes and
  - `(LastByteWritten - LastByteAcked) + y > MaxSenderBuffer`

38

RCom 21/22 - The Transport Layer

Fast Retransmit

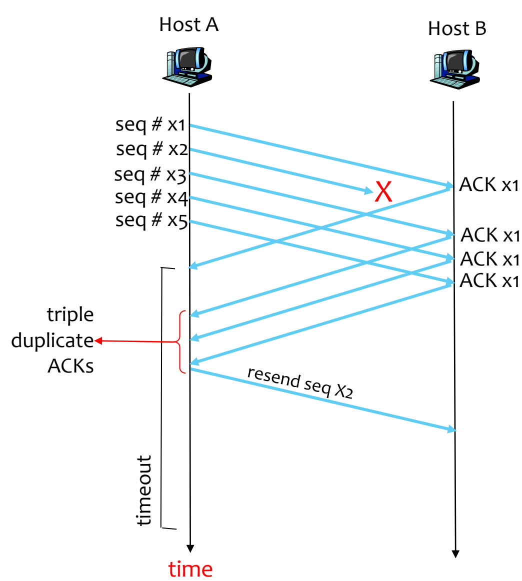
- time-out period often relatively long:
  - long delay before resending lost packet
- detect lost segments via duplicate ACKs.
  - sender often sends many segments back-to-back
  - if segment is lost, there will likely be many duplicate ACKs for that segment

- If sender receives 3 ACKs for same data, it assumes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

39

RCom 21/22 - The Transport Layer

Fast retransmit example



40

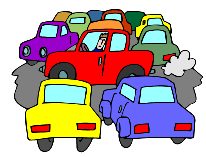
RCom 21/22 - The Transport Layer

U.PORTO

Principles of Congestion Control

Congestion:

- informally: "too many sources sending too much data too fast for network to handle"
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queuing in router buffers)



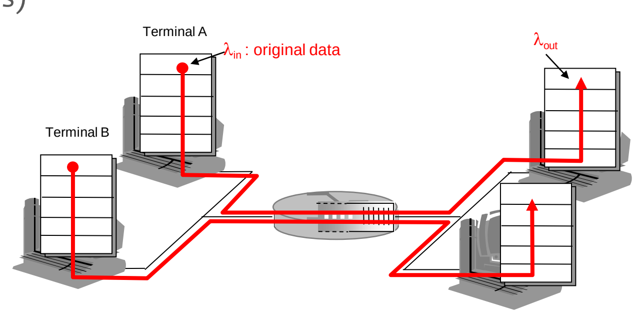
41

RCom 21/22 - The Transport Layer

U.PORTO

Congestion

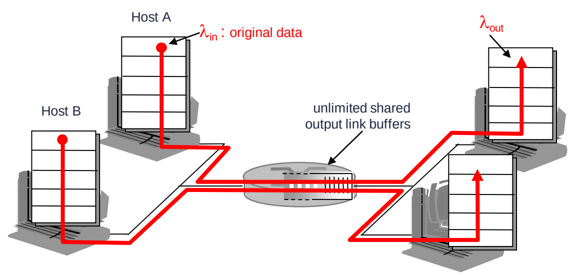
- λ_{in} → rate at which an application sends data (no header or retransmissions)
- λ_{out} → goodput
  - Data that the application receives by unit of time (no header or retransmissions)

