Today

- TCP SYN and FIN
- TCP flows and congestion

TCP header

IP proto = 6

```
0                   1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Hdr  |           |U|A|P|R|S|F|                               |
| length| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent offset         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```


connection establishment

3-way handshake

- client connect() sends
  - SYN
  - initial sequence number (ISN) (last +128000)
  - MSS option (max segment size) client will accept (MTU-40), if none (536)
- server replies with SYN and its ISN, plus ACK of client's seq. number + 1, and its MSS
- client responds with ACK of servers ISN + 1

if no server listening on TCP port, server host responds with RST, and client gets "connection refused"

connection establish

on deneb: ttcp -t -s -p9 -n1 -l1 achernar

```
DENEB.1055 > ACHERNAR.discard: S 732608000:732608000(0)
     win 8192 <mss 1460>
          4500 002c c4f2 0000 3c06 fbfd 80a9 5e4a

          80a9 5e3f 041f 0009 2baa b600 0000 0000

          6002 2000 d477 0000 0204 05b4 0001

ACHERNAR.discard > DENEB.1055: S 728320000:728320000(0)
     ack 732608001 win 8192 <mss 1460>
          4500 002c f742 0000 3c06 c9ad 80a9 5e3f

          80a9 5e4a 0009 041f 2b69 4800 2baa b601

          6012 2000 60fd 0000 0204 05b4 2000

DENEB.1055 > ACHERNAR.discard: . ack 1 win 8192
          4500 0028 c4f3 0000 3c06 fc00 80a9 5e4a
          80a9 5e3f 041f 0009 2baa b601 2b69 4801
          5010 2000 78ba 0000 0204 05b4 2000
```

connection refused

```
on deneb: telnet ahcernar 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 4341

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 0204 05b4 4341
```


connection time out

assignment 3, to dead host

```
20:39:50.821029 DENEB.1058 > dead.echo: S 1143424000:1143424000(0) win 8192
20:39:56.713164 DENEB.1058 > dead.echo: S 1143424000:1143424000(0) win 8192
20:40:08.710552 DENEB.1058 > dead.echo: S 1143424000:1143424000(0) win 8192
20:40:32.705299 DENEB.1058 > dead.echo: S 1143424000:1143424000(0) win 8192
```

- 75 seconds, for SunOS (3:17 for solaris)
- exponential backoff retry

connection close

- either client or server can initiate close()
- takes 4 packets to close (non deterministic)
- X initiates close (active close) sends FIN
- Y ACKs X's FIN
- Y then closes (passive close) and sends FIN
- X ACK's Y's FIN and goes into TIME_WAIT
- X's port will be busy til TIME_WAIT expires (30s to 2 minutes, 2MSL), may need SO_REUSEADDR socket option
- killing a process (ctrl-C) or system shutdown will cause FIN to be sent

FIN says process won't be sending any more data

varitions on close()

- close() returns immediately no more sends/recvs; send buffer sent, then FIN sent
- SO_LINGER socket option effects behavior of close()

```
  /usr/include/sys/socket.h

struct  linger {
        int     l_onoff;                /* option on/off */
        int     l_linger;               /* linger time */
};
   l_onoff=0 -- default close
   l_onoff=1 and l_linger =0 -- connection aborted (RST), data discarded
   l_onoff=1 and l_linger =N -- process waits til all data sent and ACK'd,
         or until the timer (N) expires  (N secs or .01 secs)
```

- shutdown() with SHUT_RD recvd data discarded, can still send
- shutdown() with SHUT_WR no more sends, can still recv, send buffer sent, then FIN

be sure parent and child have closed socket

TCP data flow

- SYN establishes max segment (MSS) for each side
- send no segment bigger than MSS
- send up to receivers window size
- sender holds (buffers) segment til ACK received
- sender sets a timer, if no ACK, retransmit
- receiver ACKs cumulative segments, no selective ACKs
- ACKs may piggy-back on data being sent by the receiver

more later

TCP echo

assignment 3

