CS2105 Midterm Notes

Chapter 1 - Introduction

What is the internet?:

The internet is a **network of connected devices (network of networks)** that provides many services - the world wide web (www) is just one of the many that are run over the internet. It is the infrastructure that connects hosts/destinations (terminals that are at the end of a network) to intercommunicate.

Examples of network **edges:** End hosts, servers

Examples of network **cores**: Routers, Internet service providers

The internet is run by several entities - IP addressing and naming is done by the Network Information Center (**NIC**), the internet policies and standards are enforced by the Internet Society (**ISOC**), the authority to issue and update technical standards regarding protocols is given to the Internet Architecture Board (**IAB**) while the Internet Engineering Task Force (**IETF**) is the development and engineering branch of the IAB.

What exactly are hosts?:

Hosts run network applications, and these applications communicate using **protocols**. These protocols define the **format and order** of message exchange, as well as **actions taken** upon **receiving/sending** the messages. These hosts access the internet through access networks (can be wireless (WAN) or wired. Wireless LAN like wifi provide smaller scale access within building or a fixed area, while other wider-area wireless access like 3G/4G are provided by telco.

Examples of protocols include:
Application layer:
HTTP (Hyper text transfer protocol), FTP (File transfer protocol), SMTP (Simple mail transmission protocol)

Transport layer: TCP (Transmission control protocol), RTP (Real time protocol)

Hosts connect to the access network over different physical media. Guided media means the signals propagate in solid media (i.e. fiber optics, twisted pair cable, coaxial cable) while unguided media means the signal propagates freely (Radio Waves like Wifi/Cellular, Infrared, Microwaves, Light)

DATA TRANSMISSION WITHIN A NETWORK

1. Circuit switching

End-end **resources are reserved** for 'call' between a source and destination, for the duration of the communication. **Even if there is no data being transmitted, the segment reserved just remains idle and is not shared**. **After the communication ends, then the resources are freed up.**

Pros: Consistent and predictable performance since the resources are only allocated to the communicating parties.

Cons: This strategy is not '**scalable**', meaning that if there is more demand for resources than actual access points, the communications cannot be supported. Also, an explicit setup or teardown is required.

2. Packet switching

The host sending will break the message into **smaller packets of L bits**, which are then **transmitted onto the link at a fixed rate**.

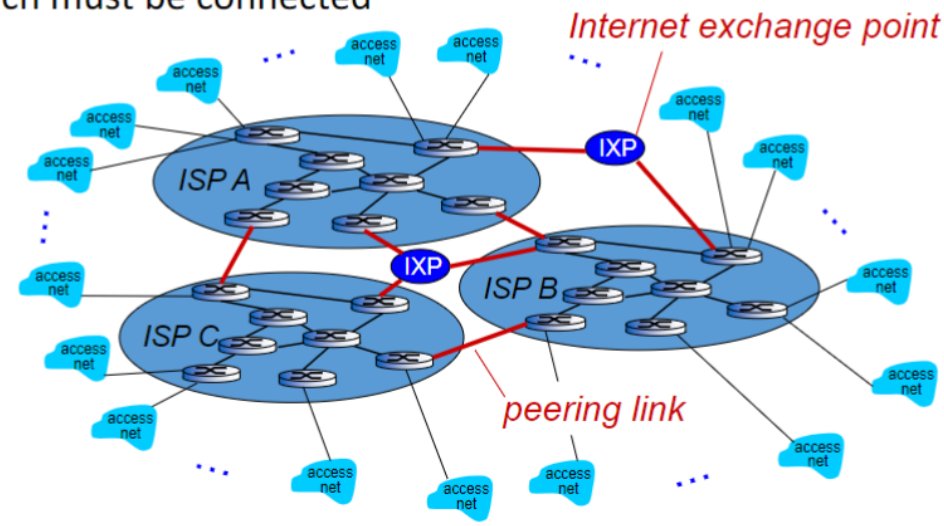
Pros: There would be a more efficient use of resources since multiple packets can send from different sources and share the bandwidth. Resources are not reserved, and can be dynamically allocated and released once the transmission is complete.

