Understanding TCP connection management
TCP connection

```
Application (client)
  Socket
  TCP software

Application (server)
  Socket
  TCP software

INTERNET
```

“Logical” connection only end hosts are aware!

State variables:
- conn status
- MSS
- windows
- ...

buffer space
normally 4 to 16 Kbytes
64+ Kbytes possible

Connection described by client&server status
Connection SET-UP duty:
1) initializes state variables
2) reserves buffer space
Connection establishment: simplest approach (non TCP)

- Connection request
- Connection granted
- Transmit data

G.Bianchi, G.Neglia, V.Mancuso
Delayed duplicate problem

Application: transactional (sell 100000$ stocks)

What is this?
Oh my God!
Too late!!!

Selling other 100000$ stocks!!!!!
Solution: three way handshake
Tomlinson 1975

Connection request (seq=X)

Connection granted (seq=Y, ack=X)

Acknowledge + data (seq=X, ack=Y)
Delayed duplicate detection

USER

- SEQ X
- SEQ Y, ACK X
- Data SEQ X, ACK Y
- duplicate
- SEQ X
- SEQ Z, ACK X
- duplicate
- Data SEQ X, ACK Y
- Reject SEQ X, ACK Z

BANK

- Application: transactional (selling stocks)
- What is this? Should be SEQ X, ACK Z!!!! STOP...
- ??? What a case: request with same indicator X? anyway...
- What is this??? Should be SEQ X, ACK Z!!!! STOP...
- Ah ah! Got the problem!
SYN (synchronize sequence numbers): used to open connection

- SYN present: this host is setting up a connection
- SEQ with SYN: means initial sequence number (ISN)
- data bytes numbered from ISN+1.

FIN: no more data to send

- used to close connection

...more later about connection closing...
Three way handshake in TCP

SRC

Connection request (SYN, ISN=100)

Connection granted (SYN, ISN=350, ACK=101)

Data segment (seq=101, ACK=351)

DEST

Full duplex connection: opened in both ways

SRC: performs ACTIVE OPEN

DEST: Performs PASSIVE OPEN

G.Bianchi, G.Neglia, V.Mancuso
Initial Sequence Number

- Should change in time
  - RFC 793 (but not all implementations are conforming) suggests to generate ISN as a sample of a 32 bit counter incrementing at 4us rate

- transmitted whenever SYN (Synchronize sequence numbers) flag active
  - note that both src and dest transmit THEIR initial sequence number (remember: full duplex)

- Data Bytes numbered from ISN+1
  - necessary to allow SYN segment ack
Maximum Segment Size - MSS

- Announced at setup by both ends.
- Lower value selected.
- MSS sent in the Options header of the SYN segment
  - clearly cannot (=ignored if happens) send MSS in a non SYN segment, as connection has been already setup
  - when SYN has no MSS, default value 536 used
- goal: the larger the MSS, the better...
  - until fragmentation occurs
  - e.g. if host is on ethernet, sets MSS=1460
    - 1500 max ethernet size - 20 IP header - 20 TCP header
MSS advertise

CLIENT (C_MSS)

Conn request (C_MSS, SYN, seq=C_ISN)

Use recv MSS

time

SERVER (S_MSS)

Conn granted (MSS, SYN, seq=S_ISN, ack=C_ISN+1)

If (S_MSS < C_MSS)
MSS = S_MSS;
else MSS = C_MSS;

Acknowledge (seq=C_ISN+1, ack=S_ISN+1)

time

Does not avoid fragmentation to occur WITHIN the network!!

G.Bianchi, G.Neglia, V.Mancuso
connection closing: an impossible problem!

Suitable Timeout settings & Extension to three/four/plus way handshake do not solve!!

TIMEOUT

CLOSE

ACK

TIMEOUT

CLOSE

OK: he has closed.
I close too: bye bye.

CLOSED

G.Bianchi, G.Neglia, V.Mancuso
Connection closing in TCP
since it is impossible problem, use simples solution (two way handshake)

→ Since connection full duplex, necessary two half-closes (each a two-way handshake) originating by both sides
→ close notified with FIN flag on
→ FIN segment ACK-ed as usual
Half close
may close one direction only - seldomly used

- Supported by system call shutdown instead of close

Application close → FIN

App read ← ACK of FIN

App write → data

Ack of data ←

EOF to app → FIN

App close ←

ACK of FIN

TIME_WAIT (30s - 2m)

G.Bianchi, G.Neglia, V.Mancuso
Connection states - Client

- **CLOSED**: client application initiates a TCP connection
- **TIME_WAIT**: wait 30 seconds
- **FIN_WAIT_2**: receive FIN, send ACK
- **FIN_WAIT_1**: receive ACK, send nothing
- **SYN_SENT**: receive SYN & ACK, send ACK
- **ESTABLISHED**: client application initiates close connection

G.Bianchi, G.Neglia, V.Mancuso
Connection States - Server

- **CLOSED**: server application creates a listen socket
- **LISTEN**: receive SYN, send SYN & ACK
- **SYN_RCVD**: receive ACK, send nothing
- **ESTABLISHED**: receive FIN, send ACK
- **CLOSE_WAIT**: send FIN
- **LAST_ACK**: receive ACK, send nothing
Why TIME_WAIT?

- **MSL** *(Maximum Segment Lifetime)*: maximum time a segment can live in the Internet
  - no timers on IP packets! Only hop counter
  - RFC 793 specifies MSL=120s, but each implementation has its own value (from 30s to 120s)

- **TIME_WAIT state**: $2 \times$ MSL
  - allows to “clean” the network of delayed packets belonging to the connection
  - $2 \times$ MSL because a lost FIN_ACK implies a new FIN from server

- during **TIME_WAIT** conn sock pair reserved
  - many implementations even more restrictive (local port non reusable)
  - clearly this may be a serious problem when restarting server daemon (must pause from 1 to 4 minutes…)
RST (Reset)

- sent whenever a segment arrives and does not apparently belong to the connection
- typical RST case: connection request arriving to port not in use

Sending RST within an active connection:

- allows *aborting release* of connection (versus *orderly release*)
  - any queued data thrown away
  - receiver of RST can notify app that abort was performed at other end