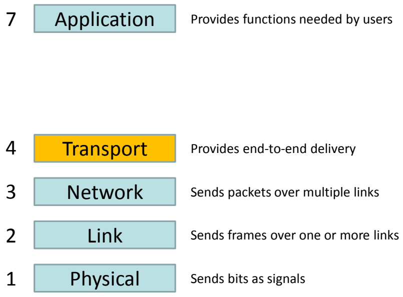
Computer Networks

Zhang, Xinyu
Fall 2014

School of Software

East China Normal University

Course Reference Model



7 Application — Provides functions needed by users

4 Transport — Provides end-to-end delivery

3 Network — Sends packets over multiple links

2 Link — Sends frames over one or more links

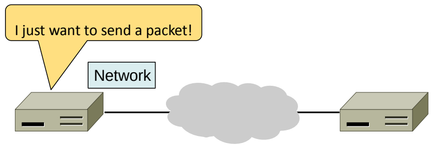
1 Physical — Sends bits as signals

User Datagram Protocols

Slides are borrowed from David Wetherall, Arvind Krishnamurthy, John Zahorjan, Washington University

Topic

- Sending messages with UDP
  - A shim layer on packets

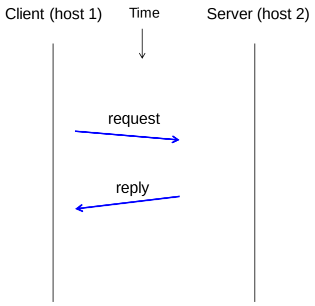


User Datagram Protocol (UDP)

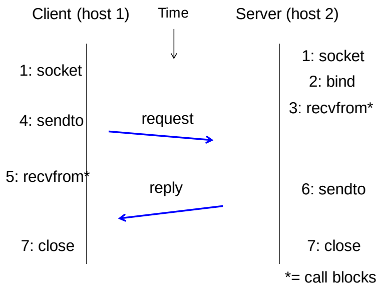
- Used by apps that don’t want reliability or bytestreams
  - Voice-over-IP (unreliable)
  - DNS (Domain Name System)
  - RPC (Remote Procedure Call), (message-oriented)
  - DHCP (bootstrapping)

(If application wants reliability and messages then it has work to do!)

Datagram Sockets



Datagram Sockets



Client (host 1)

Time

Server (host 2)

1: socket

1: socket

2: bind

3: recvfrom*

4: sendto

request

5: recvfrom*

reply

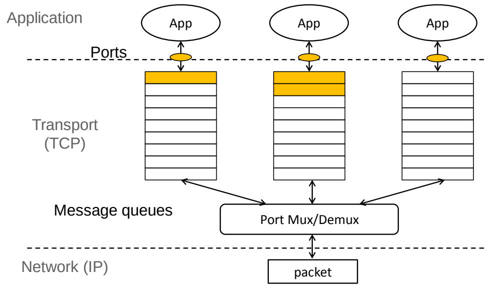
6: sendto

7: close

7: close

*= call blocks

UDP Buffering



Application

App

App

App

Ports

Transport (TCP)

Message queues

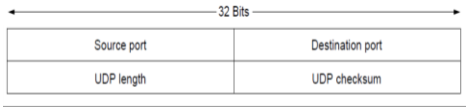
Port Mux/Demux

Network (IP)

packet

UDP Header

- Uses ports to identify sending and receiving application processes
- Datagram length up to 64K
- Checksum (16 bits) for reliability



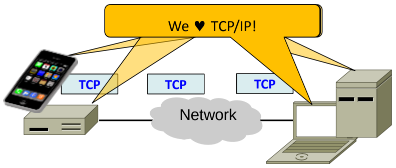
32 Bits

Source port	Destination port
UDP length	UDP checksum

Transmission Control Protocols (TCP)

Topic

- How TCP works!
  - The transport protocol used for most content on the Internet

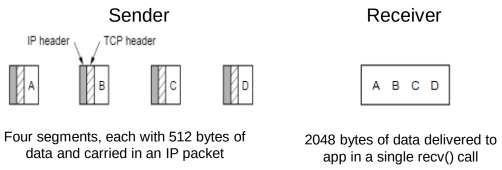


TCP Features

- A reliable bytestream service
- Based on connections
- Sliding window for reliability
  - With adaptive timeout
- Flow control for slow receivers
- Congestion control to allocate network bandwidth

