INP Grenoble DÉPARTEMENT TÉLÉCOMMUNICATIONS

Computer Networking

TCP

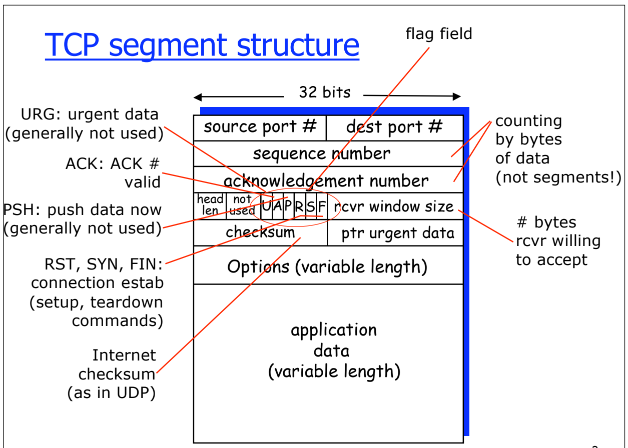
Prof. Andrzej Duda
duda@imag.fr

http://duda.imag.fr

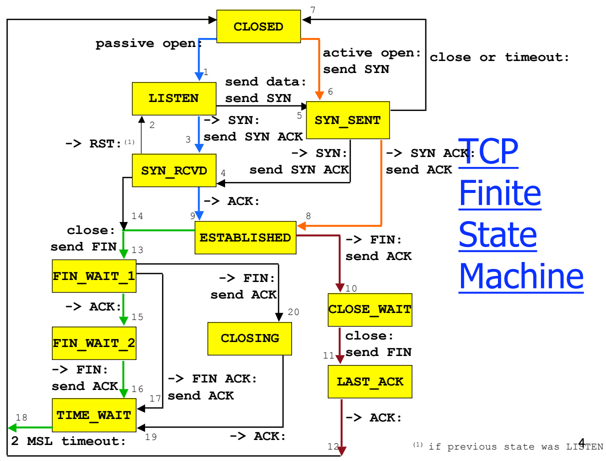
Details of the TCP protocol

- More details on TCP
  - connection management
  - reliable transfer
  - interactive traffic
  - Nagle algorithm
  - silly window syndrome
  - RTT estimation and Karn's rule
  - fast retransmit
  - congestion control

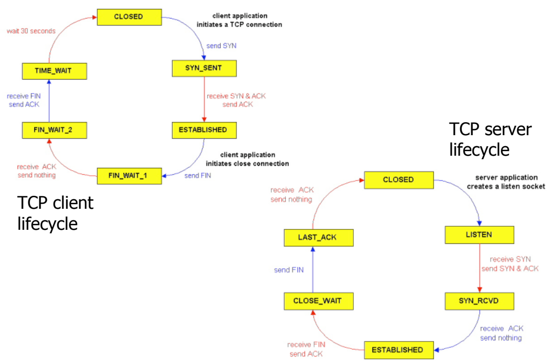
TCP segment structure



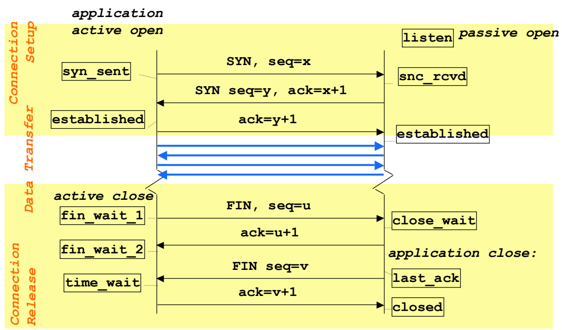
TCP Finite State Machine



TCP Connection Management (cont)