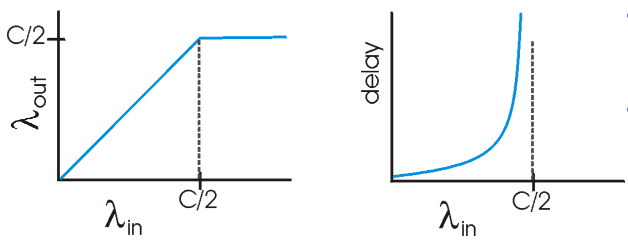


42

Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



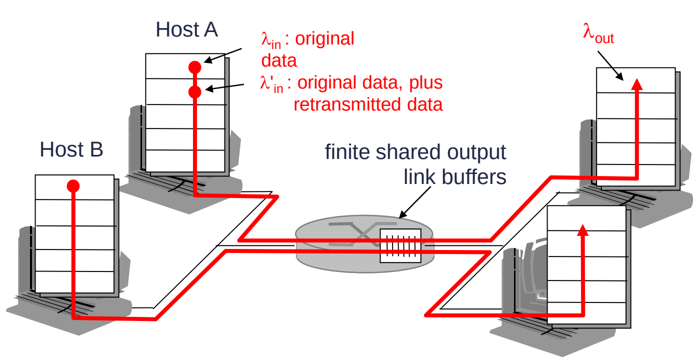


- large delays when congested
- maximum achievable throughput

43

Causes/costs of congestion: scenario 2

- one router, finite buffers
- sender retransmission of lost packet

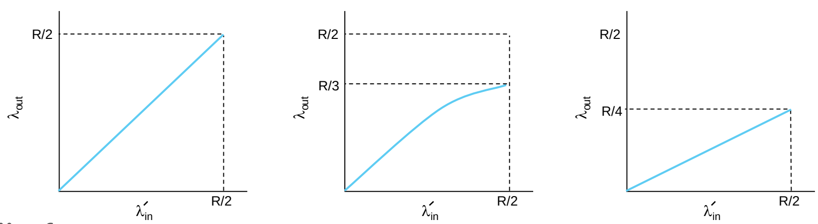


44

RCom 21/22 - The Transport Layer

Causes/costs of congestion: scenario 2

- "perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes λ'_{in} larger (than perfect case) for same λ_{out}



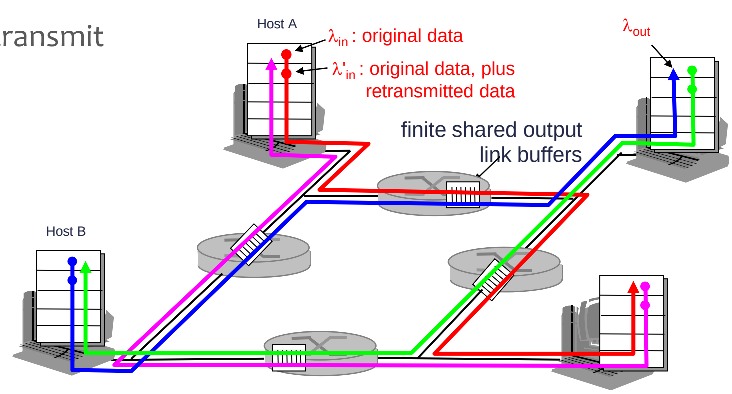
- costs" of congestion:
  - more work (retrans) for given "goodput"
  - unneeded retransmissions: link carries multiple copies of pkt

45

RCom 21/22 - The Transport Layer

Causes/costs of congestion: scenario 3

- four senders
- multihop paths
- timeout/retransmit



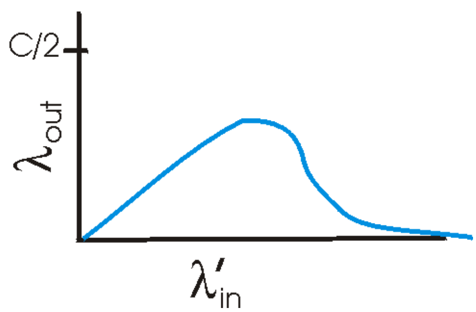
46

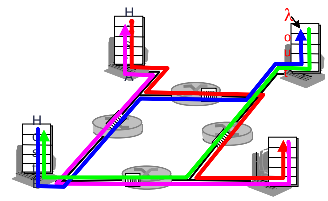
RCom 21/22 - The Transport Layer

U.PORTO

Causes/costs of congestion: scenario 3

- another "cost" of congestion:
  - when packet dropped, any "upstream transmission capacity used for that packet was wasted!