Reliable Bytestream

- Message boundaries not preserved from send() to recv()
  - But reliable and ordered (receive bytes in same order as sent)

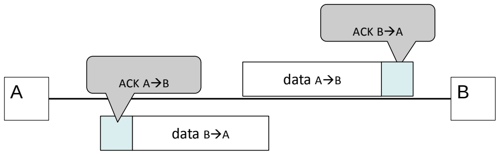

Sender
Receiver

Four segments, each with 512 bytes of data and carried in an IP packet

2048 bytes of data delivered to app in a single recv() call

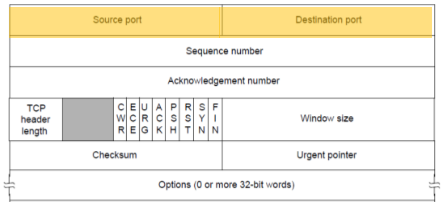
Reliable Bytestream (2)

- Bidirectional data transfer
  - Control information (e.g., ACK) piggybacks on data segments in reverse direction



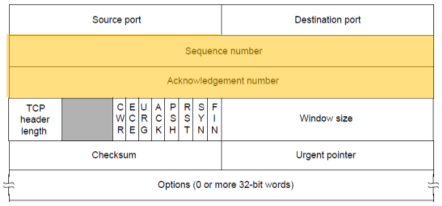
TCP Header (1)

- Ports identify apps (socket API)
  - 16-bit identifiers


Source port | Destination port
Sequence number
Acknowledgement number
TCP header length | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window size
Checksum | Urgent pointer
Options (0 or more 32-bit words)

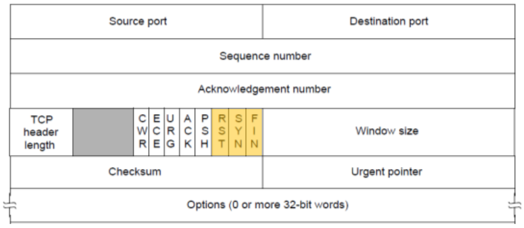
TCP Header (2)

- SEQ/ACK used for sliding window
  - Selective Repeat, with byte positions


Source port | Destination port
Sequence number
Acknowledgement number
TCP header length | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window size
Checksum | Urgent pointer
Options (0 or more 32-bit words)

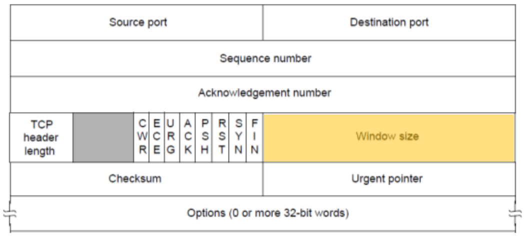
TCP Header (3)

- SYN/FIN/RST flags for connections
  - Flag indicates segment is a SYN etc.


Source port | Destination port
Sequence number
Acknowledgement number
TCP header length | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window size
Checksum | Urgent pointer
Options (0 or more 32-bit words)

TCP Header (4)

- Window size for flow control
  - Relative to ACK, and in bytes


Source port | Destination port
Sequence number
Acknowledgement number
TCP header length | CWR | ECE | URG | ACK | PSH | RST | SYN | FIN | Window size
Checksum | Urgent pointer
Options (0 or more 32-bit words)

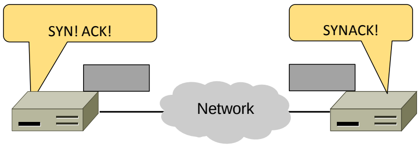
Other TCP Details

- Many, many quirks you can learn about its operation
- Biggest remaining mystery is the workings of congestion control

Connection Establishment

Topic

- How to set up connections
  - We'll see how TCP does it

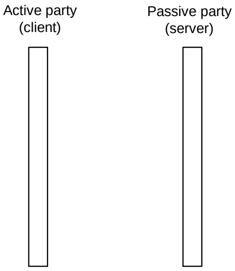


Connection Establishment

- Both sender and receiver must be ready before we start the transfer of data
  - Need to agree on a set of parameters
  - e.g., the Maximum Segment Size (MSS)
- This is signaling
  - It sets up state at the endpoints
  - Like "dialing" for a telephone call

