Chapter 3 outline

- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - flow control
  - connection management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control
TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

point-to-point:
one sender, one receiver
reliable, in-order byte
steam:
no “message boundaries”
pipelined:
TCP congestion and flow
control set window size

send & receive buffers

• full duplex data:
  – bi-directional data flow
    in same connection
  – MSS: maximum
    segment size

• connection-oriented:
  – handshaking
    (exchange of control
    msgs) init’s sender,
    receiver state before
    data exchange

• flow controlled:
  – sender will not
    overwhelm receiver
TCP segment structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source port #</td>
<td>Source port number (16 bits)</td>
</tr>
<tr>
<td>dest port #</td>
<td>Destination port number (16 bits)</td>
</tr>
<tr>
<td>sequence number</td>
<td>Sequence number (32 bits)</td>
</tr>
<tr>
<td>acknowledgement number</td>
<td>Acknowledgement number (32 bits)</td>
</tr>
<tr>
<td>head len</td>
<td>Header length (16 bits)</td>
</tr>
<tr>
<td>UAP</td>
<td>Urgent, Ack, Push flags (3 bits)</td>
</tr>
<tr>
<td>Receiver window</td>
<td>Receive window (32 bits)</td>
</tr>
<tr>
<td>checksum</td>
<td>Checksum (32 bits)</td>
</tr>
<tr>
<td>Urg data ptr</td>
<td>Urgent data pointer (16 bits)</td>
</tr>
<tr>
<td>Options (variable length)</td>
<td>Options (variable length)</td>
</tr>
<tr>
<td>application data</td>
<td>Application data (variable length)</td>
</tr>
</tbody>
</table>

URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

RST, SYN, FIN: connection estab (setup, teardown commands)

Internet checksum (as in UDP)

Counting by bytes of data (not segments!)

# bytes rcvr willing to accept
Seq. #’s:
- byte stream “number” of first byte in segment’s data

ACKs:
- 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 implementation

TCP seq. #’s and ACKs

User types ‘C’

Host A
Seq=42, ACK=79, data = ‘C’
host ACKs receipt of ‘C’, echoes back ‘C’

Host B
Seq=79, ACK=43, data = ‘C’

Seq=43, ACK=80

host ACKs receipt of echoed ‘C’

simple telnet scenario
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

SampleRTT will vary, want estimated RTT “smoother”
  average several recent measurements, not just current SampleRTT
TCP Round Trip Time and Timeout

\[\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT}\]

- Exponential weighted moving average
- Influence of past sample decreases exponentially fast
- Typical value: \( \alpha = 0.125 \)
Example RTT estimation:

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr

![Graph showing RTT estimates over time with two lines: SampleRTT and Estimated RTT.](image-url)
Setting the timeout

• **EstimatedRTT** plus “safety margin”
  – large variation in EstimatedRTT -> larger safety margin

• first estimate of how much SampleRTT deviates from EstimatedRTT:

\[
\text{DevRTT} = (1 - \beta) \text{DevRTT} + \beta |\text{SampleRTT} - \text{EstimatedRTT}|
\]

(typically, \( \beta = 0.25 \))

Then set timeout interval:

\[
\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}
\]
TCP creates rdt service on top of IP’s unreliable service

- Pipelined segments
- 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:**

- **data rcvd from app:**
  - 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)

- **timeout:**
  - retransmit segment that caused timeout
  - restart timer

- **Ack rcvd:**
  - If acknowledges previously unacked segments
  - update what is known to be acked
  - start timer if there are outstanding segments
TCP
sender
(simplified)

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 */

Comment:
- SendBase-1: last cumulatively ack’ed byte

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

Host A

SendBase = 100

Seq=92, 8 bytes data

ACK=100

X loss

time

lost ACK scenario

Host B

Seq=92, 8 bytes data

ACK=100

timeout

Host B

Seq=92, 8 bytes data

ACK=100

Seq=100, 20 bytes data

ACK=100

ACK=120

Seq=92, 8 bytes data

ACK=120

Seq=92, 8 bytes data

ACK=120

timeout

premature timeout

Host A

Seq=92, 8 bytes data

ACK=100

SendBase = 100

SendBase = 120

SendBase = 120

time
TCP retransmission scenarios (more)

Host A
Seq=92, 8 bytes data
Seq=100, 20 bytes data
SendBase = 120

Host B
ACK=100
ACK=120

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