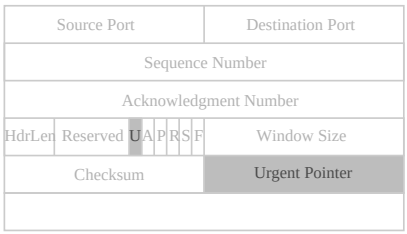
TCP Urgent Data



- Urgent Pointer plus Sequence Number indicates end of some URGENT data in the packet

TCP Urgent Data (2)

- Seq=200000 UrgPtr=123
  - The byte with sequence id 200123
    - the last of urgent data
- Urgent Pointer valid only
  - when the URG bit is set
  - URG remains set
    - until packet containing urgent data transmitted
- "Urgent" data is just data that the receiver needs to know has arrived as soon as possible
- Receiving URG switches receiver
  - into urgent processing mode
  - until urgent data has been consumed
  - then back to normal mode

TCP Urgent Spec

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment.

The urgent pointer points to the sequence number of the octet following the urgent data.
This field is only be interpreted in segments with the URG control bit set.
[...]
If the urgent flag is set, then SND.UP <- SND.NXT-1 and set the urgent pointer in the outgoing segments.

- SND.UP -- the urgent pointer
- SND.NXT -- Seq Number of next byte to send

1981 (RFC793)

TCP Correction RFC1122

4.2.2.4 Urgent Pointer: RFC-793 Section 3.1

The second sentence is in error: the urgent pointer points to the sequence number of the LAST octet (not LAST+1) in a sequence of urgent data. The description on page 56 (last sentence) is correct.

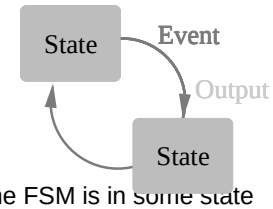
1989 (RFC1122)

- Text descriptions of protocols can be
  - ambiguous
  - contradictory
- TCP implementations
  - still need to deal with this
  - Some early implementations used each definition

Finite State Machine

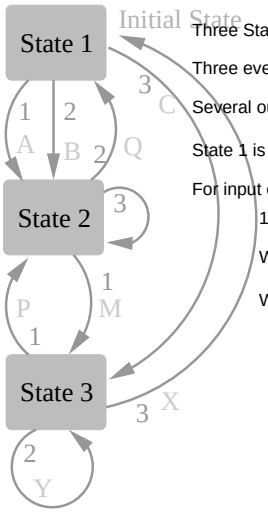
- More formal method of speificiation
  - Often depicted using drawing
- Some number of states defined
  - drawn as circles or rectangles
- In each state
  - specific events can occur
    - inputs
    - timeouts
  - cause transition to another state
  - cause output action to occur

FSM Basics



- The FSM is in some state
- An event occurs
  - drawn beside a line
  - shows transition to a new state
- Some output may accompany transition
- Transition may return to the same state
  - Or may transition to another state

FSM (example)



Three States 1 2 and 3

Three events that occur 1 2 and 3

Several output actions

State 1 is the initial state

For input events

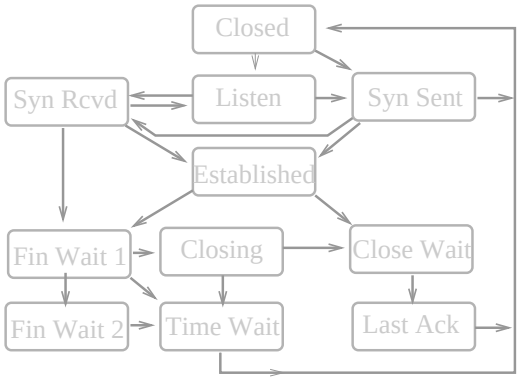
1 1 1 2 3 2 3 1 2 2 3

What states does FSM pass through?

What output actions are performed?

