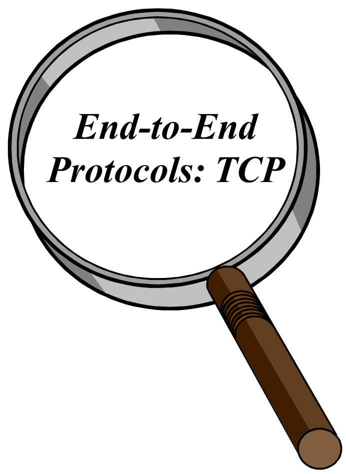
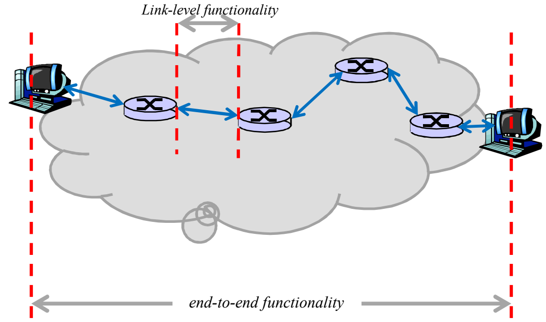


End-to-End vs. Link-by-Link

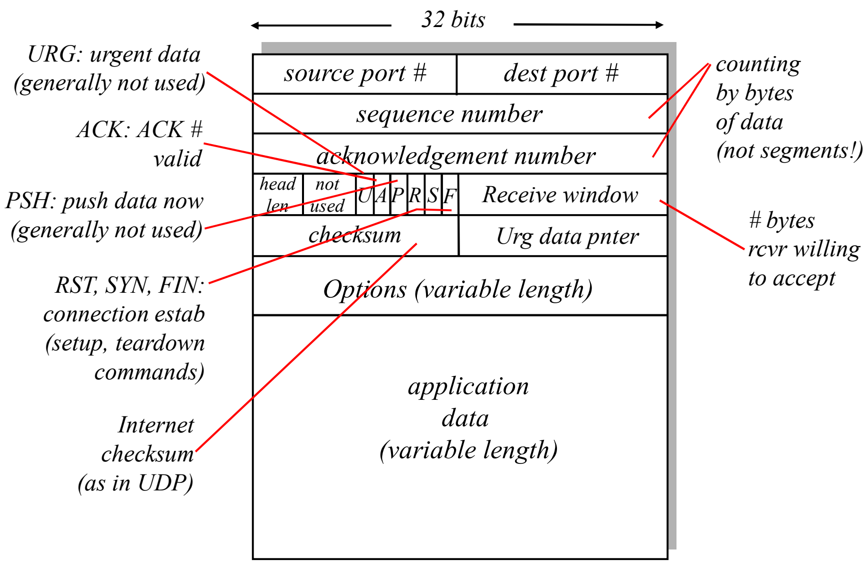
- Can end-to-end reliability be implemented through a link-by-link mechanisms?
- If so, why do we need end-to-end protocols?
- If not, then why do we need link-by-link functionality?



TCP: Overview

- End-to-end issues for providing end-to-end reliable and ordered data transfer
  - Logical connection between two remote hosts
  - RTT is not fixed, even during a connection
  - Packet re-ordering is an issue
    - MSL: Maximum Segment Lifetime
  - Resource discovery
    - Available bandwidth
    - Buffer space
  - Congestion issues in the network

TCP segment format



TCP services/components

- Reliability
  - Sequence numbers and ACKs
  - Time out mechanism
- Flow control
- Connection management
- Congestion control

Copyright ©by Zygmunt J. Haas, 2017

TCP seq. numbers, ACKs

sequence numbers:

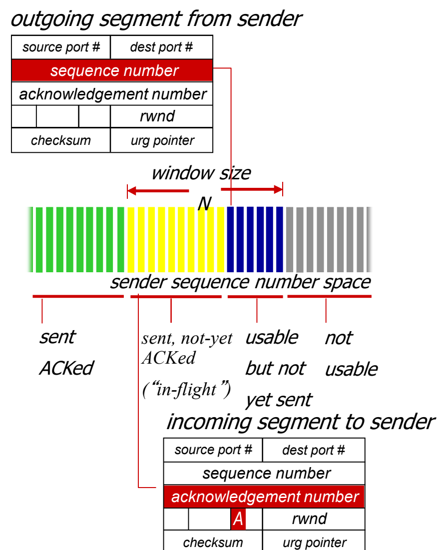
- byte stream "number" of first byte in segment's data

acknowledgements:

- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementor



Copyright ©by Zygmunt J. Haas, 2017

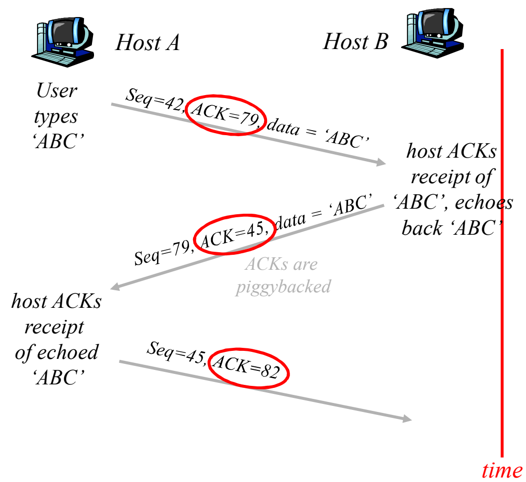
TCP seq. #'s and ACKs

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

TCP Round Trip Time and Timeout

Q: how to set TCP timeout value?

- longer than RTT
  – but RTT varies
- too short: premature timeout
  – unnecessary retransmissions
- too long: slow reaction to segment loss

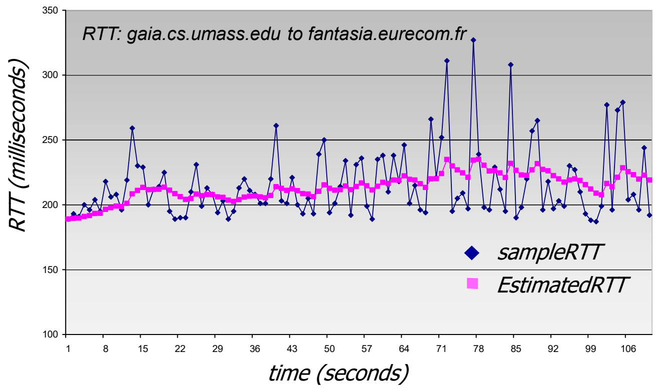
Q: how to estimate RTT?

- **SampleRTT**: measured time from segment transmission until ACK receipt
  – ignore retransmissions (Karn/Partridge algorithm)
- **SampleRTT** will vary, want estimated RTT "smoother"
  – average several recent measurements, not just current **SampleRTT**

TCP Round Trip Time, Timeout

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

- *exponential weighted moving average*
- *influence of past sample decreases exponentially fast*
- *typical value: α = 0.125*



Adaptive Retransmission *(The Original Algorithm)*

- Measure **`SampleRTT`** for each segment/ ACK pair
- Compute weighted average of RTT
  - **`EstRTT`** = α x **`EstRTT`** + $(1-\alpha)$ x **`SampleRTT`**
  - Typically, $0.8 \le \alpha \le 0.9$
- Set timeout based on **`EstRTT`**
  - **`TimeOut = 2`** x **`EstRTT`**

