Washington University in St. Louis

Engineering

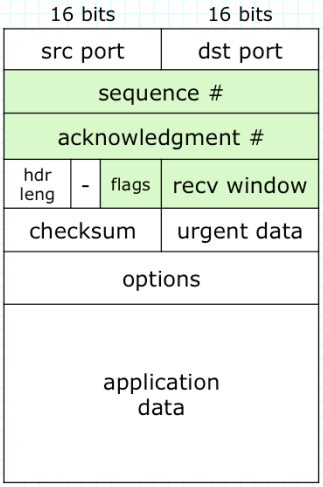
12. A Closer Look at TCP

- TCP connection management
- Reliable data transport in TCP (simplified)
- How TCP selects a timeout value

Jon Turner – slides adapted from Kurose and Ross

Reminder - TCP Segment Format

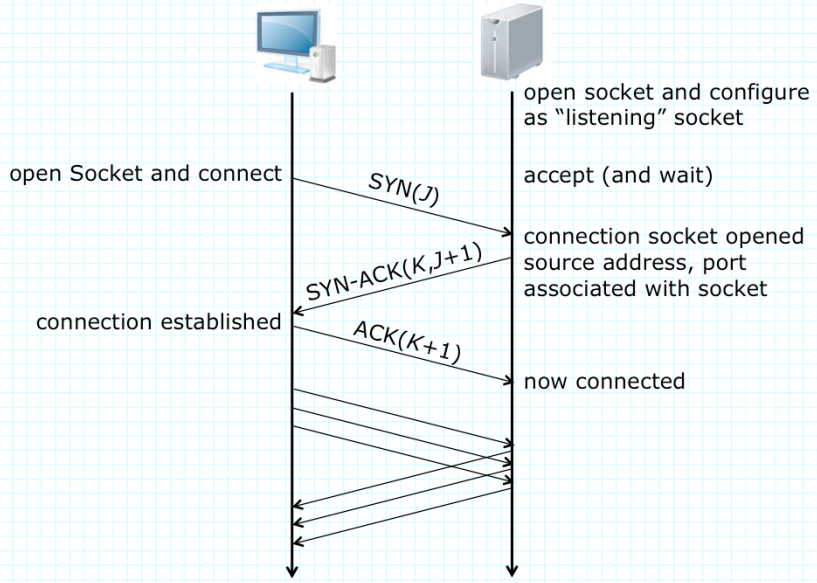
- Sequence number is sequence number of first *byte* in payload
- Ack number is sequence number of next *expected byte*
- Flags include
  - » SYN and FIN bits used to setup and teardown connections
  - » RST used to reset connection
  - » ACK flag indicates valid ack #
  - » URG flag – generally not used
  - » PSH flag – generally not used
- Receive window specifies amount of buffer space available at rcvr
  - » used for *flow control*



2

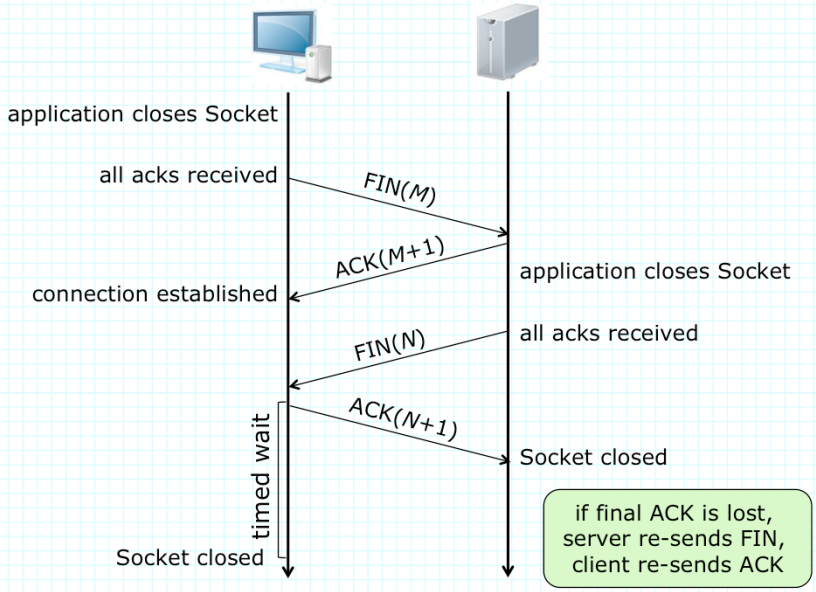
Washington University in St. Louis — Engineering

