CS2105 Finals Cheatsheet v1.0 (2020-09-28)
by Julius Putra Tanu Setiaji, page 1 of 1

1 Computer Networks and the Internet

1.1 What is the Internet

1.2 Network Edge

1.3 Network Core

1.3.1 Circuit switching

Dedicated circuit per call
- call setup required
- circuit-like (guaranteed) performance
- circuit segment idle if not used (no sharing)

1.3.2 Packet Switching

Data sent through the net in discrete chunks
- **Store-and-forward**: entire packet must arrive at a router before it can be transmitted to the next link.
- **Addressing**: each packet needs to carry source and destination information
- Users share network resources
- Resources are used on demand
- Excessive congestion is possible

1.4 Delay, Loss and Throughput in Networks

End-to-end packet delay consisting of 4 sources

1.4.1 4 Sources of Packet Delay

- Nodal Processing (d_{proc}): check bit errors, determine output link, typically < msec
- Queueing (d_{queue}): waiting in queue for transmission, depends on congestion level of router
- Transmission ($d_{trans} = \frac{L}{R}$): L = packet length (bits), R = link bandwidth (bps)
- Propagation ($d_{prop} = \frac{d}{s}$): d = length of physical link, s = propagation speed in medium ($2 \times 10^8 m/sec$)

1.4.2 Throughput

- How many bits can be transmitted per unit time
- Measured for end-to-end communication. Compare with link capacity (bandwidth) only for specific link

1.4.3 Units

- 1 byte = 8 bits
- (-) Prefixes: milli, micro, nano, pico, femto, atto, zepto, yocto
- (+) Prefixes: kilo, mega, giga, tera, peta, exa, zetta, yotta

1.5 Protocol Layers and Service Models

1.5.1 5 Layers

- **Application**: supporting network applications, e.g. FTP, SMTP, HTTP
- **Transport**: process-to-process data transfer, e.g. TCP, UDP
- **Network**: routing of datagrams from source to destination, e.g. IP, routing protocols
- **Link**: Data transfer between neighbouring network elements, e.g. Ethernet, 802.11, PPP
- **Physical**: "on the wire"

1.5.2 ISO/OSI Reference Model

Theoretical only, 2 additional layers between **Application** and **Transport**: **Presentation** (allow applications to interpret meaning of data, e.g. encryption, compression, machine-specific convention) and **Session** (synchronisation, checkpointing, recovery of data exchange)

2 Application Layer

2.1 Principles of Network Applications

2.1.1 Client-Server

- **Server**: waits for incoming requests, provides requested service to client, data centers for scaling
- **Client**: initiates contact with server, typically requests service from server, For web, client is usually implemented in browser

2.1.2 Peer-to-Peer (P2P)

- No always-on server
- Arbitrary end systems directly communicate.
- Peers request service from other peers, provide service in return to other peers
- **Self-scalability**: new peers bring new service capacity, as well as new service demands
- Peers are intermittently connected and change IP addresses (complex management)

2.1.3 Requirements of apps

- **Data integrity**: 100% reliable vs some data loss
- **Timing**: some apps require low delay to be "effective"
- **Throughput**
- **Security**

2.1.4 Definition of App-layer Protocols

- **Types of Messages exchanged,** e.g. request, response
- **Message syntax**, e.g. message fields and how they are delineated
- **Message semantics**: meaning of information in fields
- **Rules** for when and how application send and respond to messages

2.1.5 Transport-Layer Protocols

TCP	UDP
Reliable data transfer	**Unreliable** data transfer
Flow control: sender won't overwhelm receiver	**No flow control**
Congestion control: throttle sender when network is overloaded	**No congestion control**
Does not provide: timing, minimum throughput guarantee, security	**Does not provide**: timing, throughput guarantee, security

2.2 Web and HTTP

2.2.1 HTTP

- HyperText Transfer Protocol
- Client/server model
- RFC 1945 (HTTP 1.0), RFC 2616 (HTTP 1.1)
- Over TCP

2.2.2 Persistent HTTP

- Multiple objects can be sent over single TCP connection
- **Persistent with pipelining**: client may send requests as soon as it encounters a referenced object – as little as 1RTT for all referenced objects.

