CS2105 Comp. Networks Notes

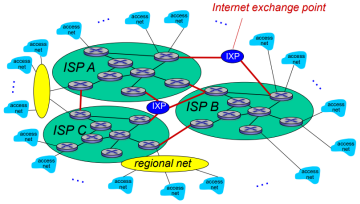
AY23/24 Sem 1. github.com/gerteck

1. Computer Networks Introduction

- Fundamental concepts and principles behind computer networking, with internet as case study.
- Connected by commmunication links and packet switches.

Internet

- The Internet is a network of connected computing devices (e.g. PC, server, laptop, smartphone).
- Such devices are known as hosts or end systems. Hosts run network applications (e.g. Tele, browser, Zoom)
- Packet switching network, users' packets share network resources that are used on demand. Excessive congestion is possible.
- **Network of networks** Hosts connect to Internet via access ISPs (Internet Service Providers), which themselves are interconnected.



Network Edge (Access Network)

- The access network is the network that physically connects an end system to the first router on a path from that end system to any distant end system.
- E.g. Residential access networks, mobile access networks.

Network Core

- A mesh of interconnected routers.
- Data is transmitted through network through:
- **Circuit switching:** dedicated circuit per call
- **Packet switching:** data sent through net in discrete "chunks"

Circuit Switching

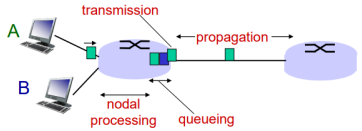
- End-end resources are allocated to and reserved for "call" between source & dest:
- call setup required, circuit-like (guaranteed) performance
- circuit segment idle **silent periods** if not used by call (no sharing)
- commonly used in traditional telephone network

Packet Switching

- Host sending function: breaks application message into smaller chunks, known as packets, of length L bits. Then, transmits packets onto the link at transmission rate R.
- Packets are passed from one router to the next, across links on path from source to destination.
- link transmission rate is aka link capacity or **link bandwidth**
- **Packet transmission delay** (time taken to transmit L-bit packet into Link) = $\frac{L}{R}$
- **Store-and-forward**: entire packet must arrive at a router before it can be transmitted on the next link.

Delay, Loss, Throughput

Four sources of packet delay, suffered at each node, from source to destination. (**Total nodal delay**)



- **Nodal Processing Delay**: Time required to check for bit errors and determine output link. Typically $<$ msec.
- **Queueing Delay**: Time waiting in queue for transmission, depends on earlier arrived packets. Typically $<$ msec. Additionally, possibility of packet loss if router queue full (buffer has reached finite capacity), and drops packet.
- **Transmission Delay**: Time to push (last bit) of packet on wire. $d_{trans} = \frac{L}{R}$.
- **Propagation Delay**: Time to propagate to next router. $d_{prop} = \frac{d}{s}$, where d is length of physical link, s propagation speed.
- End to end packet delay is total time taken for packet to travel from source to destination, consisting of the 4 delays.
- **Throughput**: How many bits can be transmitted per unit time, usually measured for end-to-end communication (as opposed to bandwidth for specific link).

Internet Protocol Stack

Protcols logically organised into 5 "layers" according to purpose. (Additionally presentation and session layers not included)

- **Application**: Where network applications and app-layer protocols reside. Packet here called message.
  Examples: HTTP, SMTP, FTP
- **Transport**: Transports app-layer messages between application endpoints. Packet here called segment.
  Examples: TCP, UDP
- **Network**: Moves packets (datagrams) from one host to another. Includes IP protocol and other routing protocols.
- **Link**: Moves packet from one node to another. Packet here called frame.
  Example: Ethernet, WiFi
- **Physical**: Moves individual bits within link-layer frame from one node to another. Link and transmission medium dependent.

2. Application Layer

Principles of Network Applications

- **Client-Server**:
  - Server waits for incoming requests and provides required services to client. Easy scalability.
  - Client initiates contact with server and requests service.
- **Peer-To-Peer (P2P)**:
  - No dedicated server, instead it relies on direct communication between pairs of intermittently connected hosts called peers.
  - Self-scalability, each peer generates workload but adds service capacity by distributing files.

