CSCE 463/612
Networks and Distributed Processing
Spring 2024

Transport Layer VI

Dmitri Loguinov
Texas A&M University

March 18, 2024

Chapter 3: Roadmap

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 Reliable Data Transfer

- TCP creates rdt service on top of IP's unreliable service
  - Hybrid of Go-back-N and Selective Repeat
- Pipelined segments
- Cumulative acks
- TCP uses single retransmission timer
  - For the oldest unACK'ed packet
  - Retx only the base
- Retransmissions are triggered by:
  - Timeout events
  - Duplicate acks
- Initially consider simplified TCP sender:
  - Ignore duplicate acks
  - Ignore flow control, congestion control

TCP Sender (Simplified)

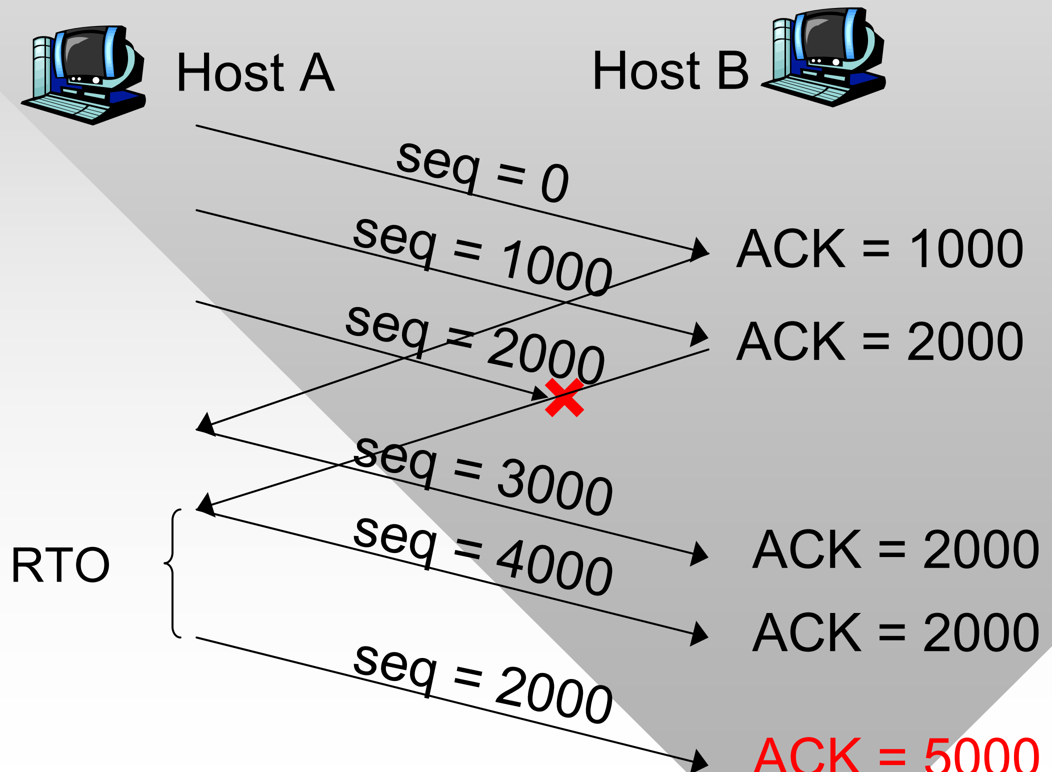
```
NextSeqNum = InitialSeqNum // random for each transfer
SendBase = InitialSeqNum
 loop (forever) {
   switch(event) {
   (a) data received from application above (assuming it fits into window):
          create TCP segment with sequence number NextSeqNum
          pass segment to IP
          NextSeqNum = NextSeqNum + length(data)
          if (timer currently not running)
                 start timer
   (b) timeout:
          retransmit pending segment with smallest sequence
          number (i.e., SendBase); restart timer
   (c) ACK received, with ACK field value of y
      if (y > SendBase) {
        SendBase = y
        if (there are currently not-yet-acknowledged segments)
            restart timer with latest RTO
        else cancel timer }
  }
} /* end of loop forever */
```

TCP Seq. #'S and ACKs

FTP Example:

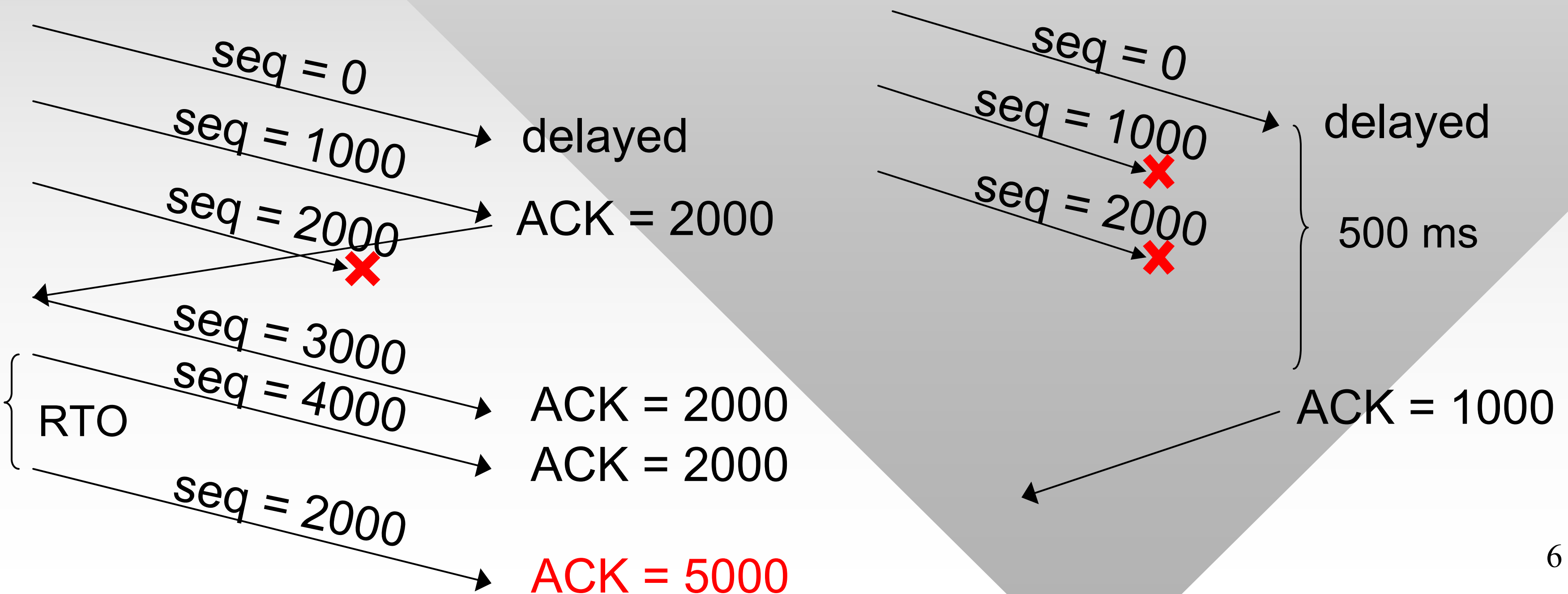
- Suppose MSS = 1,000 bytes and the sender has a large file to transmit (we ignore seq field in ACKs and ACK field in data pkts)

What is the sender window size?



TCP ACK Generation [RFC 1122, RFC 2581]

- Receiver immediately ACKs the base of its window in all cases except Nagle's algorithm:
  - For *in-order* arrival of packets, send ACKs for every *pair* of segments; if second segment of a pair not received in 500ms, ACK the first one alone



Fast Retransmit

- Time-out period often relatively long
  - Especially in the beginning of transfer (3 seconds in RFC 1122)
- Idea: infer loss via duplicate ACKs
  - Sender often sends many segments back-to-back
  - If a segment is lost, there will be many duplicate ACKs
- If sender receives 3 duplicate ACKs for its base, it assumes this packet was lost
  - Fast Retransmit: resend the base segment immediately (i.e., without waiting for RTO)
- Note that reordering may trigger unnecessary retx
  - To combat this problem, modern routers avoid load-balancing packets of same flow along multiple paths

Fast Retransmit Algorithm:

```
(c) event: ACK received, with ACK field value of y
        if (y > SendBase) {
            SendBase = y; dupACK = 0;
            if (SendBase != NextSeqNum)
                restart timer with latest RTO;
            else
                cancel timer;  // last pkt in window
        }
        else if (y == SendBase) {
            dupACK++;
            if (dupACK == 3)
                 { resend segment with sequence y; restart timer}
        }
```

a duplicate ACK for already ACKed segment

fast retransmit

Chapter 3: Roadmap

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

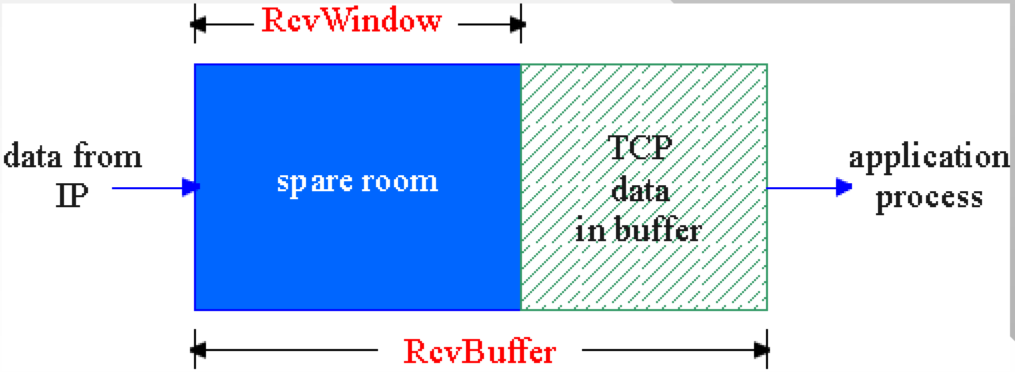
TCP Flow Control

- Assume packets received without loss, but the application does not call recv()
  - How to prevent sender from overflowing TCP buffer?