$1_A 2_M 3_P 2_Q 1_C 3_Y 3_X 1_A 2_Q 1_B 2_-$ 2

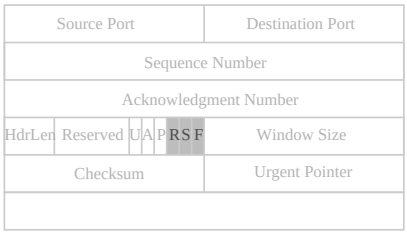
TCP Connections (States)



FSM of TCP connection machinery

- Inputs & Outputs to come later

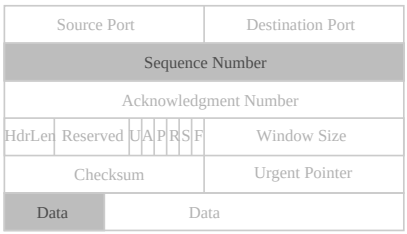
TCP Connections (Hdr Fields)



- RST - RESET
- SYN - SYNCHRONISE
- FIN - FINISH

- SYN & FIN consume sequence numbers

TCP SYN/FIN & Seq Numbers



- SYN + 1 Data Byte
  - Two sequence numbers consumed
  - Sequence number in header
    - is sequence number of SYN
  - Data gets next sequence number

TCP Specification

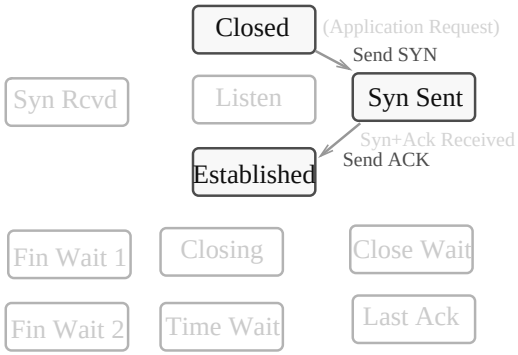
- FSM not used to specify everything

The sequence number of the first data octet in this segment (except when SYN is present).
If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

TCP Connections (Identity)

- TCP Connection identified by 4-tuple
  - Source IP Address
  - Destination IP Address
  - Source TCP Port Number
  - Destination TCP Port Number
- All remain constant for one TCP connection

TCP Client Open

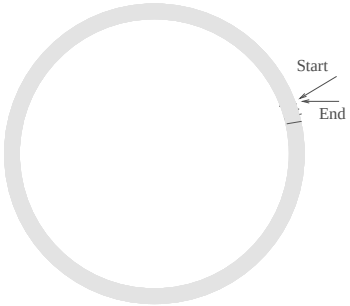


- 3-way Handshake
- SYN SYN+ACK ACK

TCP Sequence Number Choices

- Sequence number used to acknowledge data being transferred
  - (as noted previously)
- Any increasing number works for this.
- But also used to exclude old data from previous connections that remains in the network
- For this, need sequence numbers to globally (for a node) increase over time
  - New connection not re-use the sequence numbers
    - of any recent previous connection

TCP Sequence Number



TCP Sequence Numbers

- Also used to avoid connection hijacking
  - Hijacker needs to know sequence numbers
    - to generate valid packets
  - Want seq numbers to be random
- So, want a random number that increases
  - Must be random enough
    - to avoid brute force attacks
  - Sequential enough
    - to keep out old packets

TCP Specification

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation.

We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using.

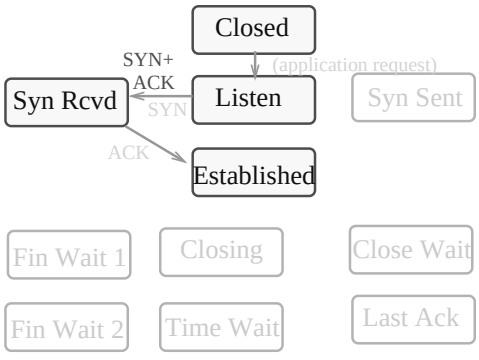
When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN.

The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4

microseconds.
Thus, the ISN cycles approximately every 4.55 hours.

Since we assume that segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours we can reasonably assume that ISN's will be unique.

TCP Server Open



- Same 3-way handshake
  - SYN SYN+ACK ACK

TCP Sequence Numbers

- Also used to avoid connection hijacking
  - Hijacker needs to know sequence numbers
    - to generate valid packets
  - Want seq numbers to be random
- So, want a random number that increases
  - Must be random enough
    - to avoid brute force attacks
  - Sequential enough
    - to keep out old packets

TCP Specification

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation.