Process Communication

- **Socket**: A software interface that process uses to send messages and receive messages from network. Generally a combination of IP address and port number.
- **IP Address**: A 32-bit quantity, uniquely identifies host.
- **Port Number**: A 16-bit integer used to identify a receiving process running in a host.

Requirements of Transport Service

- **Reliable Data Transfer**: Data to sent correctly and completely vs. loss-tolerant.
- **Throughput**: Bandwidth-sensitive apps may need guaranteed throughput of r bits/sec.
- **Timing/Delay**: Real-time applications generally require low delays to be effective.
- **Security**: Encryption, data integrity, authentication.

Transport Layer Protocols

Two main protocols for the Internet.

- **Transmission Control Protocol (TCP)**
  - Reliable data transfer, Connection-oriented service: A handshake required.
  - Flow control, Congestion control: Throttle sender when network overloaded
  - Security: Can be enhanced at the app layer with Secure Sockets Layer
  - Does not provide: Timing and throughput guarantee
- **User Datagram Protocol (UDP)**
  - Unreliable data transfer
  - Connectionless: No handshake
  - No flow control, no congestion control
  - Does not provide: Timing and throughput guarantee, security.

Application-Layer Protocols

An application-layer protocol defines:

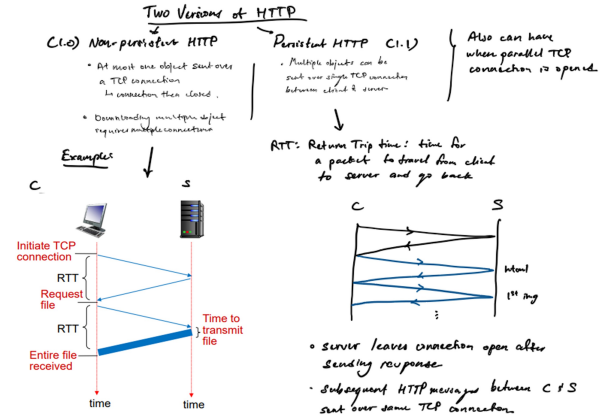
- Types of messages exchanged, e.g. request and response messages.
- Syntax of message types, e.g. fields and how they are delineated.
- Semantics of the fields, i.e. what the information means.
- Rules for when and how to send a message and respond to messages.

Web & HTTP

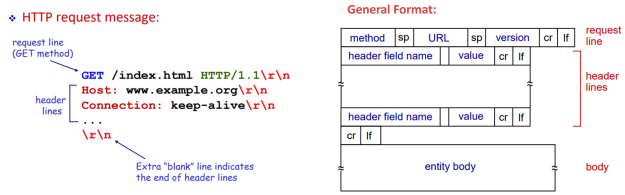
- **Webpage**: Consists of base HTML files and referenced objects. Addressable by URL.
- URL made up of hostname as well as path name. (E.g. http://www.comp.nus.edu.sg/ cs2105/img/doge.jpg)
- **HyperText Transfer Protocol** is Web's app-layer protocol.
- **Client-server model:** Client requests, receives and displays Web objects, server is Web server that sends objects in response.
- **Stateless**: server maintains no information about clients, and **Reliable**: Over TCP.
- **Three-way Handshake**: Client sends small TCP segment to ask for connection, server acknowledges and responds, client acknowledges and sends it back with request message.

HTTP versions

- **RTT**: Round trip time, time taken for packet to travel from server and back to client, does not include transmission delay.
- **Non-persistent HTTP**: Response time = 2∗ RTT+ file transmission time (per object)
- **Persistent HTTP**: Server leaves connection open after sending response, subsequent HTTP sent over same TCP connection. Also uses pipelining, send requests back to back.