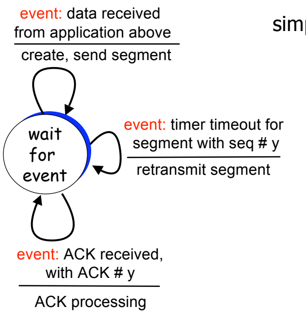
TCP Connection Phases



TCP: reliable data transfer

simplified sender, assuming

- one way data transfer
- no flow, congestion control



TCP: reliable data transfer

Simplified TCP sender

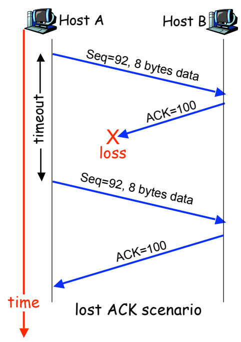
```
sendbase = initial_sequence number
nextseqnum = initial_sequence number

 loop (forever) {
  switch(event)
  event: data received from application above
       create TCP segment with sequence number nextseqnum
       start timer for segment nextseqnum
       pass segment to IP
       nextseqnum = nextseqnum + length(data)
  event: timer timeout for segment with sequence number y
       retransmit segment with sequence number y
       compute new timeout interval for segment y
       restart timer for sequence number y
  event: ACK received, with ACK field value of y
       if (y > sendbase) { /* cumulative ACK of all data up to y */
          cancel all timers for segments with sequence numbers < y
          sendbase = y
          }
       else { /* a duplicate ACK for already ACKed segment */
          increment number of duplicate ACKs received for y
          if (number of duplicate ACKS received for y == 3) {
             /* TCP fast retransmit */
             resend segment with sequence number y
             restart timer for segment y
          }
 }  /* end of loop forever */
```


TCP ACK generation [RFC 1122, RFC 2581]

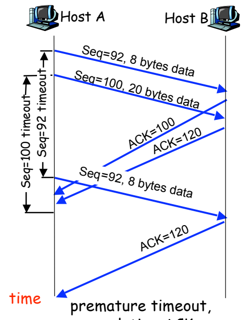
Event	TCP Receiver action
in-order segment arrival, no gaps, everything else already ACKed	delayed ACK. Wait up to 200ms for next segment. If no next segment, send ACK
in-order segment arrival, no gaps, one delayed ACK pending	immediately send single cumulative ACK
out-of-order segment arrival higher-than-expect seq. # gap detected	send duplicate ACK, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate ACK if segment starts at lower end of gap

Duplicated ACKs can be used for Fast Retransmission

TCP: retransmission scenarios: GBN + SR

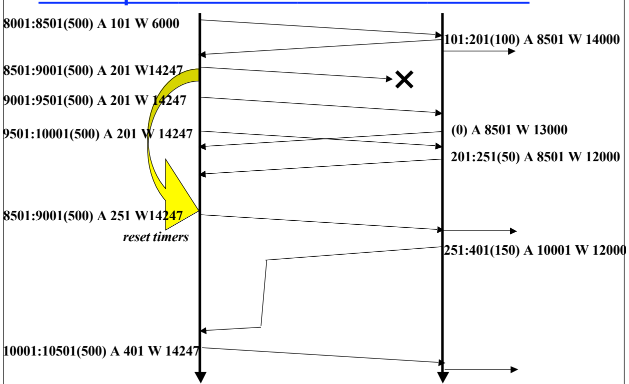


lost ACK scenario

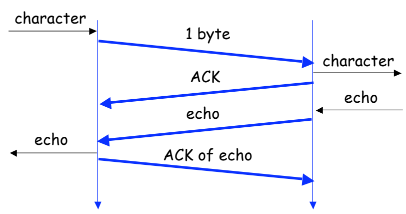


premature timeout, cumulative ACKs

Example of data transfer - Reno

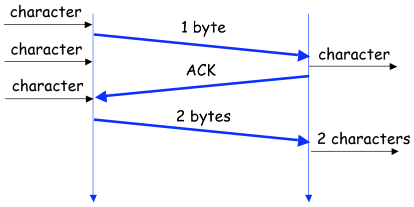


Interactive traffic



- Delayed ACK
  - ACK et echo in the same segment
  - 200 ms delay: ACK sent with echo character