We want to assure this, even if a TCP crashes and loses all knowledge of the sequence numbers it has been using.

When new connections are created, an initial sequence number (ISN) generator is employed which selects a new 32 bit ISN.

The generator is bound to a (possibly fictitious) 32 bit clock whose low order bit is incremented roughly every 4

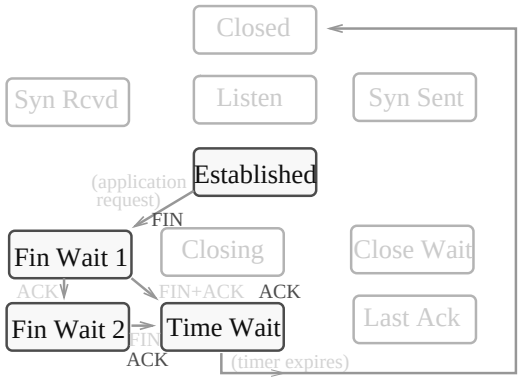
microseconds.
Thus, the ISN cycles approximately every 4.55 hours.

Since we assume that segments will stay in the network no more than the Maximum Segment Lifetime (MSL) and that the MSL is less than 4.55 hours we can reasonably assume that ISN's will be unique.

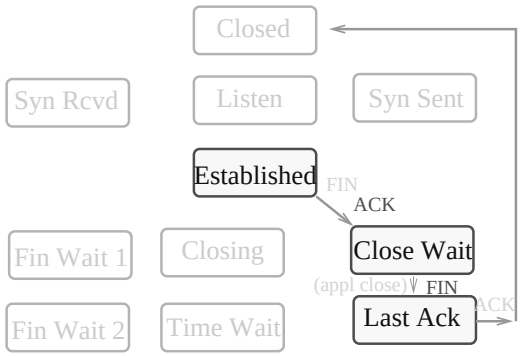
TCP Open Example

```
172.30.0.77.65291 > 172.30.0.161.9: S
    1561401491:1561401491(0)
    win 16384

172.30.0.161.9 > 172.30.0.77.65291: S
    3751259715:3751259715(0)
    ack 1561401492 win 16384

172.30.0.77.65291 > 172.30.0.161.9: .
    ack 3751259716 win 17520
```

TCP Requested Close



TCP Response Close



TCP Close Example

```
172.30.0.77.65291 > 172.30.0.161.9: F
    1561401492:1561401492(0)
    ack 3751259716 win 17520

172.30.0.161.9 > 172.30.0.77.65291: .
    ack 1561401493 win 17520

172.30.0.161.9 > 172.30.0.77.65291: F
    3751259716:3751259716(0)
    ack 1561401493 win 17520

172.30.0.77.65291 > 172.30.0.161.9: .
    ack 3751259717 win 17519
```

TCP TIME WAIT

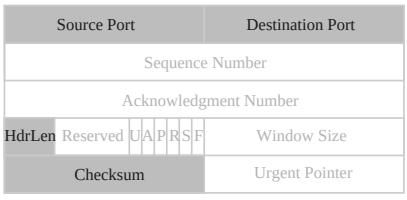
```
Active Internet connections (including servers)
Proto Local Address      Foreign Address  State
tcp   172.30.0.77.65291  172.30.0.161.9   TIME_WAIT
```

- "Connection" on client remains in TIME WAIT for twice the maximum segment lifetime
- Guards against old packets in the network

Graceful termination

- How to acknowledge the final packet ?
- Send an acknowledge ?
- But then that is the final packet ...
  - How is that acknowledged?
- Eventually, simply stop!

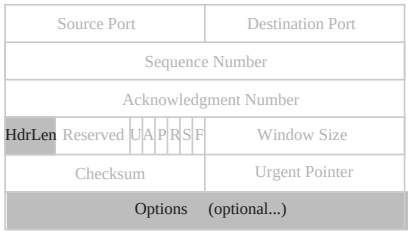
TCP (concluded)



- Source Port
- Destination Port
  - Transport address
- Checksum
  - As for UDP
    - includes pseudo-header

  - But not optional.

