CS2105 Cheatsheet 18/19 S1 Midterms

Application Layer

Processes

- Applications send `messages` to each other using `sockets`
- Application processes can only control:
  - transport protocol used
  - minor transport-layer parameters
- To send a message to another application process we need:
  - IP Address of the host
  - Destination port number

In general

- If there is Reliable Data Transfer (TCP) ⇒ No loss
- If there is high throughput ⇒ Large amounts of data can be transferred at a time
- If there is timing/latency guarantee ⇒ interactive applications feel realistic
- Protocol can specify security e.g encryption, checksum verification, end-point authentication

TCP

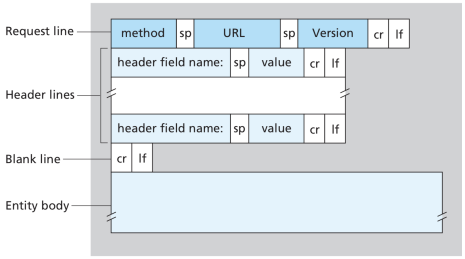
- A TCP Handshake must be formed before two-way connection between client and host
- Reliable Data Transfer: data is received in proper order without erroneous, duplicate or missing byes
- Has a flow-control and congestion-control mechanism

UDP

- Lightweight and connectionless
- Unreliable data transfer service: no guarantee of reaching in the correct order, or even reaching in the first place
- No congestion-control mechanism

HTTP

- Stateless protocol – cookies hold state (e.g preserve shopping cart)
- Common statuses 200 OK, 301 Moved Permanently, 400 Bad Request, 403 Forbidden,404 Not Found, 500 Internal Server Error
- A Web Cache keeps copies of recently requested objects in this storage, and makes TCP handshakes with server to get the object (if it's not cached)
- Non-persistence (1.0): One handshake for establishing connection, then `one file` is requested, followed by another handshake to close connection
- Persistence (1.1): One handshake for establishing connection, then files are requested sequentially. Connection is left open by server
- Pipelining (1.1): New requests made before previous ones are resolved, but order of objects is preserved (by TCP)
- Multiplexing (2): Responses can come back in any order, or even partially



DNS

- DNS (Domain Name Server) holds Resource Records (RR)
  - (Name, Value, Type, TTL)
  - `Type=A` ⇒ Name: hostname, Value: IP
  - `Type=NS` ⇒ Name: domain, Value: IP of authoritative DNS
  - `Type=CNAME` ⇒ Name: alias hostname, Value: canonical hostname
  - `Type=MX` ⇒ Name: alias hostname, Value: canonical name of mail server
- Root servers direct queries to TLD servers, only 13 root servers in the world
- TLD (Top-level Domain) responsible for `.com, .org, .net, .sg...`
- Authoritative server keeps hostname-IP mappings of organization's named hosts
- Local DNS acts as a proxy, caches previously retrieved mappings to cache. This allows for faster lookup. Owned by ISPs
- All DNS servers can implement caching, so root servers are often not called upon
- DNS uses port 53

Delay

- There are four types of delays:
  - N (Number of Bits to Transmit)
  - Transmission Delay (d_{trans}) : N / Transmission Rate
  - Propogation Delay (d_{prop}): distance / Propogation Speed
  - Queing Delay (d_{queue}): Time spent in Queue at Router R
  - Processing Delay ($d_{process}$): Time spent to process by Router R
- RTT (Round-Trip-Time): $d_{prop} + d_{queue} + d_{process}$
- Throughput (End to End delay): $N/(d_{trans}+d_{prop}+d_{queue}+d_{process})$
- Link Utilization: $d_{trans}/(d_{trans} + RTT)$

Transport Layer

In general

- Provides logical communication between `processes` by sending `segments`
- Note that the term `datagram` is used for Network layer, but is commonly used to refer to UDP packets as well
- The end goal of TCP/UDP is to extend host-to-host delivery to process-to-process delivery (transport-layer multiplexing and demultiplexing)

RDP

- Checksum implemented if channel can flip bits (2.0)
- ACK/NAK used to recover from bit errors (2.0)
- ACK/NAK used to recover from bit errors (2.0)
- Sequence number used to account for ACK/NAK corruption (2.1)
- ACK0/1 used instead of NAK (2.2) for simplicity
- Timeout added to account for packet loss (but not reordered) (3.0)