Nagle algorithm



- Sender may only send one small no acknowledged segment - tinygram (small = smaller than MSS)
  - avoid sending small segments on the network - large overhead
  - Nagle algorithm can be disabled by application (TCP_NODELAY socket option):
    - X Window

13

Silly Window syndrome

- Small advertised window

	←	Ack 0 W 2000
0:1000	→	buf = 2000, freebuf = 1000
1000:2000	→	freebuf = 0
	←	Ack 2000 W 0
		appl lit 1 octet : freebuf = 1
	←	Ack 2000 W 1
2000:2001	→	freebuf = 0
		appl lit 1 octet : freebuf = 1
	←	Ack 2001 W 1
2001:2002	→	freebuf = 0

14

Silly Window syndrome

- Sender has a lot of data to send
- Small advertised window forces to send small segments
- Solution at receiver
  - advertise window by large chunks: min (MSS, 1/2 **RcvBuffer** size)
- Solution at sender
  - delay sending small segments: send at least min (MSS, 1/2 maximum **RcvWindow**)

15

TCP Round Trip Time and Timeout

Q: how to set TCP timeout value - RTO?

- RTO: Retransmission Timeout
- longer than RTT
  - note: RTT will vary
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

Q: how to estimate RTT?

- **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions, cumulatively ACKed segments
- **SampleRTT** will vary, want estimated RTT "smoother"
  - use several recent measurements, not just current **SampleRTT**

16

TCP Round Trip Time and Timeout

```
EstimatedRTT = (1-x)*EstimatedRTT + x*SampleRTT
```

- Exponential weighted moving average
- influence of given sample decreases exponentially fast
- typical value of x: 0.125

Setting the timeout

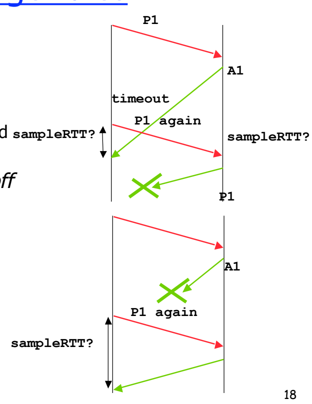
- **EstimatedRTT** plus "safety margin"
- large variation in **EstimatedRTT** -> larger safety margin

```
RTO = EstimatedRTT + 4*Deviation

Deviation = (1-x)*Deviation +
            x*|SampleRTT-EstimatedRTT|
```

17

Karn and Partridge rules

- Do not measure RTT if retransmission
  - is it the ACK for the first transmission or the second one?
- *Timer exponential backoff*
  - double RTO at each retransmission



18

Beginning of the connection

- SYN segment timeout
- 1st
  - D = 3, RTT = 0
  - RTO = RTT + 2 * D = 0 + 2 * 3 = 6 s
- 2nd
  - RTO = RTT + 4 * D = 12s
  - apply exp. backoff -> 24 s
- 3rd
  - apply exp. backoff -> 48 s
- 6, 24, 48, then drop
  - max. 75 s
- Implementation dependent

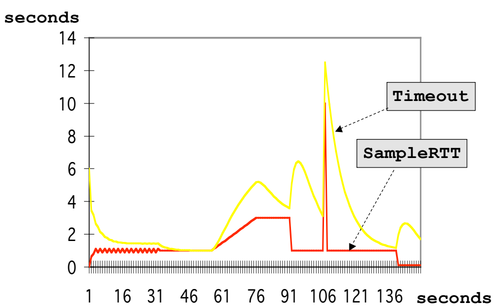
19

Data packets

- 1st
  - RTO = 1.5 s (3 ticks)
- 2nd
  - apply exp. backoff -> 3 s
- 7th
  - apply exp. backoff -> 64 s
- nth
  - max (64, 2xRTO)
- 13th
  - drop
- Total time
  - 542,5s = 9 minutes