TCP Connection Establishment



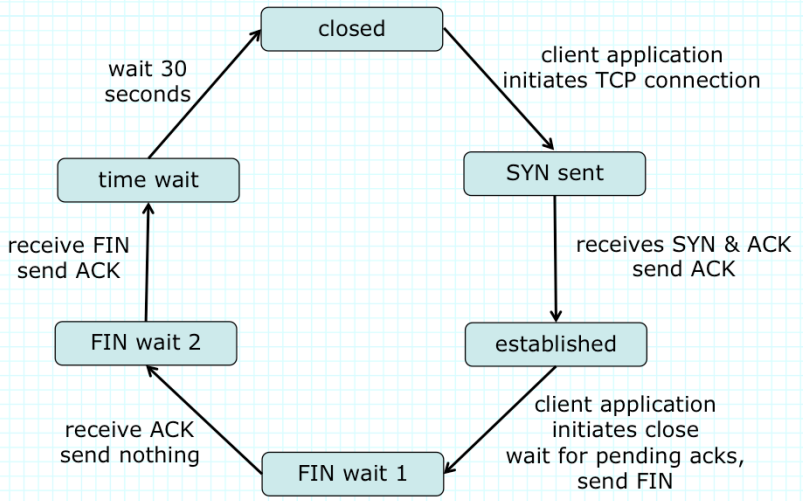
3

Typical Connection Close



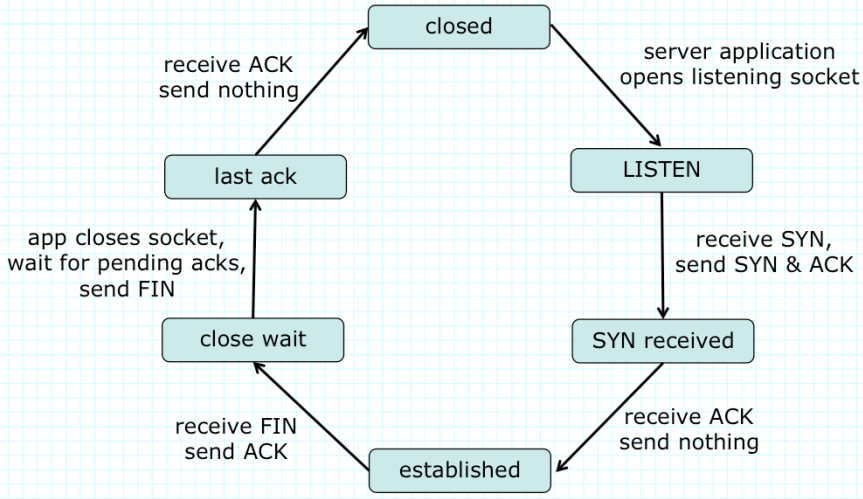
4

TCP Client "Life Cycle"



5

TCP Server "Life Cycle"



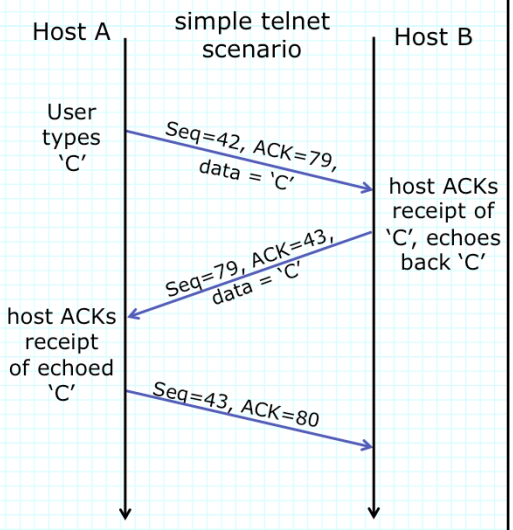
6

Exercises

1. Draw a space-time diagram showing the complete life-cycle of a TCP connection. Label the vertical line at the client with the client states. Label the vertical line at the server with the server states.