TCP Options



- Header Length
  - Allows size of options to be calculated
    - Counts 32 bit words in header
    - Minimum value 5 (20 bytes)
    - Maximum value 15 (60 bytes)
      - Max 40 bytes of options
- Options
  - Data not needed in every packet
  - Extensions to TCP

TCP Options

```
172.30.0.77.65486 > 172.30.0.161.9: S
   334775908:334775908(0) win 16384
   <mss 1460,nop,wscale 0,nop,nop, timestamp 11188 0>
```

- 4 different options used

- NOP
  - padding (keeps alignment of other options)

- MSS
- WSCALE
- TIMESTAMP

TCP Specification

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length.
All options are included in the checksum.
An option may begin on any octet boundary.
There are two cases for the format of an option:
Case 1: A single octet of option-kind.
Case 2: An octet of option-kind, an octet of option-length, and the actual option-data octets.
The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply.
The content of the header beyond the End-of-Option option must be header padding (i.e., zero).
A TCP must implement all options.

TCP Specification

Currently defined options include:

Kind	Length	Meaning
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

- End of Option
  - 1 byte 0
    - What follows is padding
  - Padding required to be 0
- No Operation
  - 1 byte 1

TCP Max Segment Size Opt

- Allows one TCP to tell the other the maximum segment size to send
  - Segment ~= packet
  - but one segment may be fragmented into several packets
- Maximum Segment Size
    - 4 bytes
  - Kind 2
  - Length 4
  - Value (2 bytes) MSS

TCP MSS Practice

- TCP will attempt to avoid fragmentation by advising peer of the maximum segment size
  - Segment size (within window) not usually important
- Works correctly only when low MTU links are directly connected to hosts
- Largely overtaken by PMTUD now
    - Or should be
- Option only permitted when SYN is set
  - Data not required in every packet

TCP Window Size Limitation

- Maximum window size is 64K bytes (-1)
- Once "window" size bytes of data have been sent, TCP must stall and wait for ACK of (at least) first segment before sending more
- At 56K bits per second
  - (original arpanet)
  - sending 64K bytes takes more than 9 seconds
- As long as RTT is less than 9 seconds,
  - (Round Trip Time)
  - TCP need never stall

TCP Window Size Limitation

- At 100 M bits/second
  - (current network speeds)
  - sending 64K bytes
    - takes about 5.5 milliseconds
- So RTT must be less than 5.5 milliseconds
  - to avoid stall
- OK for LAN
- WAN might easily have RTT of 500ms
  - TCP can only use about 1% of available bandwidth
- Want to extend TCP
  - to allow higher throughput
  - Add an option

TCP Window Scale Option

- Allows TCP systems to
  - discover that both implement the option
  - specify that all window values
    - should be multiplied by a power of two
- Limited to 2^14
  - So, max window is (64K - 1) * 2^14 ~= 1GB
  - Thus max of 1/4 of the entire window space
- If both TCPs send wscale option, window scaling protocol is used
- Each TCP indicates how much its window should be scaled
- If either does not send option
  - No window scaling
  - In either direction

TCP Option Usage

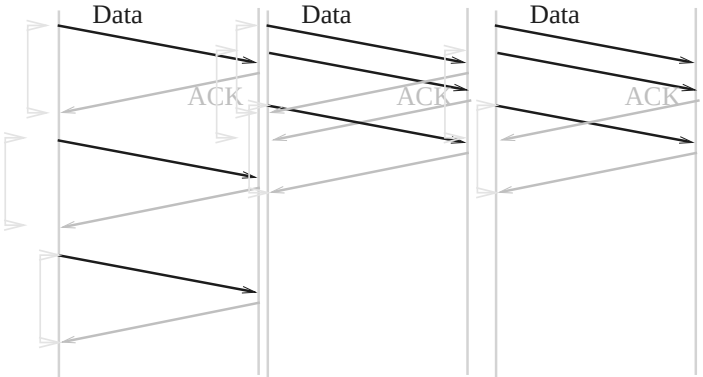
```
172.30.0.77.65486 > 172.30.0.161.9: S
   334775908:334775908(0) win 16384
   <mss 1460,nop,wscale 0,nop,nop, timestamp 11188 0>
```