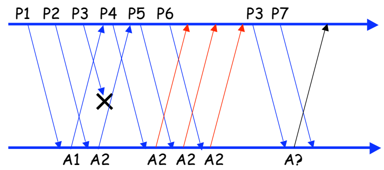
20

A Simulation of RTO



21

Fast Retransmit



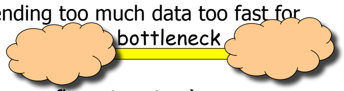
- Fast retransmit
  - timeout may be large
  - add the Selective Repeat behavior
  - if the sender receives 3 **duplicated ACKs**, retransmit the missing segment

22

Congestion Control

Congestion:

- "too many sources sending too much data too fast for *network* to handle"



- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)

Two broad approaches towards congestion control:

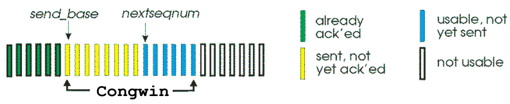
End-end congestion control:
- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

Network-assisted congestion control:
- routers provide feedback to end systems
  - single bit indicating congestion
  - explicit rate sender should send at

23

TCP Congestion Control

- end-end control (no network assistance)
- transmission rate limited by congestion window size, `Congwin`, over segments:



- w segments, each with MSS bytes sent in one RTT:

$$\text{throughput} = \frac{w * MSS}{RTT} \text{ Bytes/sec}$$

24

TCP congestion control:

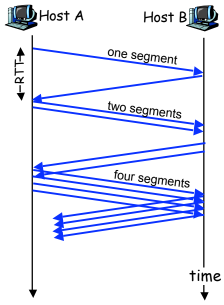
- "probing" for usable bandwidth:
  - ideally: transmit as fast as possible (`Congwin` as large as possible) without loss
  - *increase* `Congwin` until loss (congestion)
  - loss: *decrease* `Congwin`, then begin probing (increasing) again
- two "phases"
  - slow start
  - congestion avoidance
- important variables:
  - `Congwin`
  - `threshold`: defines threshold between slow start phase and congestion avoidance phase

25

TCP Slowstart

Slowstart algorithm

initialize: Congwin = 1
for (each ACK)
    Congwin++
until (loss event OR
    CongWin > threshold)



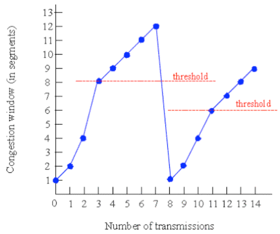
- exponential increase (per RTT) in window size (not so slow!)
- loss event: timeout (Tahoe TCP) and/or or three duplicate ACKs (Reno TCP)

26

TCP Congestion Avoidance

Congestion avoidance

/* slowstart is over */
/* Congwin > threshold */
Until (loss event) {
  every w segments ACKed:
    Congwin++
  }
threshold = Congwin/2
Congwin = 1
perform slowstart[1]



1: TCP Reno skips slowstart (fast recovery) after three duplicate ACKs

27

TCP Fairness

Fairness goal: if N TCP sessions share same bottleneck link, each should get 1/N of link capacity

WHY?

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

TCP congestion avoidance:

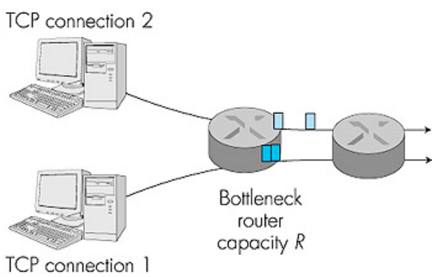
- AIMD: *additive increase, multiplicative decrease*
  - increase window by 1 per RTT
  - decrease window by factor of 2 on loss event

28

Why is TCP fair?

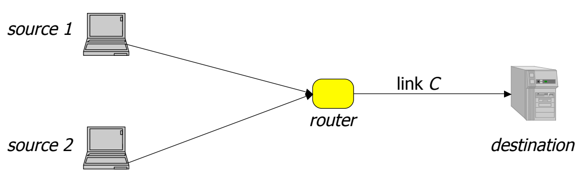
Two competing sessions:

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



29

Why AI-MD works?