TCP Sequence Numbers 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
- Out-of-order segments
  - » usually, out-of-order segments are buffered
  - » TCP RFC doesn't actually specify



8

TCP Reliable Data Transfer

- TCP creates reliable service on top of IP
  - » pipelined segments (window size expressed in bytes)
  - » cumulative acks
- TCP uses single retransmission timer
  - » controls retransmission of "oldest" unacknowledged segment
  - » when lost segment received, cumulative acks allow sender to avoid retransmitting segments first sent after lost segment
- Retransmissions are triggered by
  - » timeout events
  - » duplicate acks
- Initially consider simplified TCP sender
  - » ignore duplicate acks
  - » ignore flow control, congestion control

9

TCP Sender Events:

data rcvd from app:

- add bytes to send buffer
  - » send segment(s), if "allowed"
  - » seq # is byte-stream number of first data byte in segment
- Start timer if not already running
  - » timer controls retransmission of oldest un-acked segment
- Expiration interval `TimeOutInterval`
  - » determined dynamically based on RTT

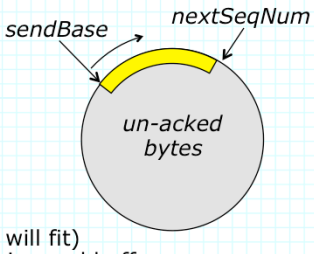
timeout:

- Retransmit segment that caused timeout
- Restart timer

Ack rcvd:

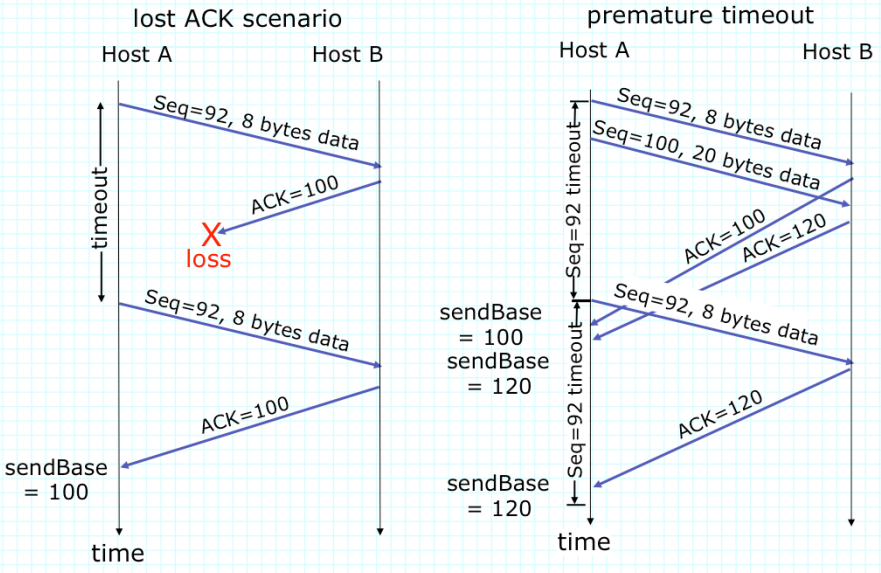
- If ack matches previously un-acked segment(s)
  - » discard bytes that have been acked
  - » re-start timer if there are still un-acked segments