- WScale == 0
  - Scale window size by 2^0
    - 2^0 == 1
    - So, no scaling
- Why is option included?
  - So other host knows
    - this TCP supports the window scale option

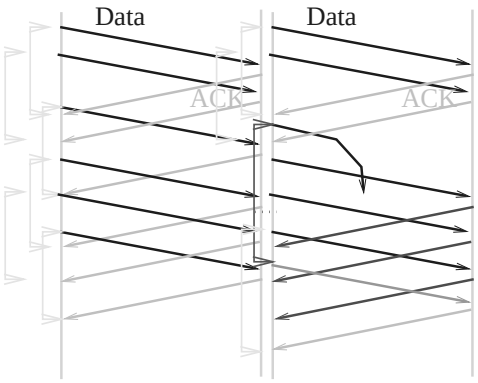
Round Trip Time Estimation

- Useful to know when to retransmit
  - if have not received ACK within the RTT
    - (plus a bit)
    - then assume packet lost
- But how to measure the RTT?
  - Measure delay between packet and its ACK
    - easy
  - But
    - Send packet
      - wait ... wait ... wait (nothing)
    - Retransmit packet
    - ACK arrives
  - Which packet was acknowledged?
    - The initial packet
      - acknowledged slower than expected
    - Or the retransmit?

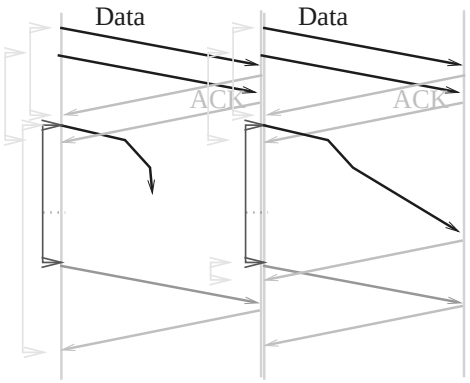
TCP RTT Measurement



TCP RTT Measurement (2)



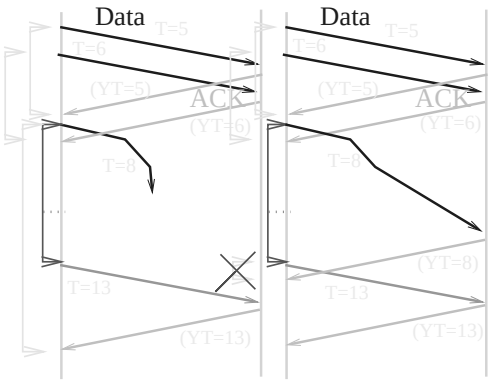
TCP RTT Measurement (3)



TCP Timestamp Option

- Each TCP can add timestamp option to every packet
- Peer TCP sends back timestamp received with each ACK
- Allows TCP to determine which packet was ACK'd
- Better than that, no need to remember when packets were sent, the returning timestamp contains that information
- Also used for long delayed old packet detection
  - extends the sequence number space

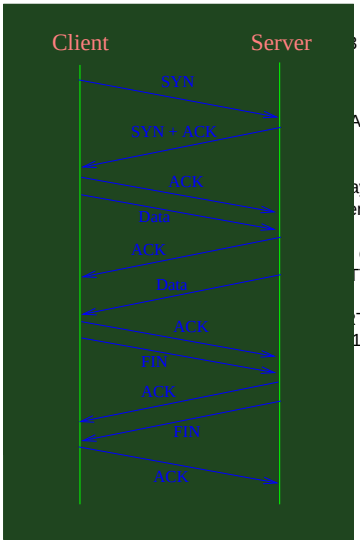
TCP RTT Measurement (tstamp)



TCP or UDP

- UDP is unreliable, not flow controlled
- TCP is reliable, has flow control
  - Often would prefer to use TCP to UDP
- But
  - Overheads are much greater

Typical TCP



3 way handshake

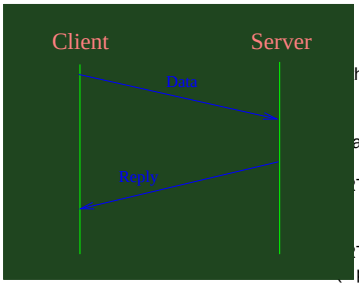
A

ay before reply
er RTT

connection
TT (but unimportant)

