# Internet Programming & Protocols Lecture 8

TCP finite state machine

TCP flow-control and bandwidth-delay

TCP on long fat pipes

## TCP: Overview

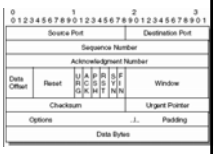
- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver


© Kurose



## TCP

- TCP provides reliable stream of bytes
- Header includes
  - Checksum
  - Sequence/ACK numbers
  - Flow control window
  - Port numbers
- Protocol provides
  - Connection establishment (SYN-ACK) and close (FIN)
  - ACK for in-order bytes received
  - Timers for packet retransmission
  - Sliding window flow control with send and receive buffers
  - Receiver buffers out of order packets
    - Duplicate ACKs
    - Then when missing packet arrives, cumulative ACK
  - Protocol is "stateful"






## TCP connection refused

- If TCP port on server is not "active", TCP replies with reset (RST)

on deneb: telnet achernar 9999

```
DENEB.1056 > ACHERNAR.9999: S 1039104000:1039104000(0)
win 8192 <mss 1460>
4500 002c c5fd 0000 3c06 faf2 80a9 5e4a
80a9 5e3f 0420 270f 3def 7800 0000 0000
6002 2000 d92b 0000 0204 05b4

ACHERNAR.9999 > DENEB.1056: R 0:0(0) ack 1039104001 win 0
4500 0028 f82f 0000 3c06 c8c4 80a9 5e3f
80a9 5e4a 270f 0420 0000 0000 3def 7801
5014 0000 10d5 0000
```

**SYN flooding**

Hacker denial of service attack sends oodles of SYN packets to active server ports. Resources are consumed awaiting completion of 3-way handshake.



## TCP connection time out

- If remote host does not respond or behind a firewall, TCP resends the SYN packet several times, backing off exponentially. connect() eventually fails with ETIMEDOUT. OS-dependent, but takes about 3 minutes and 6 tries with intervals { 3s, 6s, 12s, 24s, 48s}
- Exponential backoff wasn't in RFC 793 added in '88
- Notice today's TCP options in SYN packet in tcpdump below

```
1125876766.655553 manitou.33877 > wisp.telnet: S 167809258:167809258(0) win 5840
    <mss 1460,sackOK,timestamp 45625354 0,nop,wscale 0> (DF) [tos 0x10]
1125876769.646955 manitou.33877 > wisp.telnet: S 167809258:167809258(0) win 5840
    <mss 1460,sackOK,timestamp 45625654 0,nop,wscale 0> (DF) [tos 0x10]
1125876775.646951 manitou.33877 > wisp.telnet: S 167809258:167809258(0) win 5840
    <mss 1460,sackOK,timestamp 45626254 0,nop,wscale 0> (DF) [tos 0x10]
1125876787.646953 manitou.33877 > wisp.telnet: S 167809258:167809258(0) win 5840
    <mss 1460,sackOK,timestamp 45627454 0,nop,wscale 0> (DF) [tos 0x10]
1125876811.646953 manitou.33877 > wisp.telnet: S 167809258:167809258(0) win 5840
    <mss 1460,sackOK,timestamp 45629854 0,nop,wscale 0> (DF) [tos 0x10]
1125876859.646965 manitou.33877 > wisp.telnet: S 167809258:167809258(0) win 5840
    <mss 1460,sackOK,timestamp 45634654 0,nop,wscale 0> (DF) [tos 0x10]
```