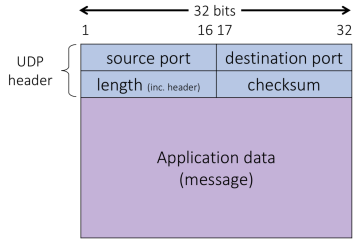
Pipelining

- Go-Back-N:
  - Receiver sends cumulative ACK ⇒ all packets $\leq n$ have been received ⇒ ensures order
  - Receiver discards any packets not in order
  - Sender attaches k-bits sequence to packet header ⇒ can represent at least $[0, n]$
  - Sliding window of size n is kept
  - Keep timer for oldest unACKed packet
  - Resend all n packets in window on timeout
- Selective Repeat:
  - Receives acknowledges packets individually
  - Receives buffers out-of-order packets for eventual in-order delivery (to upper layer
  - Sender maintains timer for each unACKed packet ⇒ only retransmit that packet on timeout
  - Sender attaches k-bits sequence to packet header ⇒ can represent at least $[0, n]$
  - Sliding window of size n is kept
  - Keep timer for oldest unACKed packet
  - Resend all n packets in window on timeout

UDP

- User creates a socket to send segment, and specifies IP address and port # in the segment
- Receiver checks destination port in segment, and redirects to the socket with that port number
- Benefits:
  - No connection set-up
  - No state to remember
  - Small header size ⇒ less overhead
  - No congestion control (as compared to TCP)
  - Checksum to verify integrity
- Checksum value included in UDP checksum field. Example checksum:
  - Split into 16bit integers
  - Add integers, use wrap-around carry
  - Compute 1's complement (flip all bits)

UDP segment structure

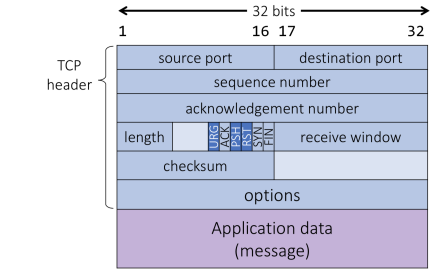


TCP

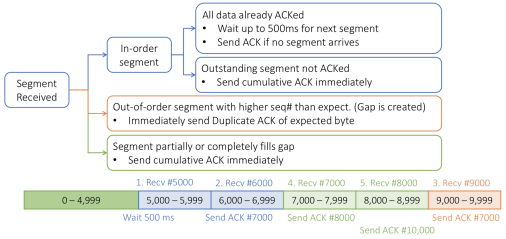
- Connection-oriented – Handshake is required before transmission
- Reliable, in-order byte stream
- Offset field allows for additional headers (it stores total number of TCP headers)
- Sequence Number is byte number of first byte of data in segment
- ACK Number is the sequence number of the next byte of data expected
- ACK Bit represents if the ACK number should be read
- SYN Bit initiates a connection
- FIN Bit is to denote closing of the connection (Final)
- RST Bit is to denote abortion of a connection due to error

- PSH Bit is to indicate that the data should be pushed to application level immediately, without waiting for additional data
- URG Bit is to indicate to a receiving station that the data is urgent and should be prioritised
- Receive window tells sender how much data can be sent
- Timeout:
  - Too long ⇒ slow reaction to loss
  - Too short ⇒ Premature timeout and unnecessary retransmissions
  - Estimate RTT using Exponential Weighted Moving Average: $RTT_E = (1-a) * RTT_E + a * RTT_s$, where a is usually $\frac{1}{8}$
  - Calculate deviation of RTT using $RTT_{dev} = (1-b) * RTT_{dev} + b * RTT_s - RTT_E$, where b is usually $\frac{1}{4}$
  - Retransmission Timeout is $RTO = RTT_E + 4 * RTT_{dev}$, the deviation is used as a "safety margin"
  - Note that the RTO is **doubled** after each timeout
  - TCP Fast transmission: If 3 Duplicate ACK

TCP segment structure



TCP Receiver Events



Network Layer

In general

- Provides logical communication between `hosts`, through `routers`
- Has two responsibilities: `forwarding` (through a single output link) and `routing` (determining the route that packets should follow)