Cons: The performance is less 'predictable' in a sense that there could be potential packet loss if the router is overwhelmed (no guarantee that the resource reaches the destination in its entirety.

Store-and-forward means that when packets are passed from one router to another through each link, the entire packet must arrive at a router before it can be transmitted on the next link (Used by the internet). Routers have in-built routing algorithms that determine the route that is taken by the packets from the source to the destination.

Connecting the Internet

- If one Global ISP is a viable business, there will be competitors
  - which must be connected



Content providers could potentially be running their own private network to bring their services closer to users (by connecting data centers to the internet, bypassing ISPs). The purpose of IXPs are such that different networks/ISPs can interconnect directly and share data with each other without routing through a third party, improving routing efficiency.

Hosts connect to the internet via **access ISP**s, which are in turn interconnected via regional ISPs, internet service providers or even higher tier (tier1 ISPs).

Packet Switching Network (in-depth)

Steps:

1. Sender transmits a packet onto the link as a sequence of bits.
2. These bits are propagated to the next node on the link (router), which stores-and-forwards the packet to the next link .
3. This is done all the way until it reaches the endpoint (receiver).

Packet loss potentially occurs when packets are queued in the router buffers to be sent out one by one. Since the buffer (like a queue) has finite capacity, if there are **no free buffer space then the new arriving packets are simply dropped**. Router buffers exist in case 2 packets want to be transmitted together, they have to be done sequentially. **Depending on the protocol**, the lost packet may be retransmitted or not.

Packet delay sources:

1. Nodal processing - time delay incurred when checking bit errors, checking the address to forward the packet to.
2. Queueing - time spent waiting in the queue **for transmission**, dependent on the congestion level of the router.
3. Transmission - pushing the packet into the wire. Formula : Length of packet / Link bandwidth (L/R)
4. Propagation - time taken for the bit to travel across the wire. Formula: Length of physical link / Propagation speed (d/s). **Independent of packet size!**

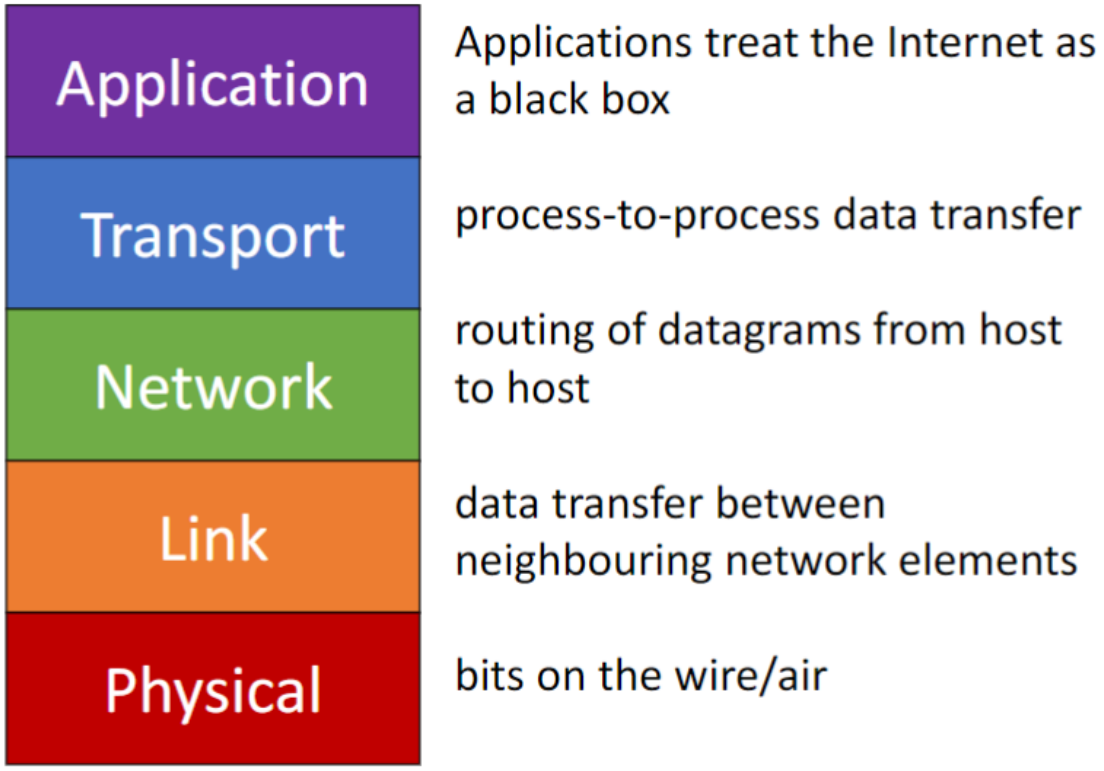
*** Transmission and Propagation are 2 completely separate steps. So once a packet has finished transmitting, the next packet in the queue can start transmitting even if the first packet has not completely propagated to the end point.