```
20:09:41.682161 CETUS2A.56787 > DENEB.echo: S
    1912773626:1912773626(0) win 8760 <mss 1460> (DF)
20:09:41.683255 DENEB.echo > CETUS2A.56787: S
    911680000:911680000(0) ack 1912773627 win 8192 <mss 1460>
20:09:41.683470 CETUS2A.56787 > DENEB.echo: . ack 1 win 8760 (DF)
20:09:41.684684 CETUS2A.56787 > DENEB.echo: P 1:1025(1024)
   ack 1 win 8760 (DF)
20:09:41.755129 DENEB.echo > CETUS2A.56787: . ack 1025 win 7168
20:09:41.775877 DENEB.echo > CETUS2A.56787: P 1:1025(1024)
   ack 1025 win 8192
20:09:41.777161 CETUS2A.56787 > DENEB.echo: P 1025:2049(1024)
   ack 1025 win 8760 (DF)
20:09:41.779602 DENEB.echo > CETUS2A.56787: P 1025:2049(1024)
   ack 2049 win 8192
20:09:41.780919 CETUS2A.56787 > DENEB.echo: P 2049:3073(1024)
   ack 2049 win 8760 (DF)
...
20:09:42.132902 CETUS2A.56787 > DENEB.echo: P 101377:102401(1024)
   ack 101377 win 8760 (DF)
20:09:42.135123 DENEB.echo > CETUS2A.56787: P 101377:102401(1024)
   ack 102401 win 8192
20:09:42.162708 CETUS2A.56787 > DENEB.echo: F 102401:102401(0)
   ack 102401 win 8760 (DF)
20:09:42.163231 DENEB.echo > CETUS2A.56787: . ack 102402 win 8192
20:09:42.172718 DENEB.echo > CETUS2A.56787: F 102401:102401(0)
   ack 102402 win 8192
20:09:42.172953 CETUS2A.56787 > DENEB.echo: . ack 102402 win 8760 (DF)
```


assignment 4

```
20:03:58.089702 WHISPER.1401 > 128.169.95.255.18765: udp 8
20:04:08.089878 WHISPER.1401 > 128.169.95.255.18765: udp 8
20:04:18.090169 WHISPER.1401 > 128.169.95.255.18765: udp 8
20:04:18.096554 arp who-has WHISPER tell DENEB
20:04:18.097134 arp reply WHISPER is-at 0:c0:4f:6b:a5:52
20:04:18.097438 DENEB.1057 > WHISPER.8628: S
  870080000:870080000(0) win 8192 <mss 1460>
20:04:18.097898 WHISPER.8628 > DENEB.1057: S
  2836632427:2836632427(0) ack 870080001 win 32120 <mss 1460> (DF)
20:04:18.098495 DENEB.1057 > WHISPER.8628: . ack 1 win 8192
20:04:18.099422 DENEB.1057 > WHISPER.8628: P 1:73(72) ack 1 win 8192
20:04:18.099769 WHISPER.8628 > DENEB.1057: . ack 73 win 32120 (DF)
20:04:18.099997 DENEB.1057 > WHISPER.8628: F 73:73(0) ack 1 win 8192
20:04:18.100218 WHISPER.8628 > DENEB.1057: . ack 74 win 32120 (DF)
20:04:18.108166 WHISPER.8628 > DENEB.1057: F 1:1(0) ack 74 win 32120 (DF)
20:04:18.108426 WHISPER.4623 > DENEB.18765: S 2
  830959293:2830959293(0) win 32120
  <mss 1460,sackOK,timestamp 358537759 0,nop,wscale 0> (DF)
20:04:18.109041 DENEB.1057 > WHISPER.8628: . ack 2 win 8192
20:04:18.109824 DENEB.18765 > WHISPER.4623: S
   870144000:870144000(0) ack 2830959294 win 8192 <mss 1460>
20:04:18.110133 WHISPER.4623 > DENEB.18765: . ack 1 win 32120 (DF)
20:04:18.110367 WHISPER.4623 > DENEB.18765: P 1:9(8) ack 1 win 32120 (DF)
20:04:18.133320 DENEB.18765 > WHISPER.4623: F 1:1(0) ack 9 win 8192
20:04:18.133642 WHISPER.4623 > DENEB.18765: . ack 2 win 32120 (DF)
20:04:18.133884 WHISPER.4623 > DENEB.18765: F 9:9(0) ack 2 win 32120 (DF)
20:04:18.134457 DENEB.18765 > WHISPER.4623: . ack 10 win 8192
```


TCP options

- appear only in SYN sequence
- variable number of options, padded to 4-byte boundary
- newer options, both sides must support
- 1-byte type code