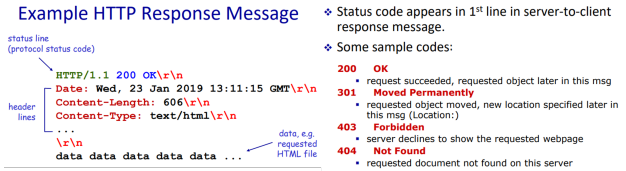


HTTP Request



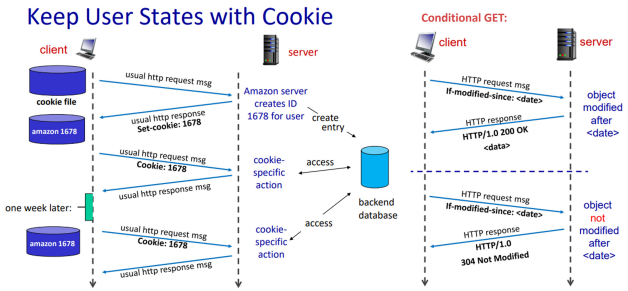
- **HTTP 1.0 Methods**: GET (gets object), POST (posts form data), HEAD (gets header without body).
- **HTTP 1.1 Methods**: GET, POST, HEAD, PUT (uploads file to path specified), DELETE

HTTP Response



Cookies

- HTTP is designed to be "stateless", server maintains no information about past client requests.
- Good to maintain states over multiple transactions, e.g. shopping carts
- **Cookie**: http messages carry "state":
  1) cookie header field of HTTP req/res messages
  2) cookie file kept on user host, managed by browser
  3) back-end database at Web site
- **Conditional GET**: In cache, specify date of cached copy in HTTP request. Server response contains no object if cached copy is up to date.



Domain Name System

DNS translates between hostname and IP addresses. Client must carry out a DNS query to determine the IP address corresponding to the server name.

- **DNS: Resource Records (RR)**
  - Mapping between host names and IP addresses (and others) are stored as resource records (RR).

RR format: `(name, value, type, ttl)`

type = A
- `name` is hostname
- `value` is IP address

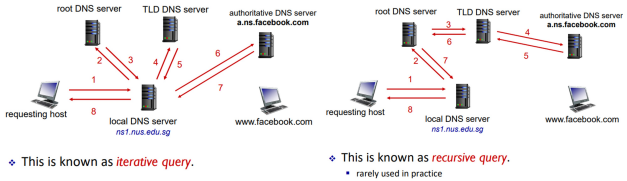
type = NS
- `name` is domain (e.g., `nus.edu.sg`)
- `value` is hostname of authoritative name server for this domain

type = CNAME
- `name` is alias name (e.g. `www.nus.edu.sg`) for some "canonical" (the real) name
- `value` is canonical name (e.g. `mgnzsqc.x.incapdns.net`)

type = MX
- `value` is name of mail server associated with `name`

- **Distributed, Hierarchical Database**: DNS servers form hierarchy to distribute mappings. Contains root servers (for Top Level Domain TLD servers), TLD servers (e.g. uk, sg), Authoritative servers.
- Local DNS servers have local cache, acts as proxy, forward query into hierarchy if answer not found locally.
- **DNS Caching**: Cache mapping, which expires after some time (TTL: time to live).
- DNS runs over **UDP**.

DNS Name Resolution



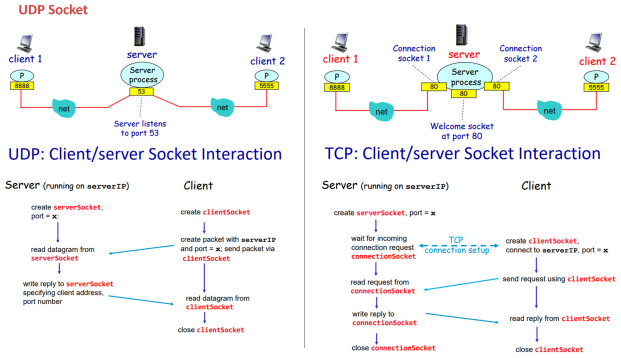
Socket Programming

Applications treat the Internet as black box, send/receive message through sockets.