2.2.3 Non-Persistent HTTP

- At most 1 object sent over a TCP connection
- Requires 2 RTTs per object
- Response time = $2 \times RTT +$ file transmission time

Refer to slides for the rest of HTTP

2.3 DNS

- **Distributed, Hierarchical Database**
- **Root Server**: answers requests for records in the root zone by returning a list of the authoritative name servers for the appropriate TLD
- **DNS Caching**: based on TTL
- Runs over **UDP**

2.3.1 Resource Records (RR)

Stores mapping between hostnames and IP addresses, 4-tuple `(name, value, type, ttl)`
- type = A, `name` is hostname, `value` is IP address
- type = NS, `name` is domain, `value` is hostname of authoritative name server for the domain
- type = CNAME, `name` is alias for some canonical name, `value` is the canonical name
- type = MX, `value` is the name of mail server assoc with `name`

2.3.2 DNS Name Resolution

- **Iterative query**: Local DNS server makes DNS requests one by one in the hierarchy
- **Recursive query** (rarely used): each server in the hierarchy asks one server higher in the hierarchy

2.4 Socket Programming

- **IP address** is used to identify a host device
- **Process**: program running within a host, identified by `(IP address :: uint32, port number :: uint16)`
- **Socket**: the software interface between app processes and transport layer protocols

2.4.1 UDP

- No "connection" between client and server.
- Sender explicitly attaches destination IP address and port number to each packet
- Receiver extracts sender IP address and port number from the received packet

2.4.2 TCP

- When client creates socket, client TCP establishes a copnnection to server TCP.
- When contacted by client, server TCP creates a new socket for server process to communicate with that client
- Allows server to talk with multiple clients individually.
- Communicates as if there is a pipe between 2 processes, sending process doesn't need to attach a destination IP address and port number in each sending attempt.

3 Transport Layer

3.1 Transport-layer Services

- Sender: Breaks app messages into **segments**, passes them to network layer
- Receiver: Reassembles segments into message, passes it to app layer
- Packet switches in between: only check destination IP address to decide routing
- Each IP datagram contains source and dest IP addresses

3.2 Connectionless Transport: UDP

- UDP addes very little on top of IP
  - Multiplexing at sender
  - Demultiplexing at receiver
  - Checksum
- UDP transmission is unreliable, often used by (loss tolerant & rate sensitive apps)

3.2.1 Connectionless De-multiplexing

When UDP receiver receives a UDP segment:
- Check destination port number in segment, and direct that segment to the socket with that port number.

3.2.2 UDP Header

16 bits each for: source port number, dest port number, length, checksum

3.2.3 UDP Checksum

- Treat segment as sequence of 16-bit integers
- Apply binary addition, wraparound carry added to the result
- Compute 1's complement to get the checksum

3.3 Principles of Reliable Data Transport

Refer to slides for rdt example protocols

3.4 Connection-oriented Transport: TCP

- **Point-to-point**: 1 sender, 1 receiver
- **Connection-oriented**: handshake before sending app data
- **Full duplex service**: bi-directional data flow in the same connection
- **Reliable, in-order byte stream**: sequence numbers to label bytes

3.4.1 Connection-oriented de-mux

A TCP connection/socket is identified by 4-tuple `(srcIPAddr, srcPort, destIPAddr, destPort)`

3.4.2 TCP: buffers and Segments

- two buffers, send and receive, are created after handshaking at both sides
- **Max Segment Size (MSS)**: typically 1460 bytes, max app-layer data one TCP segment can carry.

3.4.3 TCP Header

```
|1             16           32|
| sourcePort#      destPort#  |
|       sequence number       |
|    acknowledgement number   |
|                             |
|  checksum    |              |
```

- **Sequence Number**: byte number of the first byte of data in a segment
- **ACK number**: sequence number of the next byte of data expected by the receiver
- **Cumulative ACK**: TCP ACKs up to the first missing byte in the tream

3.4.4 TCP ACK Generation, Timeout Value, Fast Retransmission

Refer to Lecture 4,5 Slide 66