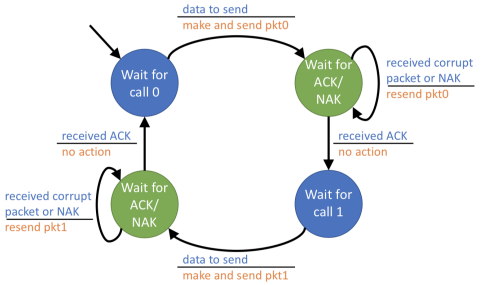
IP

- IP Address is 32bit
- IP Address is associated with an interface e.g 802.11 wifi
- However, the device that we call "router", actually forwards packets between networks. This device has multiple interfaces e.g LAN ports are all one interface
- Dynamic Host Configuration Protocol:
  - Host broadcast a *Discover* message
  - DHCP server (listening on port 67) reponds with an offer. First transaction ends here.
  - Host requests for the given IP with a new Transaction (new transaction ID used)
  - DHCP ACKnowledges request and assigns the IP
- Some IP Addresses are reserved
- Hierarchical addressing scheme:
  - IP Addresses grouped into subnets, each subnet having a certain prefix for all IP Addresses
  - Internet decides which ISP to send the packet to using the Longest Prefix of the address
  - Subnet mask is a common prefix used between hosts in the same subnet, where they can communicate between each other without a router
- Network Address Translation:
  - Entire subnet under the router is identified by a single IP Address
  - NAT translation table used to map $(IP_{subnet}, port)$ to $(IP_{global}, port)$
  - Hosts' identities are effectively "firewalled" from the outside world
  - External parties cannot identify hosts using IP, only using port (which is supposed to be used on the Application-level)

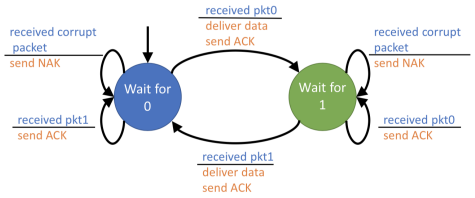
Special IP Addresses

127.0.0.1/8	Loopback address. Typically using 127.0.0.1/32
10.0.0.0/8 172.16.0.0/12 192.168. 0.0/16	Private addresses. Local communication in a private network.
255.255.255.255/32	Broadcast address. All hosts on the same subnet will receive the datagram
0.0.0.0/8	Non-routable meta-address for special use

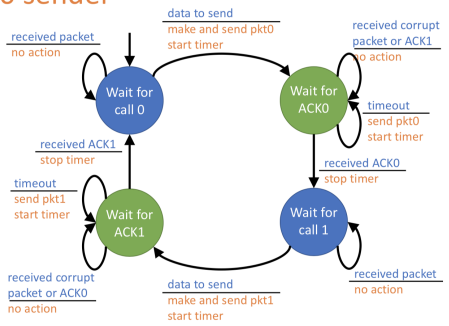
rdt 2.1 sender



rdt 2.1 receiver



rdt 3.0 sender



Java Socket Programming

TCPEchoClient

```
import java.io.*;
import java.net.*;
import java.util.*;

class TCPEchoClient {
    public static void main(String[] args) throws IOException {
        // create a client socket and connect to the server
        Socket clientSocket = new Socket("localhost", 5678);

        // read user input from keyboard
        Scanner scanner = new Scanner(System.in);
        String fromKeyboard = scanner.nextLine();
        // create output stream to server
        PrintWriter toServer = new PrintWriter(clientSocket.getOutputStream(),
                                               true);

        // write user input to the socket
        toServer.println(fromKeyboard);
        // create input stream from server
        Scanner sc = new Scanner(clientSocket.getInputStream());
        // read server reply from the socket
        String fromServer = sc.nextLine();
        // show on screen
        System.out.println("Echo from server: " + fromServer);
        clientSocket.close();
    }
}
```

TCPEchoServer

```
import java.io.*;
import java.net.*;
import java.util.*;

class TCPEchoServer {
    public static void main(String[] args) throws IOException {
        int port = 5678; // server listens to this example port
        ServerSocket welcomeSocket = new ServerSocket(port);

        while (true) { //server is always alive
            // accepts a new connection
            Socket connectionSocket = welcomeSocket.accept();
            Scanner scanner = new Scanner(connectionSocket.getInputStream());

            // read data from the connection socket
            String fromClient = scanner.nextLine();
            PrintWriter toClient = new PrintWriter(
                                   connectionSocket.getOutputStream(), true);

            // write data to the connection socket
            toClient.println(fromClient);
            connectionSocket.close();
        }
    }
}
```