Reliable Byte-Stream (TCP)

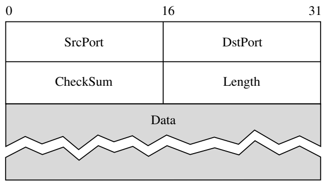
- Outline
  - Connection Establishment/Termination
  - Sliding Window Revisited
  - Flow Control
  - Adaptive Timeout

End-to-End Protocols

- Underlying best-effort network
  - drops messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay
- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to apply flow control to the sender
  - support multiple application processes on each host

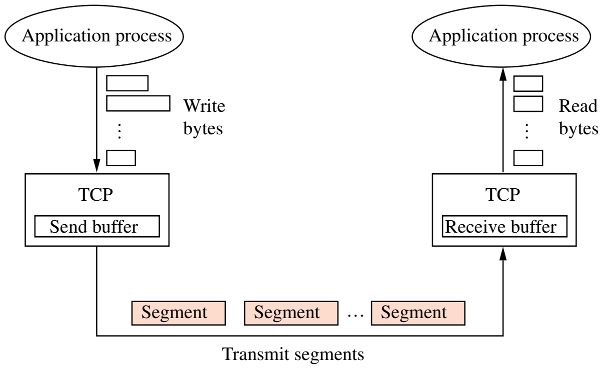
Simple Demultiplexor (UDP)

- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have *well-known* ports
  - see `/etc/services` on Unix
- Optional checksum
  - pseudo header + udp header + data
- Header format



TCP Overview

- Connection-oriented
- Byte-stream
  - sending process writes some number of bytes
  - TCP breaks into *segments* and sends via IP
  - receiving process reads some number of bytes

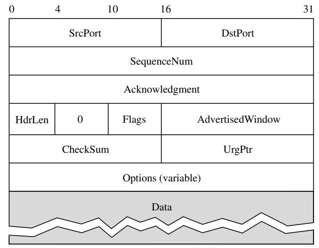


- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network

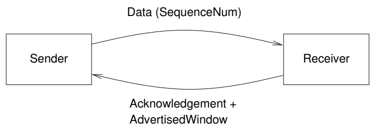
Data Link versus Transport

- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different amounts of buffering
- Potentially different network capacity
  - need to be prepared for network congestion

Segment Format

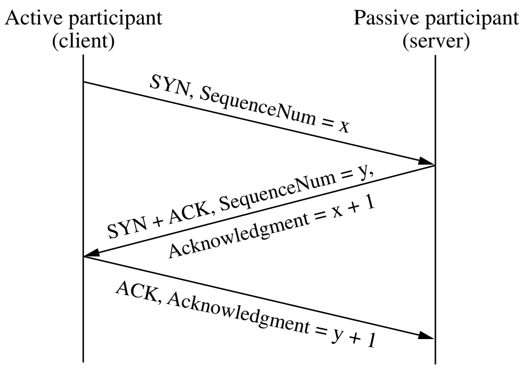


- Each connection identified with 4-tuple:
  - ⟨`SrcPort, SrcIPAddr, DstPort, DstIPAddr`⟩
- Sliding window + flow control
  - `Acknowledgment, SequenceNum, AdvertisedWindow`

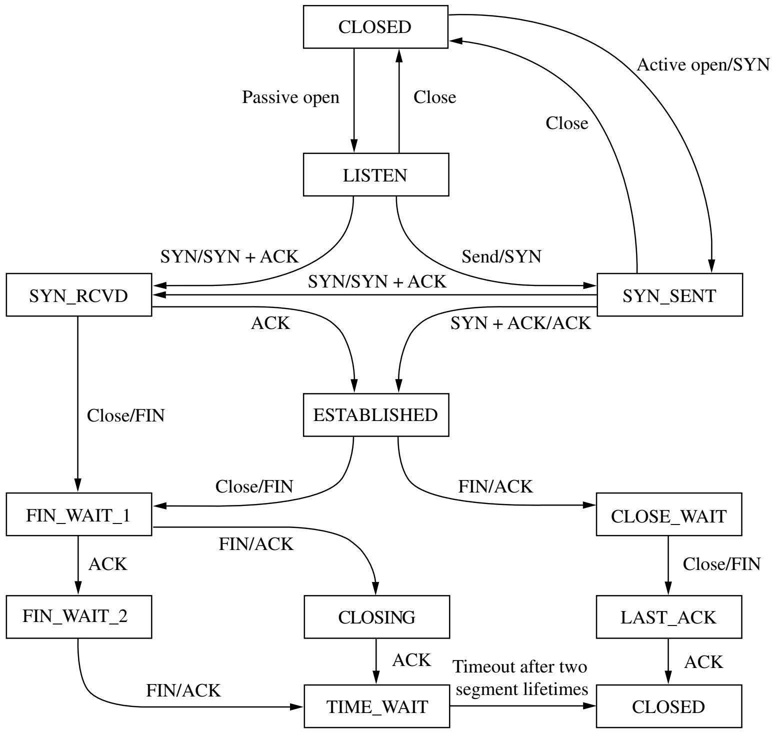


- Flags
  - `SYN, FIN, RESET, PUSH, URG, ACK`
- Checksum
  - pseudo header + tcp header + data