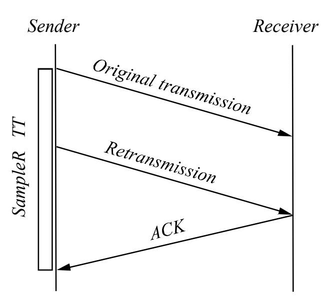
Copyright ©by Zygmunt J. Haas, 2017

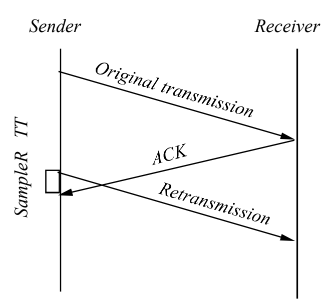
Jacobson/ Karels Algorithm

- Consider variance when setting timeout value
- New Calculations for average RTT
  - **EstRTT** = δ x **SampleRTT** + (1- δ) x **EstRTT**
  - **Diff** = **SampleRTT** - **EstRTT**
  - **Dev** = δ x |**Diff**| + (1- δ) x **Dev**
  - where δ is a factor between 0 and 1
- **TimeOut** = μ x **EstRTT** + ϕ x **Dev**
  - where μ = 1 and ϕ = 4
- Notes
  - accurate timeout mechanism important to congestion control (more about this later)

Copyright ©by Zygmunt J. Haas, 2017

Karn/Partridge Algorithm





- Do not sample RTT when retransmitting
- Double timeout after each retransmission (exponential Backoff)

Copyright ©by Zygmunt J. Haas, 2017

TCP reliable data transfer

- TCP provides reliable data transfer service on top of IP's unreliable service
- Cumulative ACKs
- TCP uses single retransmission timer
- Retransmissions are triggered by:
  - timeout events
  - duplicate ACKs
- Initially consider simplified TCP sender:
  - ignore duplicate ACKs
  - ignore flow control, congestion control

TCP sender events

When data is rcvd from application:

- Create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running (think of timer as for oldest unacked segment)
- expiration interval: TimeOutInterval

Upon timeout:

- retransmit segment that caused timeout
- restart timer

When ACK is rcvd:

- If acknowledges previously unACKed segments
  - update what is known to be ACKed
  - start timer if there are still outstanding segments

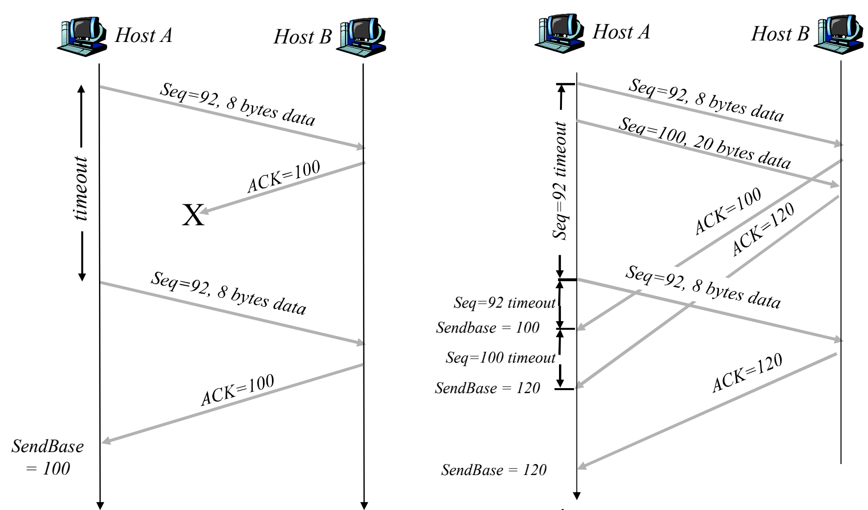
```
NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)

  event: data received from application above
      create TCP segment with sequence number NextSeqNum
      if (timer currently not running)
          start timer
      pass segment to IP
      NextSeqNum = NextSeqNum + length(data)

  event: timer timeout
      retransmit not-yet-acknowledged segment with
           smallest sequence number
      start timer

  event: ACK received, with ACK field value of y
      if (y > SendBase) {
          SendBase = y
          if (there are currently not-yet-acknowledged segments)
               start timer
          }

} /* end of loop forever */
```

TCP sender (simplified)