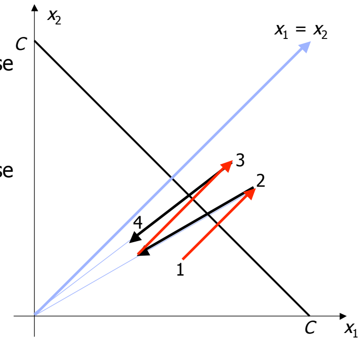


- Simple scenario with two sources sharing a bottleneck link of capacity *C*

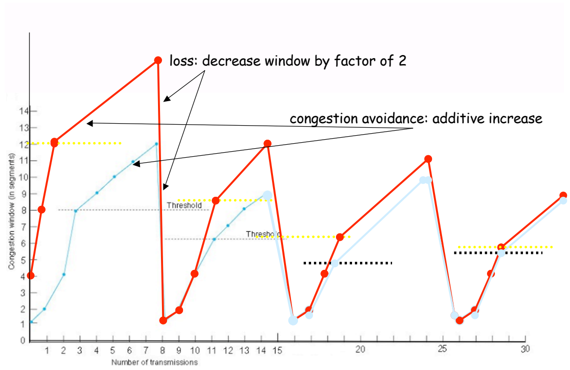
30

Throughput of sources

1. Additive increase
2. Multiplicative decrease
3. Additive increase
4. Multiplicative decrease



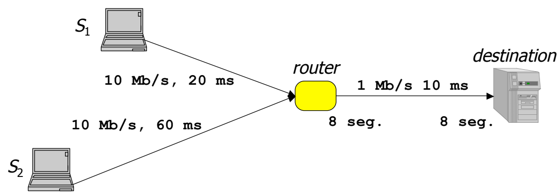
TCP Fairness



Fairness of the TCP

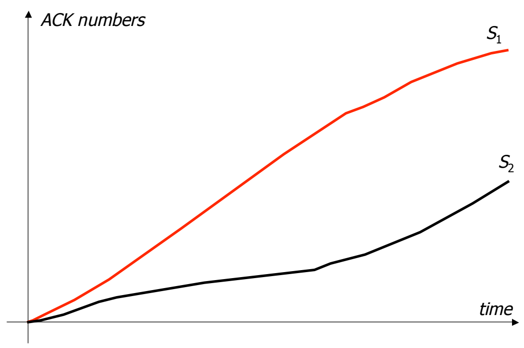
- TCP differs from the pure AI-MD principle
  - window based control, not rate based
  - increase in rate is not strictly additive - window is increased by 1/W for each ACK
- Adaptation algorithm of TCP results in a negative bias against long round trip times
  - adaptation algorithm gives less throughput to sources having larger RTT

Fairness of TCP



- Example network with two TCP sources
  - link capacity, delay
  - limited queues on the link (8 segments)
- NS simulation

Throughput in time



UDP: User Datagram Protocol [RFC 768]

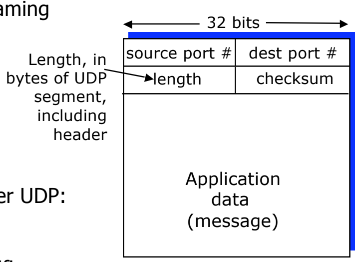
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

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

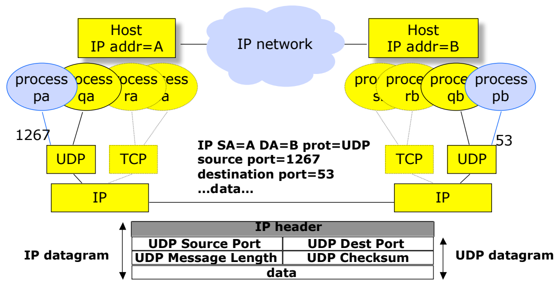
UDP: more

- 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 recover!