47

RCom 21/22 - The Transport Layer

U.PORTO

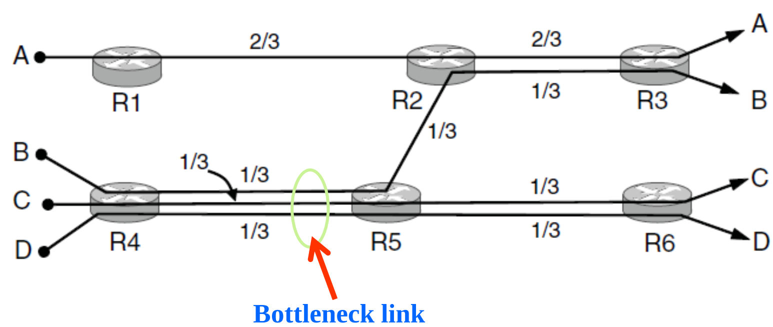
Congestion – λ_{out} decrease

- Sharing connections
- Lost packets
  - Retransmitted packets
- Delayed packets
  - Timeout and retransmission
- With multiple hops: packet drop at last hop, all previous transmissions wasted

48

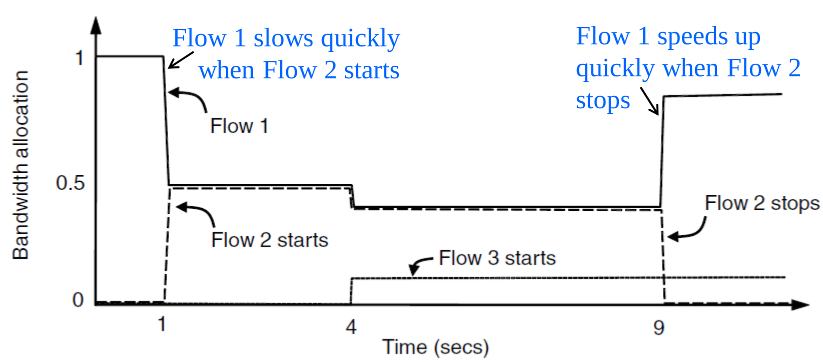
Desirable Bandwidth Allocation – Max-min fairness

Fair use gives bandwidth to all flows (no starvation)
- **Max-min fairness** gives equal shares of bottleneck



Desirable Bandwidth Allocation – Bitrates along the time

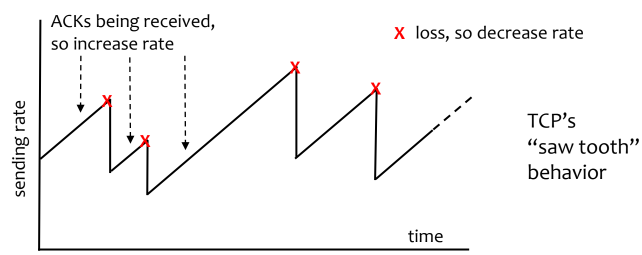
Bitrates must converge quickly when traffic patterns change



RCom 21/22 - The Transport Layer

TCP congestion control: bandwidth probing

- RFC 2581 - TCP Congestion Control
- "probing for bandwidth": increase transmission rate on receipt of ACK, until eventually loss occurs, then decrease transmission rate
  - continue to increase on ACK, decrease on loss (since available bandwidth is changing, depending on other connections in network)



51

RCom 21/22 - The Transport Layer

Additive Increase/Multiplicative Decrease

- Changes in channel capacity ➔ adjustment of transmission rate
- New variable per connection → CongestionWindow
  - limits the amount of traffic in transit
    - `MaxWin = MIN(CongestionWindow, AdvertisedWindow)`
    - `EffWin = MaxWin - (LastByteSent - LastByteAcked)`
- Bitrate (byte/s) ➔ CongestionWindow/RTT