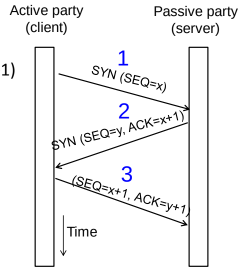
Three-Way Handshake

- Used in TCP; opens connection for data in both directions
- Each side probes the other with a fresh Initial Sequence Number (ISN)
  - Sends on a SYNchronize segment
  - Echo on an ACKnowledge segment
- Chosen to be robust even against delayed duplicates



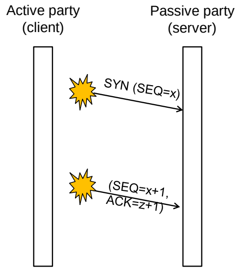
Three-Way Handshake (2)

- Three steps:
  - Client sends SYN(x)
  - Server replies with SYN(y)ACK(x+1)
  - Client replies with ACK(y+1)
  - SYNs are retransmitted if lost
- Sequence and ack numbers carried on further segments



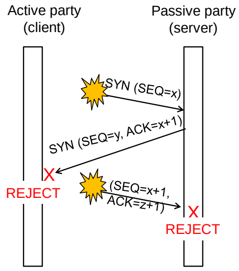
Three-Way Handshake (3)

- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  - Improbable, but anyhow …



Three-Way Handshake (4)

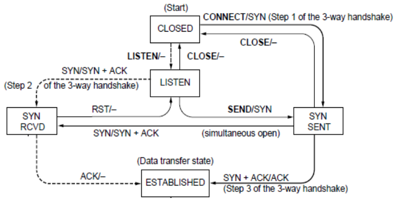
- Suppose delayed, duplicate copies of the SYN and ACK arrive at the server!
  - Improbable, but anyhow …
- Connection will be cleanly rejected on both sides



TCP Connection State Machine

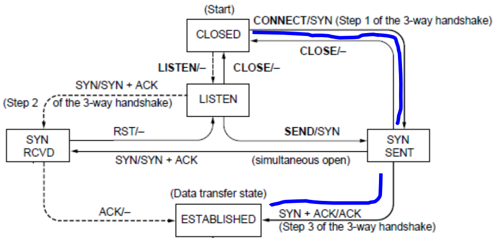
- Captures the states (rectangles) and transitions (arrows)
  - A/B means event A triggers the transition, with action B

Both parties run instances of this state machine



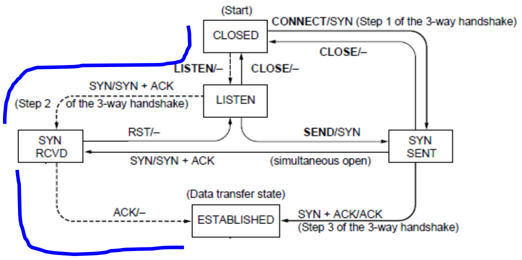
TCP Connections (2)

- Follow the path of the client:



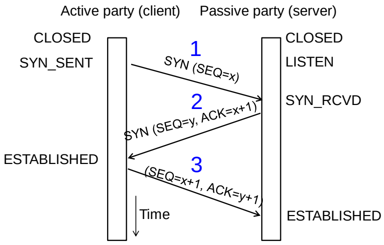
TCP Connections (3)

- And the path of the server:



TCP Connections (4)

- Again, with states …



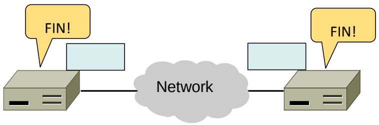
TCP Connections (5)

- Finite state machines are a useful tool to specify and check the handling of all cases that may occur
- TCP allows for simultaneous open
  - i.e., both sides open at once instead of the client-server pattern

Connection Release

Topic

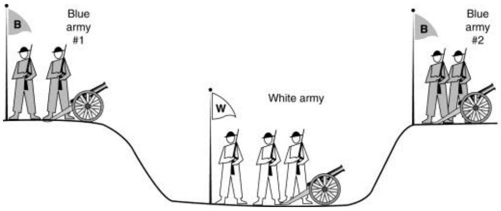
- How to release connections
  - We'll see how TCP does it



Connection Release

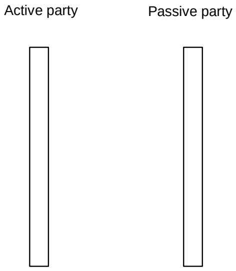
- Orderly release by both parties when done
  - Delivers all pending data and "hangs up"
  - Cleans up state in sender and receiver
- Key problem is to provide reliability while releasing
  - TCP uses a "symmetric" close in which both sides shutdown independently