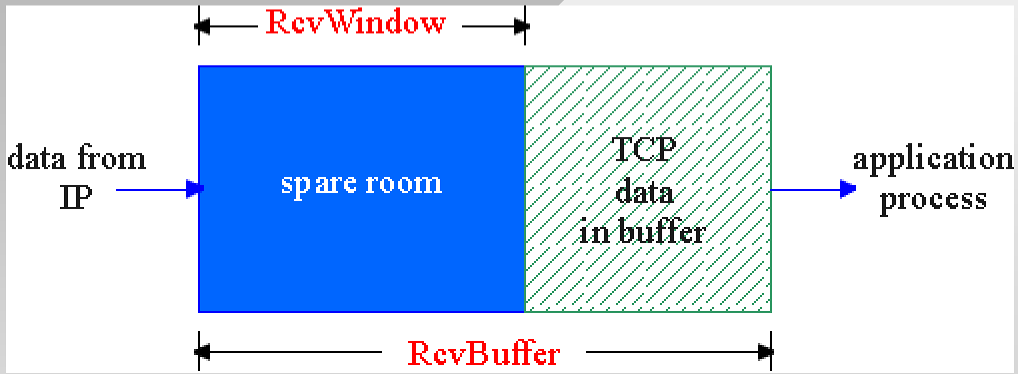
Flow control

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

- Speed-matching service: sender rate to suit the receiving app's ability to process incoming data



TCP Flow Control: How It Works



- Receiver advertises spare room by including value of `RcvWin` in segments
- Sender enforces `seq < ACK + RcvWin`
  - Guarantees receiver buffer doesn't overflow

- Spare room in buffer

```
RcvWin = RcvBuffer –
[LastByteReceivedInOrder - LastByteDelivered]
```

last ACK-1

went to application

- Combining both constraints (sender, receiver):

```
seq < min(sndBase+sndWin, ACK+RcvWin)
```

Chapter 3: Roadmap

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 Connection Management

- Purpose of connection establishment:
  - Exchange initial seq #s
  - Exchange flow control info (i.e., `RcvWin`)
  - Negotiate options (SACK, large windows, etc.)

Three way handshake:

- Step 1: client sends TCP SYN to server
  - Specifies initial seq # X and buffer size `RcvWin`
  - No data, ACK bit = 0
- Step 2: server gets SYN, replies with SYN+ACK
  - Sends server initial seq # Y and buffer size `RcvWin`
  - No data, ACK val = X+1
- Step 3: client receives SYN+ACK, replies with ACK segment
  - Seq = X+1, ACK val = Y+1
  - May contain regular data, but many servers will break
- Step 4: regular packets
  - Seq = X+1, ACK = Y+1

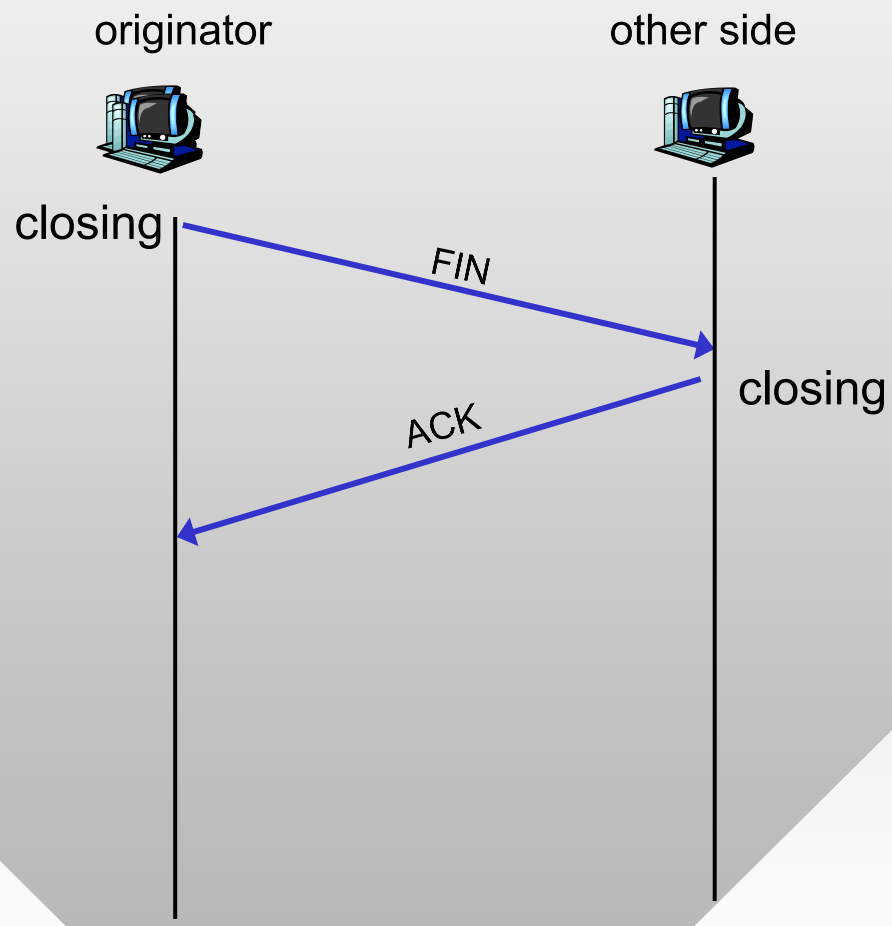
TCP Connection Management (Cont.)

