Comp3331

Wk 1

1.1

- **internet**

    - network of networks: Interconnected ISPs (Internet Servers Provider).

- **protocols**

    - define <u>format</u>, <u>order of msgs</u> sent and received among network entities, and <u>actions taken</u> of msg tranmission, receipt.

- **Internet standards**

    - RFC: Request for comments
    - IETF: Internet Engineering Task Force.

1.2

- **Access net**

    - DSL (digital subscriber line)
        - use existing telephone line
        - voice, data transmitted at **different frequencies**

    - **Cable network**

        - HFC (hybrid fiber coax): very fase
        - homes share access network (shred cable)

    - **FTTH (Fiber to the home)**

    - **Ethernet**

        - end systems typicaly connect into Ethernet switch

    - **Wireless access networks**

        - wireless LANs: within building (100ft)
        - wide-area wireless access: privided by telco (cellular) operator, 10’s km

1.3

- **Circuit Switching**

  - No sharing, circuit-like (guaranteed) performance
  - FDM (Frequency-division multiplexing) - users continously shared bandwidth
  - TDM - sequencely occupy the whole bandwidth

- **Packet Switching**

  - data is sent as packets (header + payload) independently
  - header
    - Internet Address
    - Age (TLL)
    - Checksum

- **Store and forward**

  - a packet is entirely received before forwards/processes

- **Statical multiplexing**

  - n packets comming at the same time, adjust so everyone using entire bandwidth capacity
  - need a buffer queue
  - drop packets when queue overload
  - cannot use large buffer or delay will be long

1.4

- **Packet delay**

  - Processing delay
  - Queueing delay (traffic intensity) - pkt arrival rate * pkt len / bandwidth
  - Transmission delay - pkt len/bandwidth
  - Propagation delay

- **traceroute program**

  - provides delay measurement

- **end-to-end**

  - measured the end experience to the user

Wk 2

1.5

- **Internet layer (top to bottom)**

  - Application: FTP, SMTP, HTTP, Skype…
  - Transport: TCP, UDP
  - Network: IP
  - Link: Ethernet, WiFi, PPP…

    - Physical: copper, fibre, radio…

- **Cons of layer**

    - may duplicate lower level functionality (e.g. error recovery to retransmit lost data)
    - information hiding (pkt loss due to corruption vs. congestion)
    - header become very large
    - layer violations when the gains too greate to resist (e.g. TPC-over-wireless)
    - layer violations when net work doesn't trust ends(e.g. firewalls)

- **routers don't have** transport and application layer

2.1

- **IPC (Interprocess Communication)**

    - Two process communicate in shared memory

- **Socket**

    - used in message passing across machines
    - implemented between application (process) and transport layer

- **Addressing**

    - host: unique IP address
    - process: prot nubmers

- **Server**

    - long-lived (always-on host)
    - received request
    - permanent IP address
    - static port conventions (http:80, email: 25, ssh:22)
    - data centre for scaling
    - may communicate with other server to respond

- **Client**

    - short-lived
    - send request
    - may have dynamic IP addresses
    - do not communicate directly with each other

- **Peer-to-Peer**

    - Pros
        - Self scalability - new peeres bring new service capacity, as well as new service demands
        - Spped
        - Reliability
        - Geographic distribution

- Cons
  - State uncertainty: no shared memory or clock
  - Action uncertanty: mutually conflicting decisions
  - algorithms are complex

- **TCP**

  - reliable transport
  - flow control
  - congestion control
  - connection-oriented
  - don’t have:
    - timing
    - minimum throughput guarantee
    - security

- **UDP**

  - unreliable data transfer
  - don’t have:
    - reliability
    - flow control
    - congestion control
    - timing
    - throughput guarantee
    - security
    - connection set up

2.2

- **Web page**

  - consists of base HTML-file which includs several referenced objects
  - consists of objects (e.g. HTML file, JPEG image, Java applet, audio…)
  - is addressable by a unique URL

- **URL (Uniform Resource Locator)**

  - `protocol://host-name[:port]/directory-path/resource`

- **HTTP**

  - uses TCP
  - stateless, if crashes has to start from beginning
  - HTTP messages
    - ASCII (human-readable)
    - two types: request and response

  - to keep state, use coocies

- **HTTP transmission**

- non-persistent
    - have different socket for each request
    - response time = N * 2RTT (connection + files)

- persistent without pipelining
    - response time = RTT (connection) + RTT (index) + N*RTT (files)

- persistent with pipelining
    - response time = RTT (connection) + RTT (index) + RTT (files)