Two-Army Problem



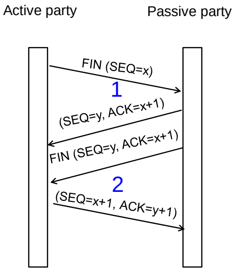
TCP Connection Release

- Two steps:
  - Active sends FIN(x), passive ACKs
  - Passive sends FIN(y), active ACKs
  - FINs are retransmitted if lost
- Each FIN/ACK closes one direction of data transfer

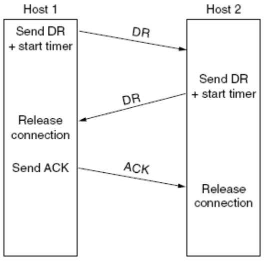


TCP Connection Release (2)

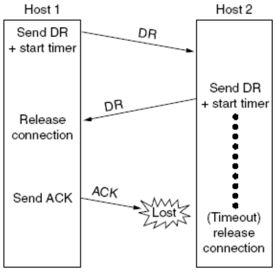
- Two steps:
  - Active sends FIN(x), ACKs
  - Passive sends FIN(y), ACKs
  - FINs are retransmitted if lost
- Each FIN/ACK closes one direction of data transfer



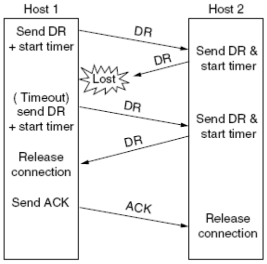
TCP Connection Release: Case 1



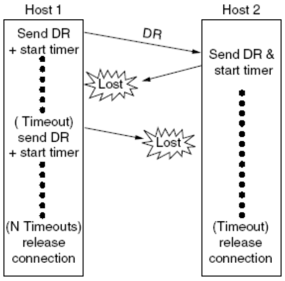
TCP Connection Release: Case 2



TCP Connection Release: Case 3

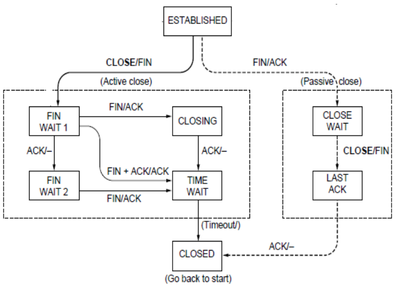


TCP Connection Release: Case 4


Host 1
Host 2
Send DR + start timer
DR
Send DR & start timer
Lost
(Timeout) send DR + start timer
Lost
(N Timeouts) release connection
(Timeout) release connection

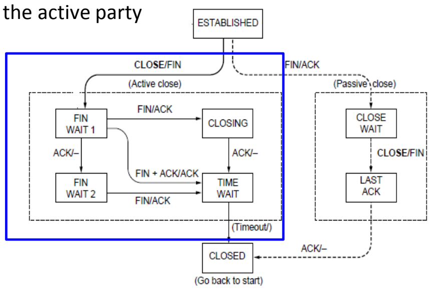
TCP Connection State Machine

Both parties run instances of this state machine



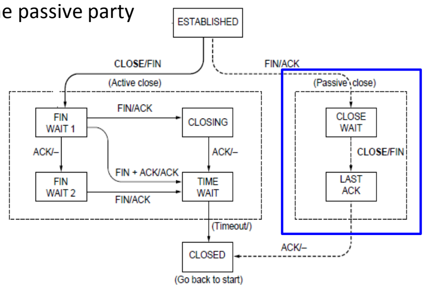
TCP Release

- Follow the active party



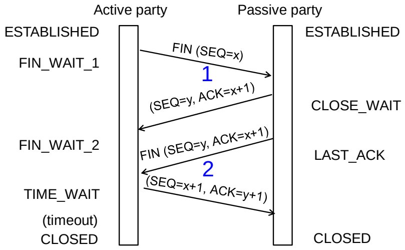
TCP Release (2)

- Follow the passive party



TCP Release (3)

- Again, with states …



TIME_WAIT State

- We wait a long time (two times the maximum segment lifetime of 60 seconds) after sending all segments and before completing the close

- Why?
  - ACK might have been lost, in which case FIN will be resent for an orderly close
  - Could otherwise interfere with a subsequent connection