Closing a connection:

- Closing a socket:
  `closesocket(sock);`

Step 1: originator end system sends TCP FIN control segment to server

Step 2: other side receives FIN, replies with ACK. Connection in "closing" state, sends FIN



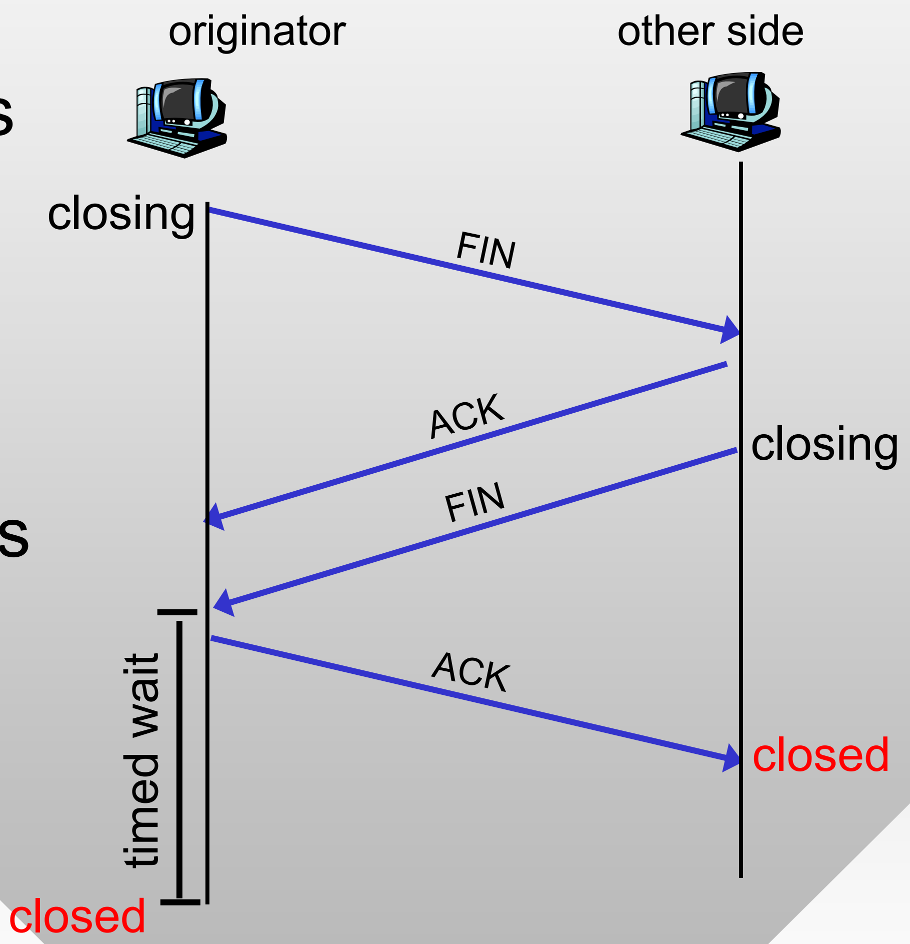
TCP initiates a close when it has all ACKs for the transmitted data

TCP Connection Management (Cont.)

Step 3: originator receives FIN, replies with ACK
- Enters "timed wait" - will respond with ACK to received FINs

Step 4: other side receives ACK; its connection considered closed

Step 5: after a timeout (TIME_WAIT state lasts 240 seconds), originator's connection is closed as well



birectional transfer means both sides must agree to close