- **Web cashes (proxy server)**

- **HTTPS**

    - HTTP over a connection encrypted by TLS (Transport Layer Security)

Wk3

2.3

- **Electornic mail**

    - main components
        - user agents
        - main server
        - SMTP (Simple Mail Transfer Protocol)

    - message stored on server
    - use TCP, port 25
    - mail server <–> mail server - must be SMTP
    - client agent <–> mail server - can be different protocol
        - POP (Post Office Protocol): server doesn't store msg
        - IMAP (Internet Mail Access Protocol): server saves msg, can access mail from different machines

- **SMTP**

    - persistent connection
    - header and body in 7-bit ASCII
    - uses CRLF.CRLF to determine end of message
    - comparison with HTTP
        - HTTP
            - pull from server
            - encapsulated objects in its own response msg

        - SMTP
            - push to client
            - multiple objects sent in multipart msg

2.4

- **DNS (Domain name system)**
  - distributed database
  - application-layer protocol
- **TLD (Top Level Domain)**
  - root: `.`
  - TLD
    - `.edu`
    - `.com`
    - `.gov`
    - `.mil`
    - `.org`
    - `.net`
    - `.uk`
  - deeper (Authoritative DNS server):
    - `.berkeley.edu`
    - `.ucla.edu`
  - deeper (Authoritative DNS server):
    - `eecs.berkely.edu`
    - `sims.berkely.edu`
  - deeper (Authoritative DNS server):
    - `instr.eecs.berkely.edu`
- **Zone**
  - a zone correspond to an administrative authority that is responsible for that portion of the hierarchy
- **Local DNS name server**
  - has local cashe of recent name-to-address translation pair
  - record has TTL (time to live), if expired will be deleted
  - name resolution
    - iterative
      - ask each domain server
      - evenly distribute load
      - main responsibility on local DNS server
    - recursive
      - root server will freak out
      - lower the performance of root server
- **RRs (DNS resource records)**
  - format: `(name, value, type, ttl)`
  - type A
    - name: hostname

        - value: IP address

    - type NS
        - name: domain name
        - value: hostname of authoritative name server for this domain

    - type CNAME
        - name: alias name
        - value: canonical (real) name

    - type MX
        - for mail exchange
        - value: name of mailserver associated with name

    - type PTR
        - reverse type A

2.6

- **Streaming multimedia: DASH (Danymic, Adaptive Streaming over HTTP)**

    - Server
        - divide video into multiple chunks
        - chunks encoded in different rate
        - <u>manifest file</u>: provides URLs for different chunks

    - Client (intelligence)
        - periodically measures server-to-client bandwidth (and choose the fastest/closest one)
        - requests 1 chunck 1 time
        - differnet coding rates at different points

- **CDN (Content Distribution Networks)**

    - Goal: bring content closer to user
    - combination of (pull) caching and (push) replication
    - store multiple copies of video at multiple geographically distributed sites
    - Netflix using own CDN

Wk 4

2.5

- **P2P**

    - not always-on server
    - comparison
        - client-server
            - server: subsequently upload N file copies at U_s bits/sec
            - client: download each copy at d_{min} bits/sec

        - time to distribute *NF*: `max{NF/Us, F/Dmin}`
        - time increase linearly

    - P2P
        - server: upload at least one copy *F* at U_s bits/sec
        - each client download one *F* at d_{min} bits/sec
        - as aggregate must upload *NF* files at $U_s+sum(U_i)$ bits/sec
        - time to distribute *NF*: `max{F/Us, F/dmin, NF/(Us+sum(Ui)}`

  - "rarest first"
  - peer re-evaluate top 4 every 10 secs
  - every 30 secs optimistic unchoke 1 random neighbour

- **DHT (Distributing Hash Table)**

  - A distributed P2P database that map strings to integers[0, 2^n - 1]
  - (key,value) pairs
  - each peer knows 1 predecessor and 2 successor

3.2

- **UDP is connectionless (no handshaking), it only identified by two tuples:**

  - dest port #
  - dest IP addr
  - server maintain single socket for all incoming pkts

- **TCP is identified by 4 tuples:**

  - source IP address
  - source port number
  - dest IP address
  - dest port number
  - server creates new socket for each TCP connection

- **TCP Socket**

  - Needs more physical socket but with same port #

3.3

- **UDP**

  - header: only 8 bytes (TCP 20 bytes)
  - the "length" field is the length of UDP segment including header (bytes)
  - checksum: one’s complement of sum
  - application: latency sensitive/time critical (e.g. DNS, routing updates, voice/video chat, gaming)