- Two types of sockets
- **TCP**: reliable, byte stream-oriented socket
- **UDP**: unreliable datagram socket

UDP vs. TCP Socket

- **UDP Socket:**Sender attach des IP address + port no. to each packet. (OS inserts add. info source IP and port). Receiver extracts sender IP + port number from packet.
- **TCP Socket:** Attempts to establish TCP connection to server first. Server TCP contacted creates new socket to communicate with client, allows server to talk with multiple clients individually.



- **TCP vs. UDP Differences**
  - In TCP, two processes communicate as if pipe between them. The pipe remains in place until one of two processes closes it. Sending process doesn't need to attach a destination IP / port number to the bytes in sending attempt as the logical pipe has been established
  - In UDP, programmers need to form UDP datagram packets explicitly and attach destination IP address / port number to every packet.

2. Transport

Transport Layer Services

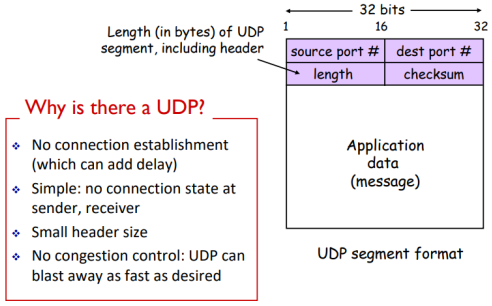
- Transport layer protocols run in hosts.
- Sender side: breaks app message into segments (as needed), passes them to network layer (aka IP layer).
- Receiver side: reassembles segments into message, passes it to app layer.
- Packet switches (routers) in between: only check destination IP address to decide routing. running on different hosts

Transport and Network Layer

- **Transport** layer takes care of logical communication between **processes**.
- **Network** layer takes care of logical communication between **hosts**. (best-effort, unreliable)
- **IP Datagram**: Contains source and dest IP addresses, carries one transport layer segment that contains source and dest port numbers.

UDP: Connectionless Transport

- UDP adds very little service on top of IP.
- **Multiplexing at sender**: UDP gathers data, forms packets, passes to IP.
- **De-multiplexing at receiver**: UDP receives packets from lower layer, checks dest port, and dispatches them to right processes.
- **Unreliable**: UDP transmission used by loss tolerant and rate sensitive apps.



UDP Checksum

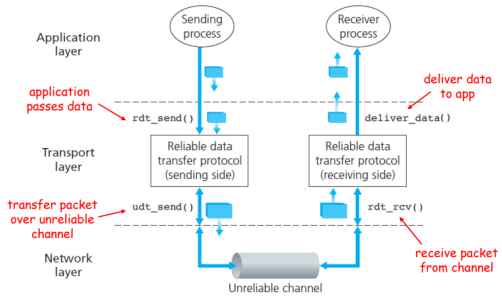
- Allows for error detection, but not correction. May be bit errors when segments are stored and passed in router memory.
- Treat UDP segment as a sequence of 16-bit integers.
- Apply binary addition on every 16-bit integer (checksum field currently 0).
- If carry from MSB, add 1 to result (wrap).
- Compute 1's complement to get UDP checksum.

Principles of Reliable Data Transfer (rdt)

We need to build a reliable transport layer protocol on top of unreliable communication.

- Factors: **Packet corruption, Packet loss, Packet reordering, Packet (Long) Delay.**
- Finite State Machines to describe protocol.

Reliable Data Transfer: **Service Model**



rdt 1.0 (Perfectly Reliable)

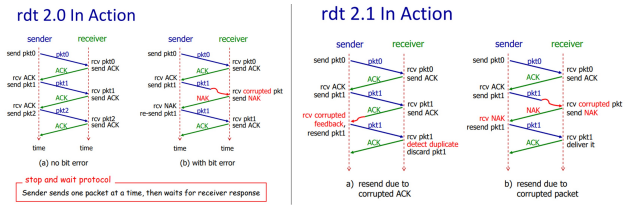
- Assumption: Underlying channel perfectly reliable.
- Sender creates packet and sends, Receiver extracts and deliver data to application.

rdt 2.0 (Corruptable Data)

- Assumption: Underlying channel may flip bits. Use **stop and wait** (for receiver response) protocol.
- Receiver uses checksum to detect bit errors, sends NAK if corrupted. Sender resends if NAK received.
- **Problem: If ACK or NACK corrupted**, no guaranteed way to recover. If packet resent, the receiver will not know it's a duplicate.

