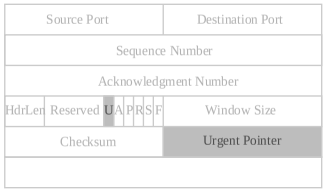
TCP Urgent Data



- 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.

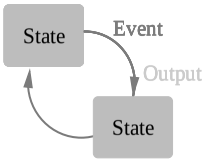
1989 (RFC1122)

- 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

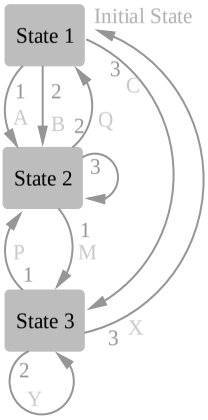
- More formal method of speificiation
  - Often depicted using drawing
- Some number of states defined
  - drawn as circles or rectangles
- In each state
  - specific events can occur
    - inputs
    - timeouts
  - cause transition to another state
  - cause output action to occur

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

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

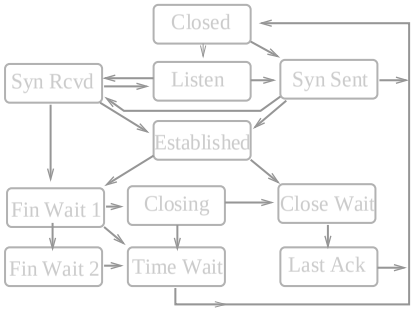
For input events

1 2 2 3 2 3 2 1 1 3

What states does FSM pass through?

What output actions are performed?

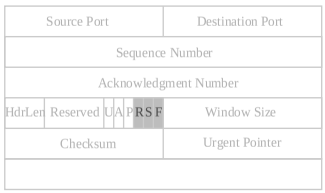
TCP Connections (States)



FSM of TCP connection machinery

- Inputs & Outputs to come later

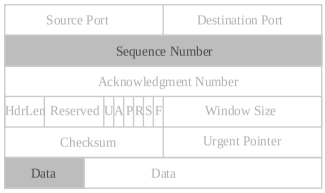
TCP Connections (Hdr Fields)



- 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

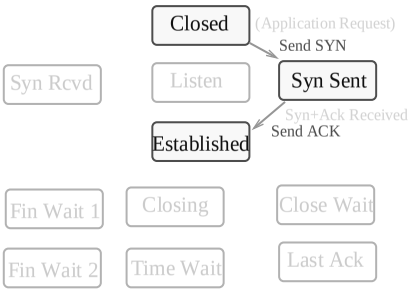
- FSM not used to specify everything

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)

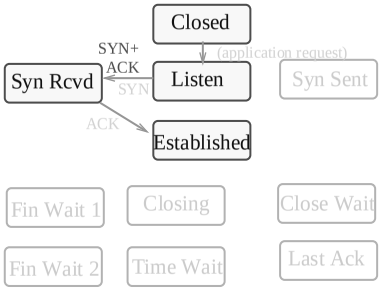
- TCP Connection identified by 4-tuple
  - Source IP Address
  - Destination IP Address
  - Source TCP Port Number
  - Destination TCP Port Number
- All remain constant for one TCP connection

TCP Client Open



- 3-way Handshake
- SYN SYN+ACK ACK

TCP Server Open

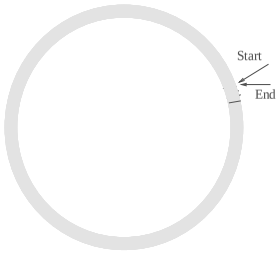


- Same 3-way handshake
  - SYN SYN+ACK ACK

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



etc

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.

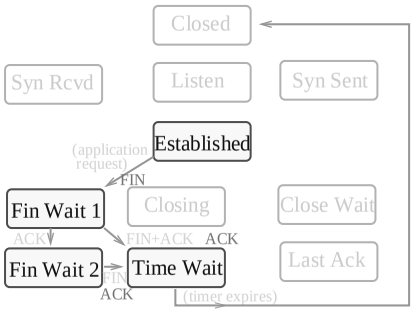
Thus, the ISN cycles approximately every 4.55 hours.

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.

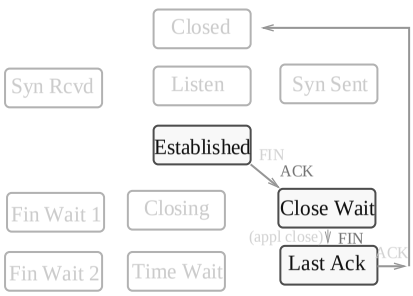
TCP Open Example

```
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
```