```
Type
 0    end of list
 1    no-op
 2   len(4),MSS (2 bytes)    max segment size
 3   len(3),shiftcount       window scale factor
 4   2 permitted             selective ACK (experimental)
 8   len(10),timestamp(4),timestamp(4) timestamp
```

option dump

```
20:45:28.223288 WHISPER.4624 > DENEB.echo: S
   1163963750:1163963750(0) win 32120
  <mss 1460,sackOK,timestamp 358784764 0,nop,wscale 0> (DF)
          4500 003c 9549 4000 4006 e80d 80a9 5dc8
          80a9 5e4a 1210 0007 4560 ad66 0000 0000

          a002 7d78 53ed 0000 0204 05b4 0402 080a

          1562 9efc 0000 0000 0103 0300

20:45:28.224300 DENEB.echo > WHISPER.4624: S
  1186688000:1186688000(0) ack 1163963751 win 8192 <mss 1460>
          4500 002c 09fb 0000 3c06 b76c 80a9 5e4a
          80a9 5dc8 0007 1210 46bb 6c00 4560 ad67
          6012 2000 0317 0000 0204 05b4 0001

20:45:28.224597 WHISPER.4624 > DENEB.echo: . ack 1 win 32120 (DF)
```


MSS

Maximum Segment Size option

- MSS sent with SYN ($MSS = MTU - 40$)
- if no MSS, 536 (often for non-local)
- biggest segment receiver will accept
- sender won't send anything bigger than MSS to receiver
- can be different for each end
- TCP's attempt to avoid fragmentation
- intermediate routers could still cause fragmentation

flow management

- sequenced and ACK'd segments
- flow control with sliding windows
- congestion control with Nagle, delayed ACK, slow start

TCP flows

- stream is ordered by sequence number
- receiver ACK's last byte seen
- receiver can delay ACK, ACK several segments with one ACK
- ACK's try to piggyback on data flowing back
- no selective ACK, out of sequence segment is not ACK'd, previous ACK is re-sent til missing segment arrives
- receiver usually keeps out of seqence segments
- sender must hold segment in its buffers til it's ACK'd

TCP flow control

- 16-bit window field: how much space is available at receiver
- initial window is SO_RCVBUF
- sliding window, amount of space relative to ack. number
- each side advertises its current receive window
- sender keeps track of how much un-ack'd data is in "flight"
- window may shrink if receving application isn't reading fast enough
- window could go to zero, sender stops
- throughput improves with bigger buffers (SO_SNDBUF and SO_RCVBUF socket options)

0 window

If receive window goes to 0 and receiver isn't sending data, then sender continues to send 1-byte packets to detect when window opens again. Controlled by **persist timer**.

```
15:25:07.999439 thdsun.1574 > victory.7654: P 1:1025(1024) ack 1 win 16384
15:25:07.999439 thdsun.1574 > victory.7654: . 1025:2421(1396) ack 1 win 16384
15:25:08.009439 thdsun.1574 > victory.7654: . 2421:3817(1396) ack 1 win 16384
15:25:08.009439 thdsun.1574 > victory.7654: . 3817:5213(1396) ack 1 win 16384
15:25:08.009439 thdsun.1574 > victory.7654: . 5213:6609(1396) ack 1 win 16384
15:25:08.009439 victory.7654 > thdsun.1574: . ack 6609 win 27920 (DF)
15:25:08.009439 thdsun.1574 > victory.7654: . 6609:8005(1396) ack 1 win 16384
15:25:08.009439 thdsun.1574 > victory.7654: . 8005:9401(1396) ack 1 win 16384
...
15:25:08.049441 thdsun.1574 > victory.7654: . 44301:45697(1396) ack 1 win 1638
15:25:08.049441 victory.7654 > thdsun.1574: . ack 41509 win 5584 (DF)
15:25:08.059441 thdsun.1574 > victory.7654: . 45697:47093(1396) ack 1 win 1638
15:25:08.059441 victory.7654 > thdsun.1574: . ack 45697 win 4188 (DF)
15:25:08.069441 thdsun.1574 > victory.7654: . 47093:48489(1396) ack 1 win 1638
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)
```


persist timer

indefintely probe receiver ...