TT + server delay
1 packets)

UDP Alternative



he data

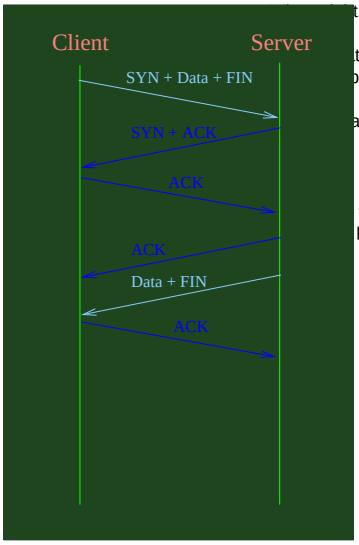
ay

TT

TT + server delay
packets)

- 2 packets instead of 11
- 1 RTT instead of 2

Minimal TCP



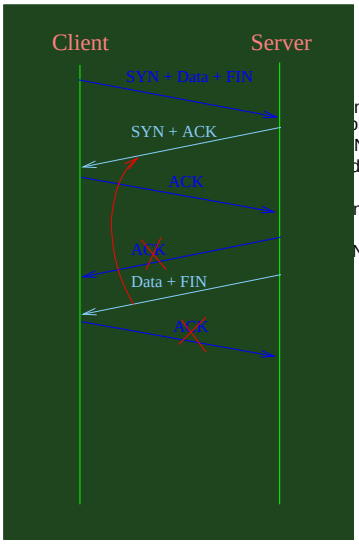
t be merged in TCP?

ta in SYN packet
ut FIN in SYN packet

a reply can carry FIN

+ server delay
packets)

TCP Cannot Do

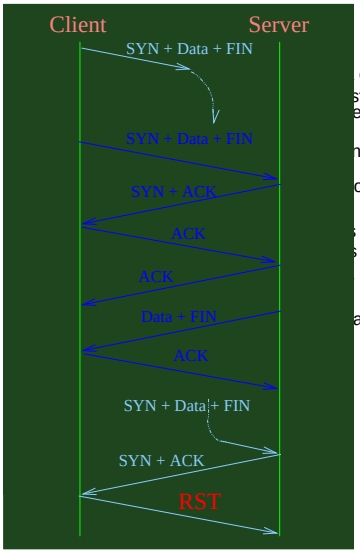


not return data with its SYN
hibits passing any data in
N to application until
dshake is complete

not known that SYN is new

N+ACK acks only the SYN

Old Duplicate SYN



client sending a SYN
st in the net and
ed

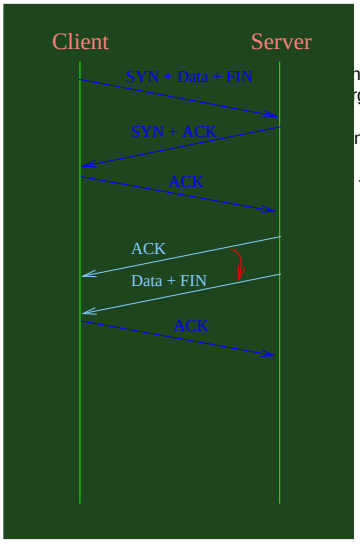
n continues as normal

ost" SYN arrives

normal SYN+ACK reply
not expecting this
RST

a were already delivered?

TCP Can Sometimes Do



n ACKs data in SYN packet)
rged with DATA reply packet

n Server Delay

+ server delay
(5 packets)

Maximum Transaction Rate

- Bounded by 2 * RTT + server delay
- But also limited by TIME WAIT state
  - No more data on same connection for 2*MSL
  - Client picks a different port number
    - different connection
- If 1000 connections / second, and MSL == 2 mins (120 secs)
  - Then 240*1000 connections in TIME WAIT state
    - Consumes lots of memory
    - (if 40 bytes/connection, almost 10MB)
- Worse! Impossible, only 65536 ports!
  - Thus limited to about 270 trans/sec