UDP segment format

End to end UDP communication



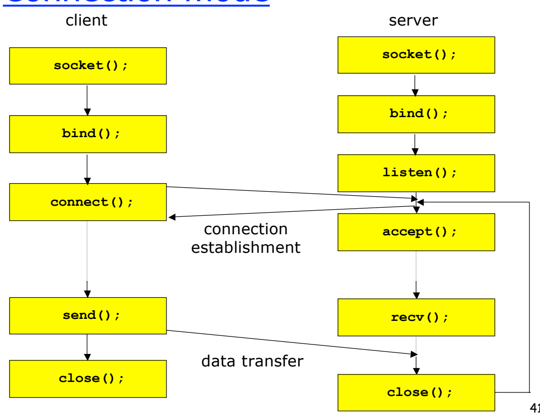
Sockets

- Interface between applications and the transport layer protocols
  - socket - communication end-point
  - network communication viewed as a file descriptor (socket descriptor)
- Two main types of sockets
  - connectionless mode (or datagram, UDP protocol)
  - connection mode (or stream, TCP protocol)

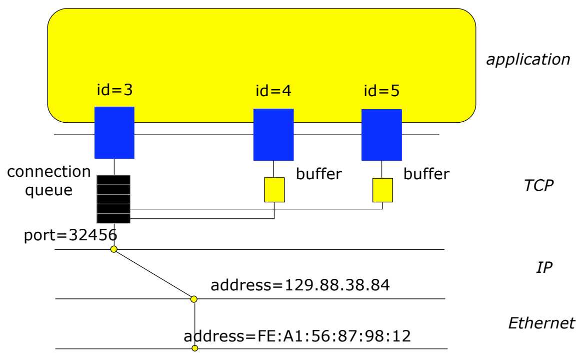
Connection mode

- System calls in connection mode (TCP)
  - `socket` - create a socket descriptor
  - `bind` - associate with a local address
  - `listen` - signal willingness to wait for incoming connections (S)
  - `accept` - accept a new incoming connection (S)
  - `connect` - ask to establish a new connection (C)
  - `send` - send a buffer of data
  - `recv` - receive data
  - `close` - close socket descriptor

Connection mode



Connection mode

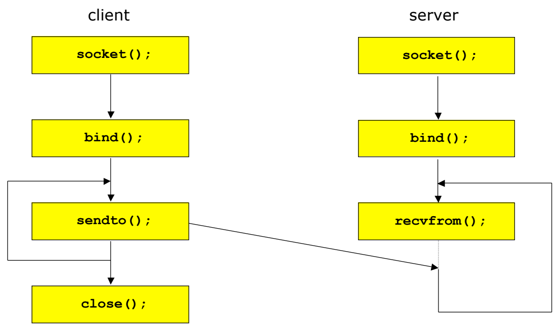


Connectionless mode

- System calls in connectionless mode (UDP)
  - `socket` - create a socket descriptor
  - `bind` - associate with a local address
  - `sendto` - send a buffer of data
  - `recvfrom` - receive data
  - `close` - close socket descriptor

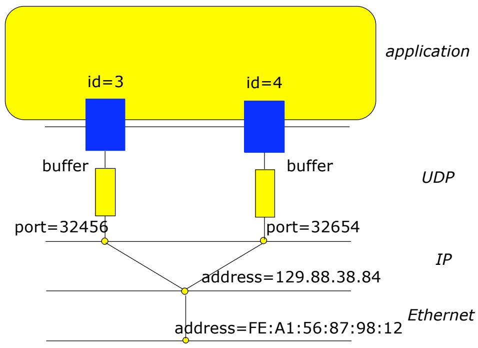
43

Connectionless mode



44

Connectionless mode



45

Summary

- TCP protocol is complex!
  - connection management
  - reliable transfer
  - interactive traffic, Nagle algorithm
  - silly window syndrome
  - RTT estimation and Karn's rule
  - fast retransmit
  - congestion control
- UDP is simple
  - adds multiplexing to IP datagrams
  - used by RTP/RTCP for multimedia streaming
- Sockets - application interface to network communication
  - connection sockets (TCP)
  - connectionless sockets (UDP)

46