Comment:
- SendBase-1: last cumulatively ACKed byte

Example:
- SendBase-1 = 71; y= 73, so the rcvr wants 73+ ; y > SendBase, so that new data is ACKed

TCP transmission scenarios

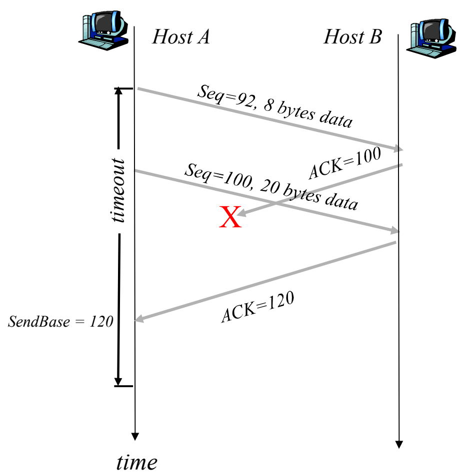


lost ACK scenario

premature timeout

Copyright ©by Zygmunt J. Haas, 2017

TCP transmission scenarios (more)



Cumulative ACK scenario

TCP ACK generation [RFC 1122, RFC 2581]

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	*Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK*
Arrival of in-order segment with expected seq #. One other segment has ACK pending	*Immediately send single cumulative ACK, ACKing in-order segments*
Arrival of out-of-order segment higher-than-expect seq. # . Gap detected	*Immediately send duplicate ACK, indicating seq. # of next expected byte*
Arrival of segment that partially or completely fills gap	*Immediate send ACK, provided that segment starts at lower end of gap*

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

Copyright ©by Zygmunt J. Haas, 2017

The Fast Retransmit Algorithm:

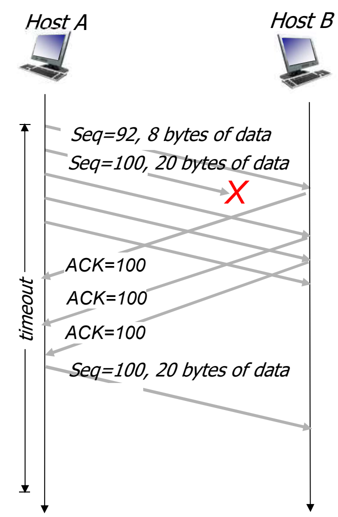
```
event: ACK received, with ACK field value of y
            if (y > SendBase) {
                 SendBase = y
                 if (there are currently not-yet-acknowledged segments)
                      start timer
               }
            else {
                   increment count of dup ACKs received for y
                   if (count of dup ACKs received for y = 3) {
                        resend segment with sequence number y
                   }
```

a duplicate ACK for already ACKed segment

fast retransmit

Copyright ©by Zygmunt J. Haas, 2017

TCP Fast Retransmit

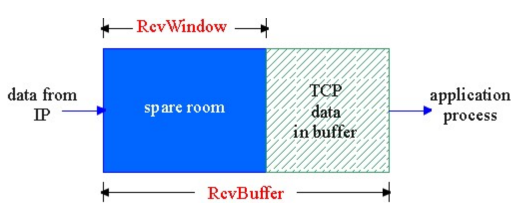


fast retransmit after sender receipt of triple duplicate ACK

TCP Flow Control

- receive side of TCP connection has a receive buffer:

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

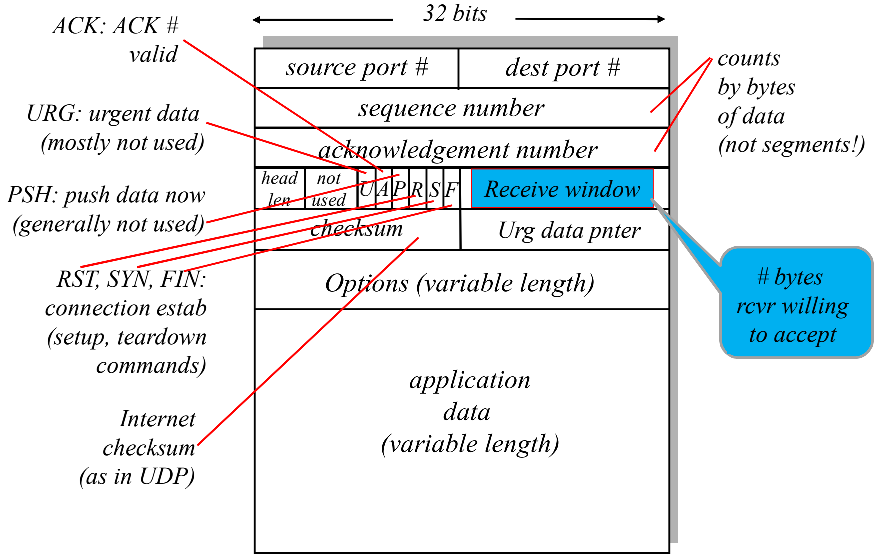


- speed-matching service: matching the send rate to the receiving app's drain rate

- app process may be slow at reading from buffer

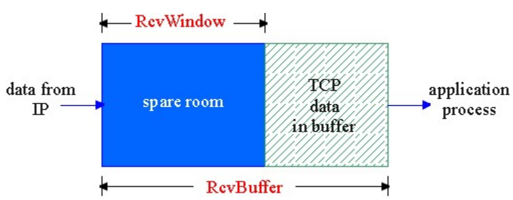
Copyright ©by Zygmunt J. Haas, 2017

TCP segment format



Copyright ©by Zygmunt J. Haas, 2017

TCP Flow control: how it works



(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer

= `RcvWindow`

= `RcvBuffer-[LastByteRcvd - LastByteRead]`

- Rcvr advertises spare room by including value of **`RcvWindow`** in segments
- Sender limits unACKed data to **`RcvWindow`**
  - guarantees receive buffer doesn't overflow

Copyright ©by Zygmunt J. Haas, 2017

TCP Flow control

- Host sends 1 byte in a segment
  (Inefficient usage of bandwidth)
  – 40 bytes TCP+IP headers
  – 1 byte application data

<u>The "Silly Window Syndrome"</u>

Receiver: opens window of 1 byte;

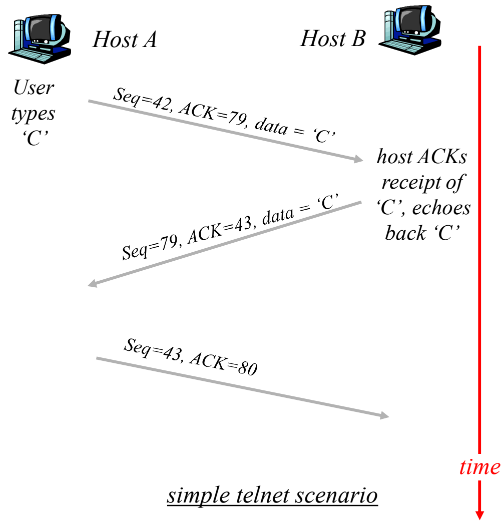
Sender: sends one-byte segment

Problem: Extremely inefficient

Solutions:

(1) after advertising window of size 0, receiver can only open MSS-size window

(2) Nagle's Algorithm



Nagle's Algorithm

<u>Nagle's Algorithm</u>

```
if available data and window ≥ MSS:
   send a full segment
else
   if there is unACKed data in transmit
     buffer the new data until ACK received
   else
     send all the new data
```

Note: May result in a single byte per RTT; to turn off Nagle's algorithm, use TCP_NODELAY option

TCP Connection Management

- <u>Connection Establishment</u>

  Initialize TCP variables:
  - seq. #s, buffers, flow control info (e.g. **RcvWindow**)