52

Additive Increase/Multiplicative Decrease

- Algorithm
  - increases CongestionWindow by 1 segment
    - for each RTT (Round Trip Time) ➔ additive increase
  - divide CongestionWindow by 2
    - when there is a packet loss ➔ multiplicative decrease
- In practice,
  - Increases by ACK received
  - MSS → Maximum Segment Size
  - $Inc = MSS \times \frac{MSS}{CongestionWindow}$
  - `CongestionWindow`$_{new}$ `+= Inc`

If we express CongestionWindow as N × MSS we can see that Inc

$$= MSS \times \frac{MSS}{CongestionWindow}$$
$$= MSS \times \frac{MSS}{N \times MSS}$$
$$= MSS \times \frac{1}{N}$$

per ACK received

TCP Congestion Control: more details

Segment loss event: reducing cwnd

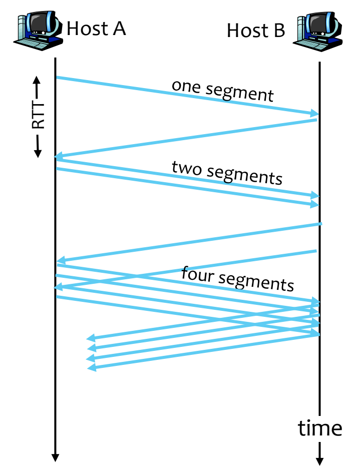
- **timeout:** no response from receiver
  - cut cwnd to 1
- **3 duplicate ACKs:** at least some segments getting through (recall fast retransmit)
  - cut cwnd in half, less aggressively than on timeout

ACK received: increase cwnd

- Slow start phase:
  - increase exponentially fast (despite name) at connection start, or following timeout
- congestion avoidance:
  - increase linearly

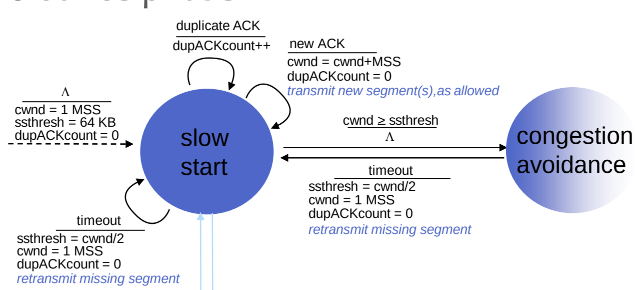
TCP Slow Start

- when connection begins, cwnd = 1 MSS
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate
- increase rate exponentially until first loss event or when threshold reached
  - double cwnd every RTT
  - done by incrementing cwnd by 1 for every ACK received



Transitioning into/out of slowstart

- `ssthresh:` `cwnd` threshold maintained by TCP
- on loss event (timeout): set `ssthresh` to `cwnd/2`
  - remember (half of) TCP rate when congestion last occurred
- when `cwnd >= ssthresh`: transition from slow start to congestion avoidance phase



RCom 21/22 - The Transport Layer

U.PORTO

TCP: congestion avoidance

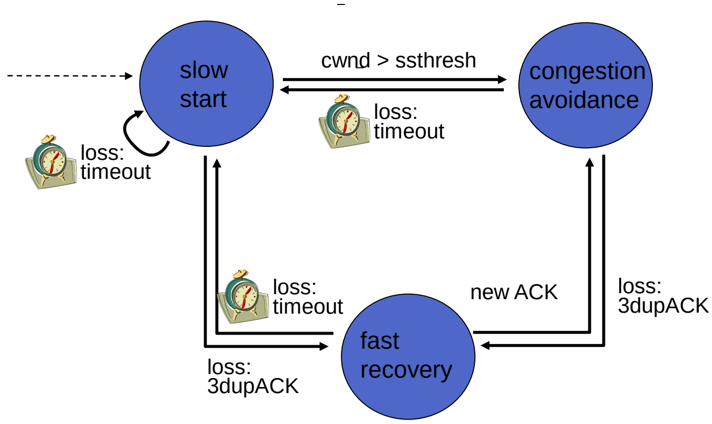
- when **cwnd > ssthresh** grow **cwnd** linearly
  - increase **cwnd** by 1 MSS per RTT
  - approach possible congestion slower than in slow start
  - implementation: **cwnd = cwnd + MSS * (MSS/cwnd)** for each ACK received