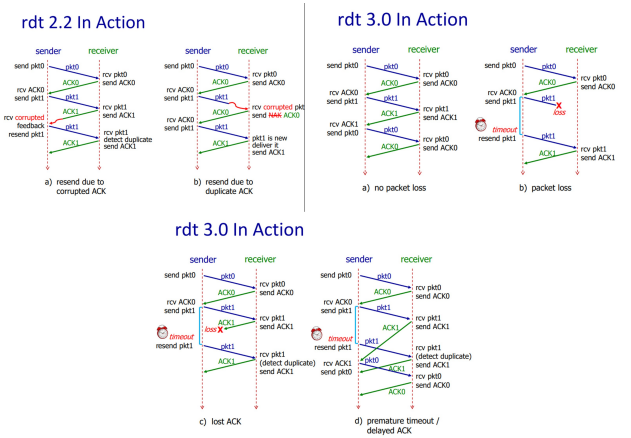
rdt 2.1

- Add **sequence number to packet**, alternate 1 & 0. Sequence number detects duplicates.
- Same as rdt2.0, but receiver knows if it is duplicate.



rdt 2.2

- **Use ACK of last packet sequence number for NAK.**
- Receiver explicitly include seq. no, duplicate ACKs results in retransmit current pkt.



rdt 3.0 (Corruptable, Lossy, Delay)

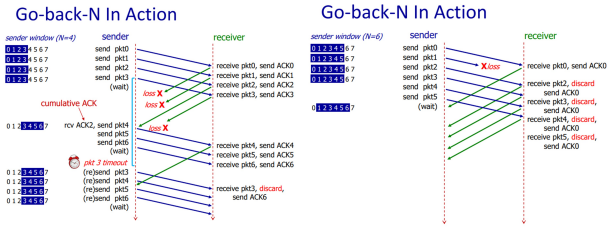
- Assume corruption, packet loss/delay, no re-order.
- To detect packet loss, use **sender timeout**. Sender retransmits if no ACK received till timeout.
- If packet/ACK just delayed, retransmission may generate duplicates but receiver can use seq. no. to detect.
- **rdt 3.0 performance**: Utilisation rate of sender low. For RTT 30ms, L = 8000b, link 1GBps, d_{trans} = 0.008ms, send 8000 bits per 30.008ms. Utilisation 0.027%.
- Stop and Wait limits use of physical resources.

Pipelining

- **Pipelined Protocols**: sender allows multiple, “in-flight”, yet to-be-acknowledged packets.
  - range of sequence numbers must be increased
  - buffering at sender and/or receiver
- Number of packets sent at once is called **window size**.
- **Benchmarked Pipelined Protocols**: Go-Back-N (GBN), Selective Repeat (SR).
- Assumption of corruption, packet loss / delay.

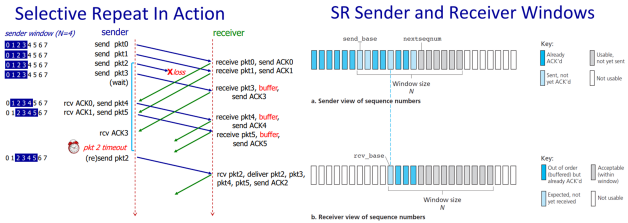
Go-back-N

- Sliding window, slides forward only when ACK received for the leftmost packet in window.
- Requires k bits in packet header for 2^k sequence number.
- Sender keeps only 1 timer for oldest unACKed packet.
- Receiver only accepts ACK packets that arrive in order, discards out of order packets ACK last in-order sequence number. (cumulative ACK).