Simplified TCP Sender

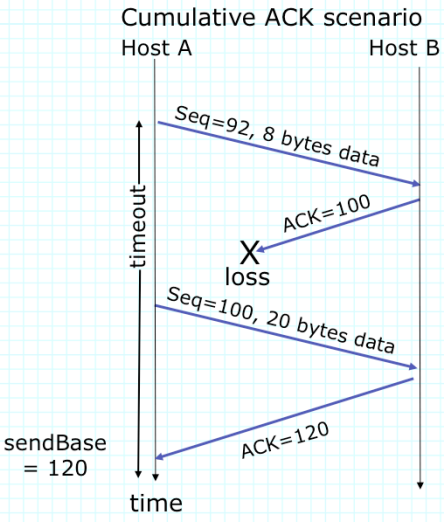



```
nextSeqNum = initialSeqNum
sendBase = initialSeqNum

loop (forever) {
  switch(event)

  event: data received from application above
      add received bytes to send buffer (as many, as will fit)
      if allowed, create/send segment(s) using bytes in send buffer
      if timer currently not running, start timer
      pass segment to IP
      nextSeqNum = nextSeqNum + length(segment)

  event: timeout
      retransmit not-yet-acknowledged segment with
            smallest sequence number
      re-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 Retransmission Scenarios



TCP Retransmission Scenarios



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 500 ms 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 both 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	Immediately send ACK, provided that segment starts at lower end of gap

14

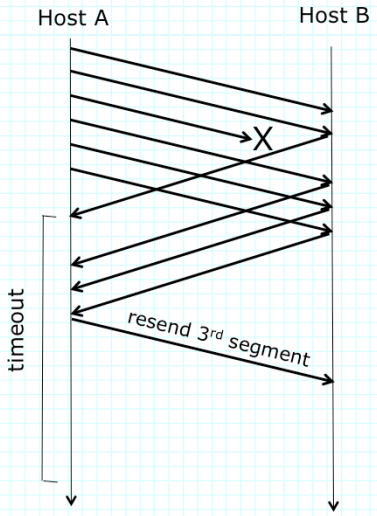
Exercises

1. Draw a space-time diagram showing two hosts communicating with TCP. Suppose *A* sends 8 packets to *B*, each containing 10 bytes of data, and that the fourth and fifth packets are lost. Show these packets and the acks sent by *B.* Show the sequence numbers in the ACKs. Suppose that when *A* retransmits the fourth packet, it is also lost, but the fifth packet gets through. Show these packets and resulting ACKs. Finally, show what happens when the fourth packet is retransmitted the second time and successfully delivered to *B*. At what point in this scenario does the application at *B* receive the eighth packet?

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 duplicate ACKs for the same data, it concludes that segment after ACKed data was lost:
  - *fast retransmit*: re-send segment before timer expires
- Why wait for 3 duplicate ACKs?
  - since IP may deliver segments out-of-order, a duplicate ACK does not necessarily mean a lost segment
  - but, with 3 duplicate ACKs, a lost segment is likely

Fast Retransmit Illustration



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)
                    re-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

TCP Round Trip Time and Timeout

- How should TCP select its timeout value?
  - must be longer than RTT, but not too much longer
    - if too long, retransmissions needlessly delayed
    - if too short, may retransmit when not necessary
  - but different connections have different RTTs and RTT varies during a connection as well
- How to estimate RTT?
  - `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
  - `SampleRTT` will fluctuate, want estimated RTT "smoother"
    - average several recent measurements, not just current `SampleRTT`

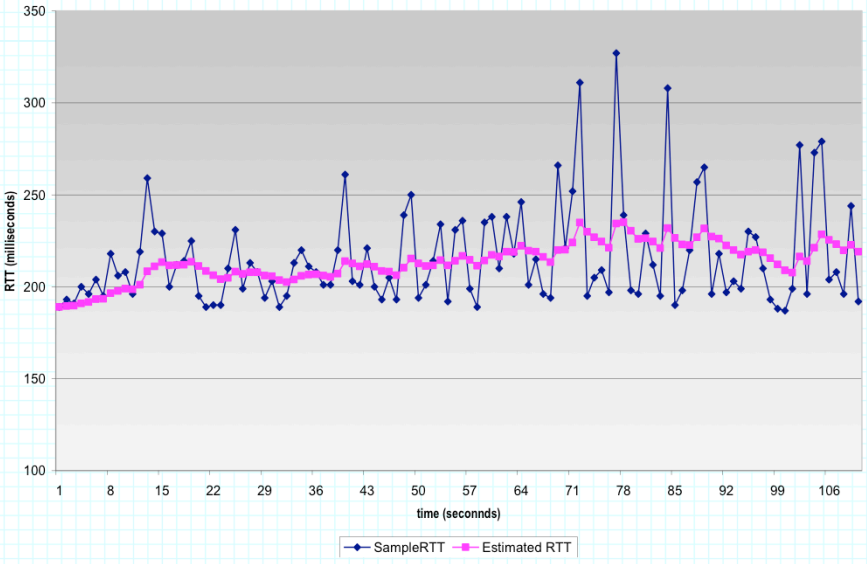
    `EstimatedRTT = (1-α)*EstimatedRTT + α*SampleRTT`

    where α is typically 0.125; this method called exponential weighted moving average

Washington University in St. Louis

Engineering

Example RTT estimation:



20

Computing the Timeout

- Timeout should be `EstimatedRTT` plus "safety margin"
  - » if large variation in `EstimatedRTT` need larger safety margin
- First estimate of how much SampleRTT deviates from EstimatedRTT

  `DevRTT = (1-β)*DevRTT + β*|SampleRTT-EstimatedRTT|`

  typically, β = 0.25
- Then set timeout interval

  `TimeoutInterval = EstimatedRTT + 4*DevRTT`

Exercises

1. Consider a TCP connection linking a residential client *A* to a remote server that is 2000 km away from *A*. Suppose the DSL link connecting *A* to its ISP has a download rate of 2 Mb/s and can store packets with a total length of 64 KB. Ignoring all other considerations, what is the minimum delay on the path from the server to *A*? If the queue is oscillating rapidly between being empty and full, what would you expect the estimated RTT to be? What about the *DevRTT* value? Compute the resulting timeout interval. Is this value "safe"? Is it too conservative?