## TCP sending one byte

- How many packets does TCP take to send one date byte?
- How many extra (header) bytes? How long? (# of RTTs)
- Contrast with sending one byte with UDP …

```
manitou.33878 > whisper.5001: S 885110161:885110161(0) win 5840 <mss
    1460,sackOK,timestamp 45695408 0,nop,wscale 0> (DF)
  4500 003c f421 4000 4006 a9ec c0a8 0104
  a024 3add 8456 1389 34c1 b591 0000 0000
  a002 16d0 cded 0000 0204 05b4 0402 080a
  02b9 41b0 0000 0000 0103 0300
whisper.5001 > manitou.33878: S 2714686246:2714686246(0) ack 885110162 win 5792
    <mss 1436,sackOK,timestamp 160286560 45695408,nop,wscale 5> (DF)
   4500 003c 0000 4000 3406 aa0e a024 3add
   c0a8 0104 1389 8456 a1ce d326 34c1 b592
   a012 16a0 883c 0000 0204 059c 0402 080a
   098d c760 02b9 41b0 0103 0305
manitou.33878 > whisper.5001: . ack 1 win 5840 <nop,nop,timestamp 45695411
    160286560> (DF)
  4500 0034 f422 4000 4006 a9f3 c0a8 0104
  a024 3add 8456 1389 34c1 b592 a1ce d327
  8010 16d0 b6bb 0000 0101 080a 02b9 41b3
```

```
manitou.33878 > whisper.5001: P 1:2(1) ack 1 win 5840 <nop,nop,timestamp 45695411
160286560> (DF)
    4500 0035 f423 4000 4006 a9f1 c0a8 0104
    a024 3add 8456 1389 34c1 b592 a1ce d327
    8018 16d0 9cd5 0000 0101 080a 02b9 41b3
    098d c760 20
manitou.33878 > whisper.5001: F 2:2(0) ack 1 win 5840 <nop,nop,timestamp 45695411
160286560> (DF)
    4500 0034 f424 4000 4006 a9f1 c0a8 0104
    a024 3add 8456 1389 34c1 b593 a1ce d327
    8011 16d0 b6b9 0000 0101 080a 02b9 41b3
    098d c760
whisper.5001 > manitou.33878: . ack 2 win 181 <nop,nop,timestamp 160286562 45695411>
(DF)
    4500 0034 1445 4000 3406 95d1 a024 3add
    c0a8 0104 1389 8456 a1ce d327 34c1 b593
    8010 00b5 ccd3 0000 0101 080a 098d c762
    02b9 41b3
whisper.5001 > manitou.33878: F 1:1(0) ack 3 win 181 <nop,nop,timestamp 160286562
45695411> (DF)
    4500 0034 1446 4000 3406 95d0 a024 3add
    c0a8 0104 1389 8456 a1ce d327 34c1 b594
    8011 00b5 ccd1 0000 0101 080a 098d c762
    02b9 41b3
manitou.33878 > whisper.5001: . ack 2 win 5840 <nop,nop,timestamp 45695413
160286562> (DF)
    4500 0034 f425 4000 4006 a9f0 c0a8 0104
    a024 3add 8456 1389 34c1 b594 a1ce d328
    8010 16d0 b6b4 0000 0101 080a 02b9 41b5
    098d c762
8 packets received by filter
```



## TCP available window

- If receiver application stops reading network data, its buffers fill and its available window goes to 0. Sender probes with exponential backoff to 60 seconds, then persists. ANIMATION

```
15:25:08.069441 thdsun.1574 > victory.7654: . 48489:49885(1396) ack 1 win 1638
15:25:08.069441 victory.7654 > thdsun.1574: . ack 49885 win 1396 (DF)
15:25:08.069441 thdsun.1574 > victory.7654: . 49885:51281(1396) ack 1 win 1638
15:25:08.089442 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:25:08.089442 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:25:12.929625 thdsun.1574 > victory.7654: . 51281:51282(1) ack 1 win 16384
15:25:12.929625 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:25:17.929814 thdsun.1574 > victory.7654: . 51281:51282(1) ack 1 win 16384
15:25:17.929814 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:25:22.930002 thdsun.1574 > victory.7654: . 51281:51282(1) ack 1 win 16384
15:25:22.930002 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:25:30.930304 thdsun.1574 > victory.7654: . 51281:51282(1) ack 1 win 16384
15:25:30.930304 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:25:46.930908 thdsun.1574 > victory.7654: . 51281:51282(1) ack 1 win 16384
15:25:46.930908 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:26:18.932116 thdsun.1574 > victory.7654: . 51281:51282(1) ack 1 win 16384
15:26:18.932116 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:27:18.934380 thdsun.1574 > victory.7654: . 51281:51282(1) ack 1 win 16384
15:27:18.934380 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
15:28:18.946637 thdsun.1574 > victory.7654: . 51281:51282(1) ack 1 win 16384
15:28:18.946637 victory.7654 > thdsun.1574: . ack 51281 win 0 (DF)
```



## Lost connection

- Connection breaks while sender is sending data. Retransmit with exponential backoff, eventually write() fails (connection closed (RST))

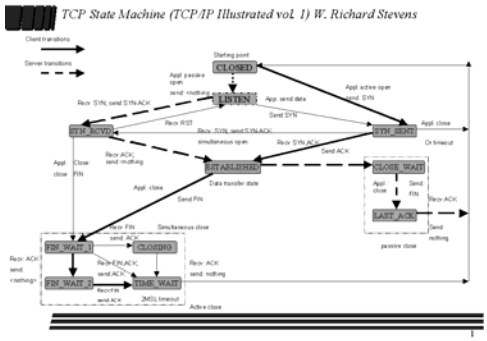
```
14:04:09.729116 thdsun.1566 > victory.7654: P 1:6(5) ack 1 win 16384
14:04:09.729116 victory.7654 > thdsun.1566: P 1:6(5) ack 6 win 32736 (DF)
14:04:09.779118 thdsun.1566 > victory.7654: . ack 6 win 16379
                Tom pulls out Ethernet cable
14:04:26.079726 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:04:26.679749 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:04:28.679824 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:04:32.679973 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:04:40.680271 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:04:56.680869 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:05:28.682063 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:06:32.684451 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:07:36.686838 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:08:40.689225 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379
14:09:44.691611 thdsun.1566 > victory.7654: P 6:11(5) ack 6 win 16379

exponential backoff, max 11 tries  (not in RFC 793)
```

Note: if net comes back up, your app may still take many more seconds before it resumes