Throughput: Bits transmitted per unit time. Refers to end to end communication. For a specific link, we use the term link capacity or link bandwidth.

Protocols

As mentioned above, protocols define the format and order of messages exchanged and the actions taken after messages are sent/received.

Protocols are typically organized into layers depending on their purposes. Each layer provides a service.



Chapter 2 - Application Layer

The Application Layer protocol is used by **every single** internet application. Network applications run on **hosts** and contain communicating processes - referring to Server Processes as well as Client Processes.

Server processes **wait to be contacted** while the **Client** processes **initiate the connections**.

Application Architecture

1. Client-Server

Clients are the one who **initiate contact** with **servers**, **requesting some service** from the server. Servers simply wait for the incoming requests, then provide the relevant requested service to the client.

2. Peer-to-peer

There is **no presence of a server** that is constantly on to provide services. End systems (i.e. hosts) directly communicate with each other, requesting and providing service in return to others. This architecture is tough to manage but it is highly scalable - eliminate bottleneck of a central server, distributed resources.

3. Hybrid - combination of client-server and P2P

Instant messaging is a form of hybrid architecture. The communication between two users is peer to peer (direct communication between ends) but some services are centralized (i.e. online status which is communicated to a central server)

What are priorities for services for apps?:

1. Data integrity - reliable data transfer where every single packet sent is received on the other end
2. Timing - low communication delay, packets are sent through as quickly as possible
3. Throughput - some apps require minimum bandwidth to be effective
4. Security - data integrity is not compromised, encryption of important data (like passwords)

Application layer protocols

They define the type of messages exchanged, message syntax and semantics as well as rules of the exchange (how apps send and respond to messages). These **protocols often make use of internet transport protocols** to facilitate communication over the internet.

Network processes can be identified through the combination of IP address and port number. Each host has a unique IP address (32 bit number) and the port number is local to the host, and identifies the port where the process is being run on (16 bit number). Assigned by IANA

Internet Transport Protocols (Transport layer)

TCP (Transmission Control protocol)	UDP (User Datagram protocol)
Connection Oriented - Time is required to setup a connection	Does not establish a connection
Flow Controlled - Sender is unable to flood/overwhelm receiver with data	No flow control mechanism
Congestion Controlled - Mitigate congestion across a link	No congestion control mechanism
Reliable - Data is guaranteed to be delivered, but no guarantee on throughput or delay	Unreliable data transfer - data transmitted is not guaranteed to be received by the receiving process

HTTP - Web's Application Layer Protocol

Hyper-Text Markup Language (HTML) is the basis of a webpage. A webpage is a HTML file with several other objects - can be another HTML file, jpeg image, audio file etc.. A web object is accessible using a Uniform Resource Locator (URL).

Web resources are requested and received using the HTTP - Hyper-Text Transfer Protocol. It makes use of a client/server model, where the client is typically a web browser that requests, subsequently receives then displays Web objects, and server is the one responding to the requests with the required web objects.

HTTP/1.0 - 1996, HTTP/1.1 - 1999, HTTP/2 - 2015, HTTP/3 - 2022

HTTP is designed to remain **stateless**, i.e. it maintains no information about past client requests. Cookies help to maintain state between sessions.