```
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)
```

TCP avoids small window updates (silly window syndrome)

lost connection

Sun OS retries

```
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

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

lost connection

solaris

```
13:30:09.443252 KENNER.52182 > victory.7654: P 7:13(6) ack 8 win 9772 (DF)
13:30:09.443252 victory.7654 > KENNER.52182: P 8:14(6) ack 13 win 32736 (DF)
13:30:09.493254 KENNER.52182 > victory.7654: . ack 14 win 9772 (DF)

13:30:41.074433 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
13:30:42.504486 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
13:30:45.374593 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
13:30:51.114807 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
13:31:02.595236 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
13:31:25.546092 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
13:32:11.447804 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
13:33:07.709902 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
13:34:03.961999 KENNER.52182 > victory.7654: P 13:19(6) ack 14 win 9772 (DF)
```


RTT

roundtrip time

- TCP starts a timer for every data packet sent (not for ACK-only)
- TCP continually estimates RTT $RTT = .9 \times RTT + .1 \times latestRTT$
- ACK received, timer cancelled, have latestRTT
- 3 duplicate ACK's received, retransmit lost segment
- timer expires, exponential backoff using RTT
- RTT needs to reflect actual time
- too small and you may add to congestion
- too big and you may slow application

RTT equation now includes mean deviation, RFC 1222

congestion control

- avoid tiny grams
- exponential backoff
- slow start

delayed ACK, RFC 813

```
wait (up to 200 ms) to send ACK if no data to send
send ACK if 2nd full segment arrives or have data to send
```

Nagle algorithm (disable with TCP_NODELAY sockopt)

```
can have only one oustanding un-ACK'd segment
no additional small segments can be sent til ACK recvd
sender will accumulate data to send (interactive sessions)
```


assignment 3

sending 2K packet to echo server, we get Nagle'd and slow ACK'd

```
  menkar echoing 2k packets to whisper
39.975513 MENKAR.2319 > whisper.8989: . 1:1461(1460) ack 1 win 8192
39.975562 whisper.8989 > MENKAR.2319: . ack 1461 win 30660 (DF)
39.976000 MENKAR.2319 > whisper.8989: P 1461:2001(540) ack 1 win 8192
39.978696 whisper.8989 > MENKAR.2319: . 1:1461(1460) ack 2001 win 32120 (D

40.178545 MENKAR.2319 > whisper.8989: . ack 1461 win 8192
40.178591 whisper.8989 > MENKAR.2319: P 1461:2001(540) ack 2001 win 32120
40.183020 MENKAR.2319 > whisper.8989: . 2001:3461(1460) ack 2001 win 8192
40.183134 whisper.8989 > MENKAR.2319: P 2001:3461(1460) ack 3461 win 32120
40.183507 MENKAR.2319 > whisper.8989: P 3461:4001(540) ack 2001 win 8192
40.194201 whisper.8989 > MENKAR.2319: . ack 4001 win 32120 (DF)

40.378562 MENKAR.2319 > whisper.8989: . ack 3461 win 8192
40.378596 whisper.8989 > MENKAR.2319: P 3461:4001(540) ack 4001 win 32120
40.383012 MENKAR.2319 > whisper.8989: . 4001:5461(1460) ack 4001 win 8192
40.383118 whisper.8989 > MENKAR.2319: P 4001:5461(1460) ack 5461 win 32120
40.383501 MENKAR.2319 > whisper.8989: P 5461:6001(540) ack 4001 win 8192
40.394205 whisper.8989 > MENKAR.2319: . ack 6001 win 32120 (DF)
```

2KBytes every .2 seconds

TCP_NODELAY on server would fix it

SO_DEBUG

- setsockopt() with SO_DEBUG
- kernel trace of messages in small circular kernel buffer
- use **trpt** to display buffer for TCP trace (cryptic)
- includes TCP state info
- you can add SO_DEBUG as an option to your own applications
- many clients support debug (telnet, ftp, ttcp)
- only SunOS ?