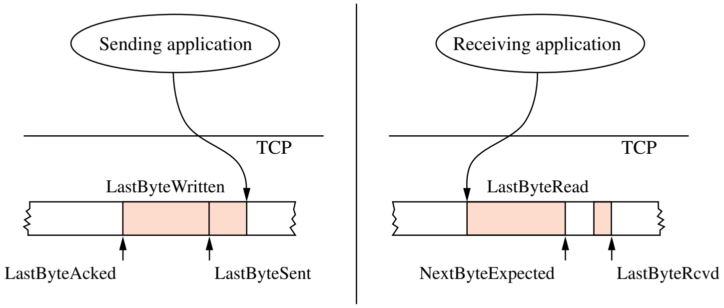
Connection Establishment and Termination



State Transition Diagram



Sliding Window Revisited



- Each byte has a sequence number
- ACKs are cumulative
- Sending side
  - `LastByteAcked` $\leq$ `LastByteSent`
  - `LastByteSent` $\leq$ `LastByteWritten`
  - bytes between `LastByteAcked` and `LastByteWritten` must be buffered
- Receiving side
  - LastByteRead < NextByteExpected
  - `NextByteExpected` $\leq$ `LastByteRcvd + 1`
  - bytes between `NextByteRead` and `LastByteRcvd` must be buffered

Flow Control

- Sender buffer size: `MaxSendBuffer`
- Receive buffer size: `MaxRcvBuffer`
- Receiving side
  - `LastByteRcvd - NextByteRead` $\leq$ `MaxRcvBuffer`
  - `AdvertisedWindow = MaxRcvBuffer - (LastByteRcvd - NextByteRead)`
- Sending side
  - `LastByteSent - LastByteAcked` $\leq$ `AdvertisedWindow`
  - `EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)`
  - `LastByteWritten - LastByteAcked` $\leq$ `MaxSendBuffer`
  - block sender if `(LastByteWritten - LastByteAcked) +` y $>$ `MaxSendBuffer`
- Always send ACK in response to an arriving data segment
- Persist when `AdvertisedWindow=0`

Keeping the Pipe Full

Wrap Around: 32-bit `SequenceNum`

Bandwidth	Time Until Wrap Around
T1 (1.5Mbps)	6.4 hours
Ethernet (10Mbps)	57 minutes
T3 (45Mbps)	13 minutes
FDDI (100Mbps)	6 minutes
STS-3 (155Mbps)	4 minutes
STS-12 (622Mbps)	55 seconds
STS-24 (1.2Gbps)	28 seconds

Bytes in Transit: 16-bit `AdvertisedWindow`

Bandwidth	Delay $\times$ Bandwidth Product
T1 (1.5Mbps)	18KB
Ethernet (10Mbps)	122KB
T3 (45Mbps)	549KB
FDDI (100Mbps)	1.2MB
STS-3 (155Mbps)	1.8MB
STS-12 (622Mbps)	7.4MB
STS-24 (1.2Gbps)	14.8MB

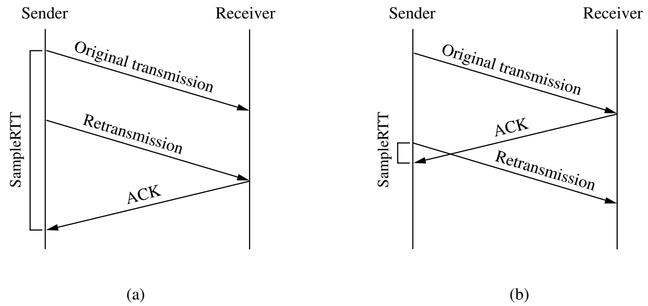
TCP Extensions

- Implemented as header options
- Store timestamp in outgoing segments
- Use 32-bit timestamp to extend sequence space (PAWS)
- Shift (scale) advertised window

Adaptive Retransmission (Original Algorithm)

- Measure `SampleRTT` for each segment/ACK pair
- Compute weighted average of RTT
  - `EstimatedRTT` = $\alpha\times$ `EstimatedRTT` + $\beta\times$ `SampleRTT`
  - where α + β = 1
  - α between 0.8 and 0.9
  - β between 0.1 and 0.2
- Set timeout based on `EstimatedRTT`
  - `TimeOut` = 2 $\times$ `EstimatedRTT`

Karn/Partridge Algorithm



- Do not sample RTT when retransmitting
- Double timeout after each retransmission

Jacobson/Karels Algorithm

- New calculation for average RTT

`Difference = SampleRTT - EstimatedRTT`

`EstimatedRTT = EstimatedRTT +` ($\delta \times$ `Difference)`

`Deviation = Deviation +` δ `( |Difference |- Deviation)`

  - where δ is a fraction between 0 and 1

- Consider variance when setting timeout value

`TimeOut =` $\mu \times$ `EstimatedRTT +` $\phi \times$ `Deviation`

  - where $\mu = 1$ and $\phi = 4$

- Notes
  - algorithm only as good as granularity of clock (500ms on Unix)
  - accurate timeout mechanism important to congestion control (later)