## TCP finite state machine






## TCP states

- **SYN-SENT** represents waiting for a matching connection request after having sent a connection request.
- **SYN-RECEIVED** represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.
- **ESTABLISHED** represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.
- **FIN-WAIT-1** represents waiting for a connection termination request from the remote TCP, or an acknowledgment of the connection termination request previously sent.
- **FIN-WAIT-2** represents waiting for a connection termination request from the remote TCP.
- **CLOSE-WAIT** represents waiting for a connection termination request from the local user.
- **CLOSING** represents waiting for a connection termination request acknowledgment from the remote TCP.
- **LAST-ACK** represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP (which includes an acknowledgment of its connection termination request).
- **TIME-WAIT** represents waiting for enough time to pass to be sure the remote TCP received the acknowledgment of its connection termination request. (2MSL wait state)
- **CLOSED** represents no connection state at all.

A TCP connection progresses from one state to another in response to events.



## State transitions

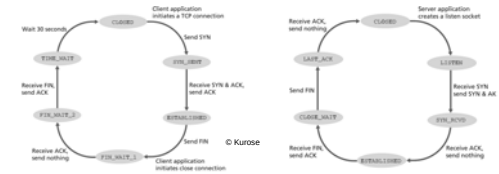



Figure 3.40 + A typical sequence of TCP states visited by a client TCP  Figure 3.41 + A typical sequence of TCP states visited by a server-side TCP

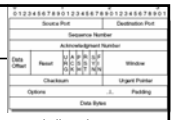
```
netstat –a
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 *:32768                 *:*                     LISTEN
tcp        0      0 *:7777                  *:*                     LISTEN
tcp        0      0 whisper.cs.utk.edu:ssh  catv:2486               ESTABLISHED
tcp        0      0 whisper.cs.utk.ed:42917 thoth.cs.utk.edu:ldap   TIME_WAIT
```

## Funky state transitions

- Simultaneous open
  - Unlikely, but possible for both ends to initiate SYN for the same port pair at the same instant
  - TCP handles this, one flow is created. (OSI creates two)
- Simultaneous close
  - Both sides send FIN at the same instant
  - TCP handles this OK
  - Both sides go: FIN_WAIT_1 → CLOSING → TIME_WAIT
- Timeout transitions
  - Exponential backoff retries for
    - No reply to SYN
    - No reply to SYN-ACK (SYN flooding)
    - No reply to data transfer write()
    - No response to FIN
  - Quiet time (2MSL wait)



## TCP reset (RST)



- RST packet (TCP header bit)
  - sent when a connection request (SYN) arrives and no process is listening on that port
  - Socket option (SO_LINGER) allows abortive close -- sends RST instead of normal FIN processing
  - If retransmits of packet fail, sender returns error to write() and sends RST
  - If delayed packets (or bogons) arrive for a closed or non-existent connection, the host sends back a RST
- Hacker note:
  - You can't just send random RST packets and close other's TCP connections. The kernel checks that sequence/ack numbers are "proper" and that the port numbers jive.
  - A hacker that is able to observe (sniff) an active TCP connection can create counterfeit RST packets and terminate a flow



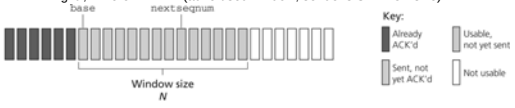
## Kernel data associated with a TCP connection

- Kernel data structure associated with socket descriptor for active TCP connection (/usr/src/linux/include/net/sock.h)
  - Send and receive buffers and control info
  - Sliding window control info (left edge, right edge)
  - Send and ACK sequence numbers (last sent, last ACKd)
  - State info (SYN-sent, ESTABLISHED, etc.)
  - Option info (nagle, so_reuse, etc)
  - Timer info, retry counters etc.
  - RTT variables
  - Urgent pointer stuff
  - Congestion control info (cwnd, ssthresh) … later



## TCP flow control

- TCP has two algorithms for throttling the sender
  - Flow control – end-to-end sliding window protocol to keep sender from overruning the receiver
  - Congestion control – algorithm(s) for socially acceptable behavior on the net
- TCP flow control
  - Sliding window size (bytes) provided in TCP header (16 bits)
  - Initial window size from RCVBUF size of socket
  - Window size can shrink and grow as receiver consumes incoming bytes
  - TCP assures that sender never has more than N unacknowledged bytes "in flight", where N = min(advertised window, sender's SNDBUF size)








## SNDBUF RCVBUF

- TCP sender's SNDBUF holds unsent and/or unacknowledged bytes
  - As bytes are acknowledged, the left edge slides to the right
  - If a packet is lost, the timeout, causes a retransmission from data in the SNDBUF
  - If the receiver's advertised window is 0 and the SNDBUF is full, write()'s block
- TCP receiver's RCVBUF holds bytes unread by the application or out-of-order bytes following "missing" bytes from lost packet(s)
  - RFC 793 says nothing about receiver having to retain out of order data, but most implementation do today
  - Receiver only ACK's last byte of "contiguous" data received
  - If receiver application is not reading data, advertised window can go to 0






## Next time …

- Performance tools