ttcp/trpt

```
  FDDI (MTU 4352) ttcp between two Sun's -- slow???
0.36 SYN_SENT:output [677d2a00..677d2a04)@0(win=1000)<SYN> -> SYN_SENT
0.36 CLOSED:user CONNECT -> SYN_SENT
0.36 SYN_SENT:input 2e93d400@677d2a01(win=1000)<SYN,ACK> -> E
0.36 E:output 677d2a01@2e93d401(win=1000)<ACK> -> ESTABLISHED
0.37 E:output [677d2a01..677d2e01)@2e93d401(win=1000)<ACK,PUSH> -> ESTABLISHED
0.37 E:output [677d2e01..677d3601)@2e93d401(win=1000)<ACK,PUSH> -> ESTABLISHED
0.37 E:input 2e93d401@677d2e01(win=8000)<ACK> -> ESTABLISHED

0.54 E:input 2e93d401@677d3601(win=8000)<ACK> -> ESTABLISHED
0.54 E:output [677d3601..677d3e01)@2e93d401(win=1000)<ACK,PUSH> -> ESTABLISHED

0.74 E:input 2e93d401@677d3e01(win=8000)<ACK> -> ESTABLISHED
0.74 E:output [677d3e01..677d4601)@2e93d401(win=1000)<ACK,PUSH> -> ESTABLISHED

0.94 E:input 2e93d401@677d4601(win=8000)<ACK> -> ESTABLISHED
0.94 E:output [677d4601..677d4e01)@2e93d401(win=1000)<ACK,PUSH> -> ESTABLISHED
```

SNDBUF should always be at least 3*MSS

limits & timers

/usr/include/netinet/tcp_timers.h

- snd/rcv bufs: 8K
- max UDP pkt: 9000
- max settable: 52,400 (SunOS), 64K
- connect timeout: 75s
- delayed-ACK timeout: 200ms
- keepalive: 2hr+
- retransmit: 5-9+ minutes
- close wait: 30s (2MSL)
- 0-window persist: forever @ 60s
- IP frag assembly: 30s

TCP uses a 200ms and 500ms timer

netstat -s

```
TCP     tcpRtoAlgorithm     =     4     tcpRtoMin           =   200
        tcpRtoMax           = 60000     tcpMaxConn          =    -1
        tcpActiveOpens      = 86813     tcpPassiveOpens     =589652
        tcpAttemptFails     =  2201     tcpEstabResets      = 31590
        tcpCurrEstab        =    80     tcpOutSegs          =96278649
        tcpOutDataSegs      =86696691   tcpOutDataBytes     =284330482
        tcpRetransSegs      =806887     tcpRetransBytes     =592177765
        tcpOutAck           =9476125    tcpOutAckDelayed    =4754620
        tcpOutUrg           =   379     tcpOutWinUpdate     =   365
        tcpOutWinProbe      = 14270     tcpOutControl       =1490125
        tcpOutRsts          = 79302     tcpOutFastRetrans   = 67374
        tcpInSegs           =84971880
        tcpInAckSegs        =67297742   tcpInAckBytes       =239210115
        tcpInDupAck         =1188481    tcpInAckUnsent      =     0
        tcpInInorderSegs    =56656001   tcpInInorderBytes   =1107905274
        tcpInUnorderSegs    = 27191     tcpInUnorderBytes   =9233710
        tcpInDupSegs        = 16578     tcpInDupBytes       =6707303
        tcpInPartDupSegs    =   913     tcpInPartDupBytes   =372814
        tcpInPastWinSegs    =   163     tcpInPastWinBytes   =134543765
        tcpInWinProbe       =   165     tcpInWinUpdate      = 12141
        tcpInClosed         = 43275     tcpRttNoUpdate      =390895
        tcpRttUpdate        =7448238    tcpTimRetrans       =1271701
        tcpTimRetransDrop   =  1447     tcpTimKeepalive     =  1988
        tcpTimKeepaliveProbe=   906     tcpTimKeepaliveDrop =    34
        tcpListenDrop       =    88     tcpListenDropQ0     =     0
        tcpHalfOpenDrop     =     0
```

socket options

- **SO_RCVBUF SO_SNDBUF** socket buffers (performance enhancers)
- **SO_LINGER** change close behavior
- **SO_REUSEADDR** avoid "port in use", TIME_WAIT
- **TCP_NODELAY** disable Nagle
- **SO_KEEPALIVE** 2hr idle check
- **SO_DEBUG** enable kernel trace

next time

TCP vs UDP