Selective Repeat

- Receiver individually acknowledges all correctly received packets.
- Buffers out-of-order packets, for eventual in-order delivery to upper layer.
- Sender maintains timer for each unACKed packet, When timer expires, retransmit only unACKed packet.



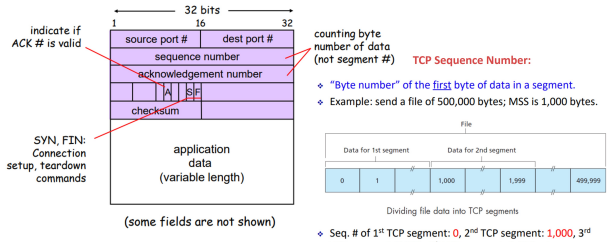
TCP: Connection-oriented Transport

- **Connection oriented**: handshake before sending data. **Point to point**: one sender, one receiver. The connection is **duplex** (bidirectional data flow). **Reliable in-order**.
- TCP socket is fully identified by four-tuple: (source IP addr, source port no., dest IP addr, dest port no.).
- **Multiplexing**: TCP gathers data from processes, form transport-layer segments including app data and 4-tuple pass to network layer.
- **Demultiplexing**: Connection socket created, server already noted 4-tuple. Subsequent packets directed, or demultiplexed, to the appropriate socket using those 4 values.
- TCP creates **buffers** after handshaking.

TCP Segment / Header

- The maximum segment size (MSS) depends on maximum transmission unit (MTU).
- Generally MSS is 1460 bytes, (MTU is 1500 bytes for Ethernet and PPP link-layer protocols.) 40 bytes split half for TCP and IP header.
- TCP Seq. no is ”byte no.”, first b of data in segment.
- TCP Ack. no is ”seq no.” of next b expected by receiver.
- Checksum computation uses 1s complement (UDP same)

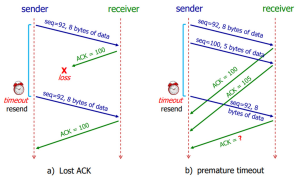
TCP Header



TCP ACK Generation [RFC 2581]

Event at TCP receiver	TCP receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	Delayed ACK: wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	Immediately send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment higher-than-expect seq. # (gap detected)	Immediately send *duplicate ACK*, indicating seq. # of next expected byte
Arrival of segment that partially or completely fills gap	Immediately send ACK, provided that segment starts at lower end of gap

TCP Timeout / Retransmission



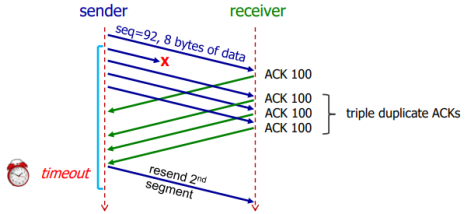
- Random initial sequence number: Minimise probability of some segment from previous connection mistaken as from current connection

TCP Timeout Value

- Determining TCP appropriate timeout value:
- too short timeout: premature timeout and unnecessary retransmissions.
- too long timeout: slow reaction to segment loss. Timeout interval must be longer than RTT – but RTT varies!
- TCP computes (and keeps updating) timeout interval based on estimated RTT. (TimeoutInterval = EstimatedRTT + 4*DevRTT)

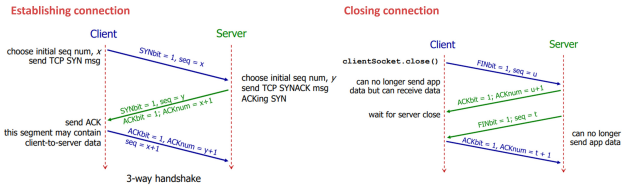
TCP Fast Retransmission

- Timeout period is often relatively long. long delay before resending lost packet.
- **Fast retransmission**: If sender receives 4 ACKs for same segment, suppose segment is lost, resend segment (even before timer expires).



TCP Handshake / Closing

- Before exchanging app data, TCP sender and receiver “shake hands”, agree on connection and exchange connection parameters.
- Closing: Client, server each close their side of connection, send TCP segment with FIN bit = 1



3. Network

asdf