57

RCom 21/22 - The Transport Layer

U.PORTO

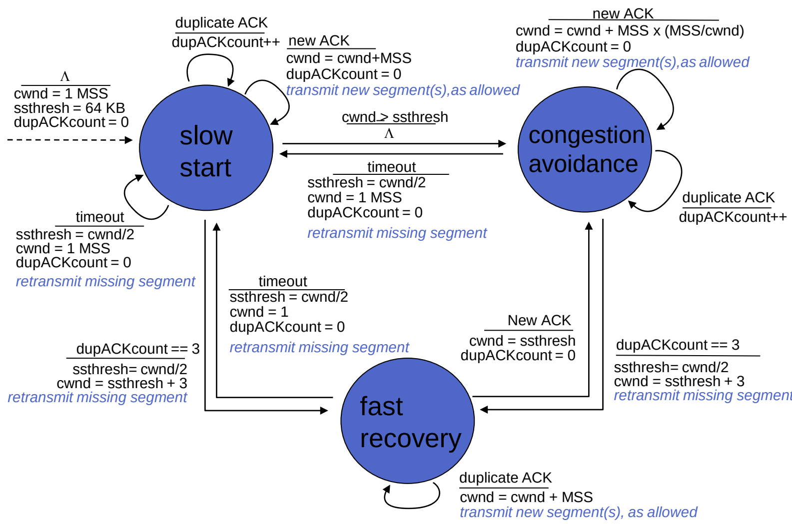
TCP congestion control FSM: overview



58

RCom 21/22 - The Transport Layer

TCP congestion control FSM: details

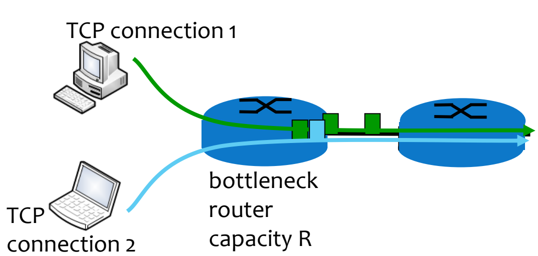


59

RCom 21/22 - The Transport Layer

TCP Fairness

fairness goal: if **K** TCP sessions share same bottleneck link of bandwidth **R**, each should have average rate of **R/K**



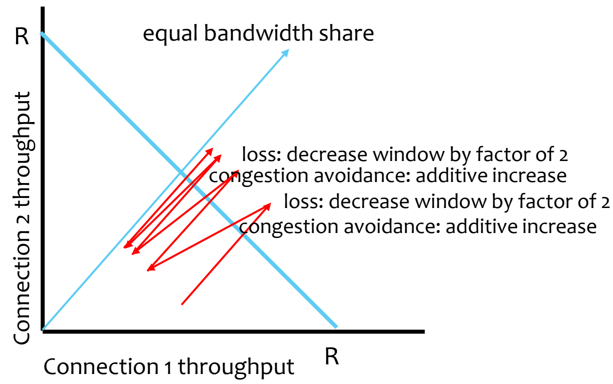
60

RCom 21/22 - The Transport Layer

Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughput increases
- multiplicative decrease decreases throughput proportionally



61

RCom 21/22 - The Transport Layer

Fairness (more)

Fairness and UDP

- multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- web browsers do this
- example: link of rate R with already 9 TCP connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets ~R/2 !

62

Homework

1. Review slides

2. Read from Tanenbaum
   - Chapter 6 – The Transport Layer

3. Answer questions at Moodle

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

End of Transport Layer