3.4

- **ARQ (Automatic Repeat Request)**
    - Stop-and-Go (Stop-and-wait)
    - Pipelining
        - Go-back-N
        - Selective Repeat
- **rdt2.0: channel with bit error**
    - recover from error:
        - ACK (acknowledgements)
        - NAKs (negative acknowledgements)
    - fails if ACK and NAKs corrupt
- **rdt2.1**
    - server
        - add seq #: only need two seq # (1 and 0, repeatly)
    - receiver
        - discard duplicate
        - can't know if ACK/NAK successfully sent
- **rdt2.2**
    - NAK-free
    - ACK # indicates which pkt successfully received
- **rdt3.0 (Stop-and-Go (Stop-and-wait))**
    - has timer: if timeout and didn't receive ACK then resend
    - still discard duplicate

Wk 5

- **Stop-and-Go (Stop-and-wait)**
    - Utilisation factor:
      `Usender = ((len of pkt) / (R bits/sec)) / (RTT + (len of pkt) / (R bits/sec))`
- **Pipelining**
    - Utilisation factor:
      `Usender = ((pipes# * len of pkt) / (R bits/sec)) / (RTT + (len of pkt) / (R bits/sec))`
- **Go-Back-N**
    - seq # store in binary bit, $[0, 2^m - 1]$, where m=size of bit field
    - Sender
        - window size: $< 2^m$ (max = seq# -1)
        - slides window forward upto ACK

        - buff out-of-order ACK
    - Receiver
        - window size: 1
        - doesn’t buff out-of-order pkt
        - keep sending last ACK if receive out-of-order pkt
    - discard duplicate
    - not efficient if lost
- **Selective Repeat**
    - Sender
        - window size: ≤ 2^{m-1} (max = size of bit field/2)
        - slides window forward for in-order-ACK
        - only resends pkt for which ACK not received
    - Receiver
        - window size: same as sender
        - slides window forward for in-order-pkt
        - buffer out-of-order pkt

3.5

- **TCP header: 20 bytes**
- **reliable transport solution**
    - checksum (for error detection)
    - timer (for loss detection)
    - ACK (cumualtive/selective)
    - Seq# (duplicates, windows)
    - Sliding Window (for efficiency)
- **TCP segment**
    - sent when full (excluding the header, only data size)
    - not full but dictated by application
        - minimum: size = 0 (ACK)
        - Telnet: size = 1 byte
    - size
        - structure: `{ IP Data [TCP Data(segment) | TCP hdr] | IP hdr }`
        - MSS (Maximum Segment Size) = whole IP pkt size - IP hdr size - TCP hdr size
    - seq#
        - ISN (initial sequence number) + size of segment
        - ISN starts from random #
    - ACK#
        - = next expected byte ("what byte is next")
        - = seq# + size of segment

- **TCP RTT**
    - `TimeoutInterval = EstimatedRTT + 4 * DevRTT (safety margin)`
- **TCP retransmission**
    - avoid repeatly sending same pkt, only send latest ACK# as seq# pkt
    - e.g. if ACK100 lost, but ACK120 sent, sender get ACK120, still send pkt120 as it knows pkt100 is received successfully
- **TCP ACK retransmit**
    - receiver: wait upto 500ms, and send generated ACK for all pkts.
    - avoid send ACK too frequently
- **TCP fast retransmit**
    - sender: resend pkt after receiving last ACK for 4 times (1 initially, 3 repeated) even if no timeout
- **TCP flow control**
    - receiver controls sender so won't overflow
    - receiver "advertise" free buffer space (by `rwnd` value, default 4096 bytes) in TCP header
- **Connection management**
    - Establish connection
        - 3-way handshake
            - client: send SYNbit=1, initial seq#
            - server: send ACK# = client's seq# + 1(ACKbit), another seq#
            - client: send ACK = server's seq# + 1(ACKbit)
            - TCP connection Established
        - SYN loss
            - wait for 3sec by default (some are 6sec)
            - re-establish
    - Close connection
        - client: send FINbit, seq#
        - server: send ACK# = client's seq# + 1
        - server: send FINbit, another seq#
        - client: send ACK# = server's seq# + 1
        - TCP connection closed
        - `TIMED_WAIT` (2*max segment lifetime): Can retransmit ACK if last ACK or FIN is lost
    - Normal Termination:
        - client: send FINbit, seq#
        - server: send ACK# and FIN together
        - client: send ACK of FIN