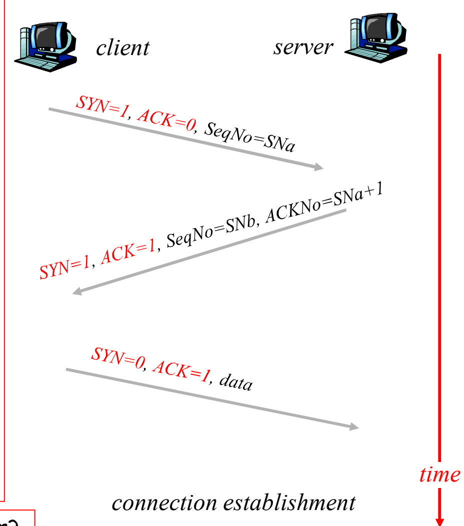
<u>Three way handshake:</u>

<u>Step 1:</u> client host sends TCP SYN segment to server
- specifies <u>initial seq #</u>
- no data

<u>Step 2:</u> server host receives SYN, replies with SYN+ACK segment
- server allocates buffers
- specifies server initial seq. #

<u>Step 3:</u> client receives SYN+ACK, replies with ACK segment, which may contain data

Q: How to choose the initial sequence number?

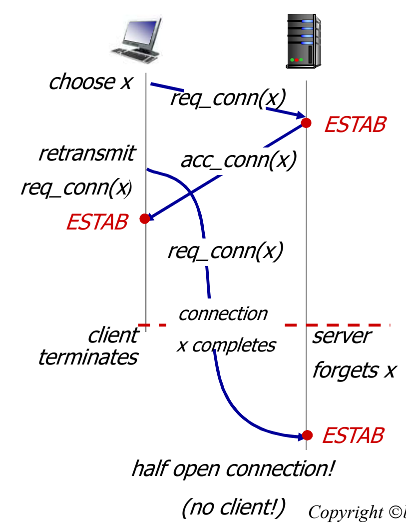


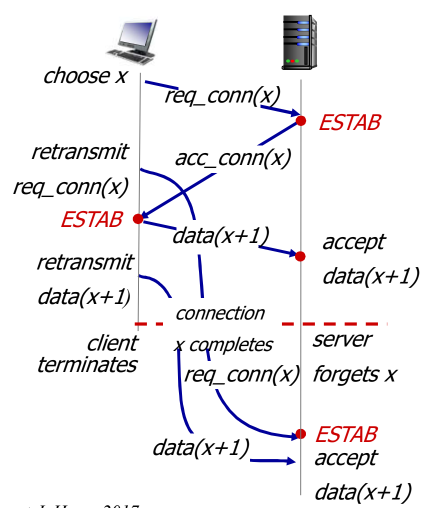
connection establishment

Copyright ©by Zygmunt J. Haas, 2017

Agreeing to Establish a Connection

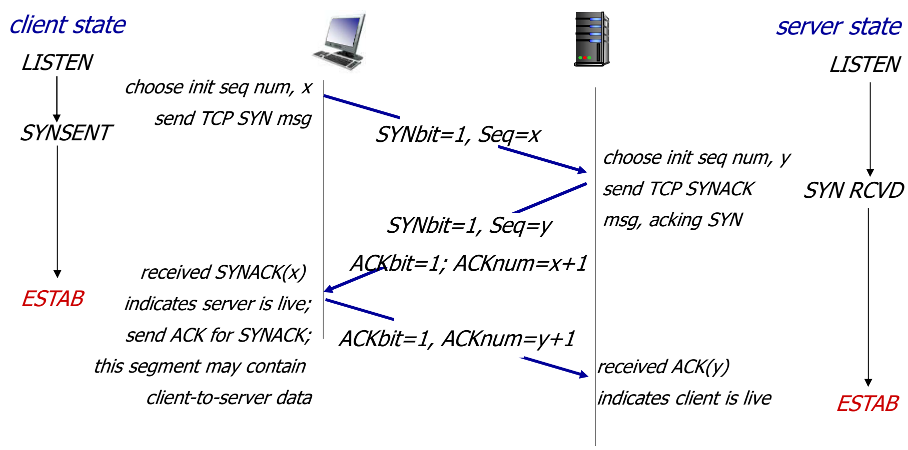
2-way handshake failure scenarios:





Copyright ©by Zygmunt J. Haas, 2017

TCP 3-way Handshake



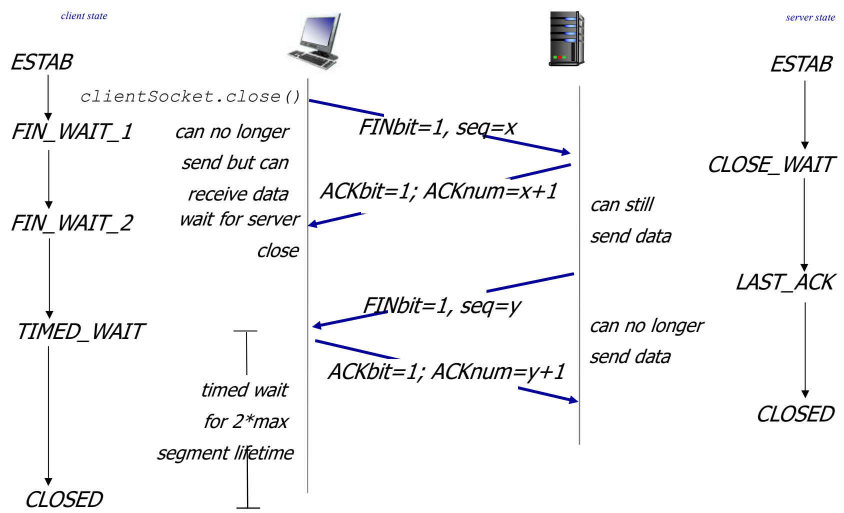
Copyright ©by Zygmunt J. Haas, 2017

TCP: Closing a Connection

- ❖ client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- ❖ respond to received FIN with ACK
  - on receiving FIN, ACK can be combined with own FIN
- ❖ simultaneous FIN exchanges can be handled

Copyright ©by Zygmunt J. Haas, 2017

TCP: Closing a Connection



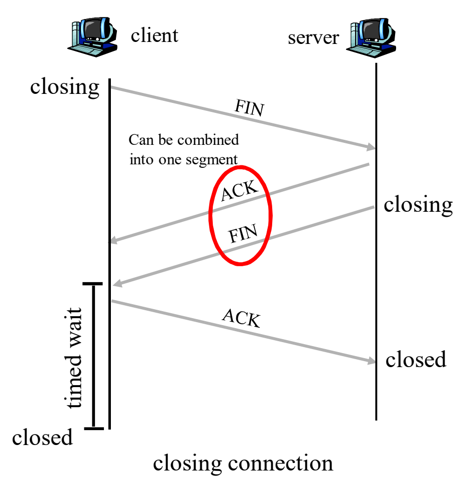
Copyright ©by Zygmunt J. Haas, 2017

TCP Connection Management (cont.)

Step 3: client receives FIN, replies with ACK.

- Enters "timed wait" - will respond with ACK to received FINs

Step 4: server, receives ACK. Connection closed.



Copyright ©by Zygmunt J. Haas, 2017

Protection Against 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

- Implication – Reduced throughput
  - MSL (Maximum Segment Lifetime) assumed to be 120 secs
  - No two TCP segments can have the same SeqNo within 120 secs

Copyright ©by Zygmunt J. Haas, 2017

Performance: Keeping the Pipe Full

- 16-bit **`AdvertisedWindow`**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB (= 1.5*10^6 / 8 * .1)
Ethernet (10 Mbps)	125KB
T3 (45 Mbps)	562.5KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.9MB
STS-12 (622 Mbps)	7.7MB
STS-24 (1.2 Gbps)	15MB

The amount of in-transit unACKed TCP data

- Delay = "round-trip time" (100 msec in the above example)
- 16 bit receiver window represents at most: 2^16 * 8 = 0.5 MB of data

Copyright ©by Zygmunt J. Haas, 2017