HTTP **makes use of TCP** as **transport service**.

Client will **initiate a TCP connection** to the server, which has to **accept the TCP connection** request from the client. HTTP messages are **exchanged** over this connection, which is subsequently closed upon the conclusion of the communication. This process repeats for each object in the html file.

RTT (Round Trip Time): Time taken for a packet to travel from the client to server and back.

HTTP total response time includes: 1 RTT (For establishing connection back and forth), 1 RTT for HTTP Request (and response) + File transmission time

Persistent HTTP vs Non-Persistent HTTP

Non-Persistent HTTP (HTTP/1.0)	Persistent HTTP (HTTP/1.1 onwards)
Per object requires 2 RTT (Each object is sent over a separate TCP connection)	After the initial TCP connection is established and response to the first HTTP request, TCP connection is left open. Subsequent HTTP messages are sent over the same TCP connection
OS overhead for each TCP Connection	As soon as referenced object is encountered, request for object is sent over the TCP connection
Parallel connections are often required to fetch referenced objects (multiple TCP connections started at the same time, each transmitting one object)	

Sequential vs Pipeline vs Multiplexing

1.0/Sequential: Requests-response have to go out and come in one at a time. For example if there is request1, response 1 has to come in before request 2 goes out.

1.1/Pipeline: Requests can flow out but the responses have to come in in order (i.e. request 1, 2, 3 can send out but 1, 2, 3 need to come in sequentially)

2/Multiplexing: Requests can flow out and the responses can come in in any order, even if the response is a partial response (half the packets of 1, then 3, then 2,

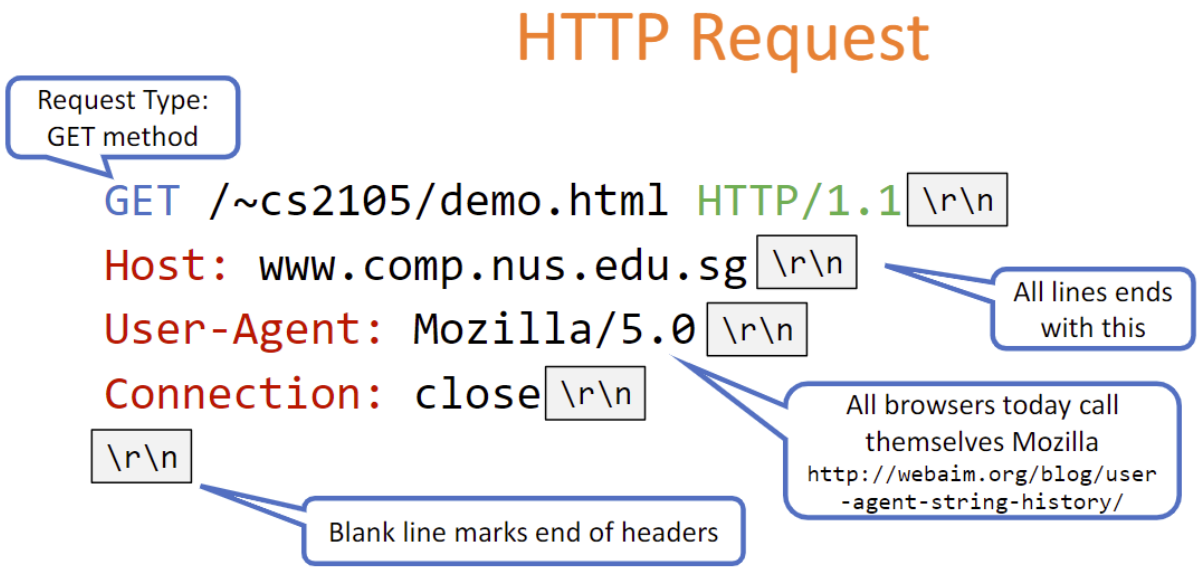
then 1 again etc.)

HTTP/3 makes use of UDP instead!

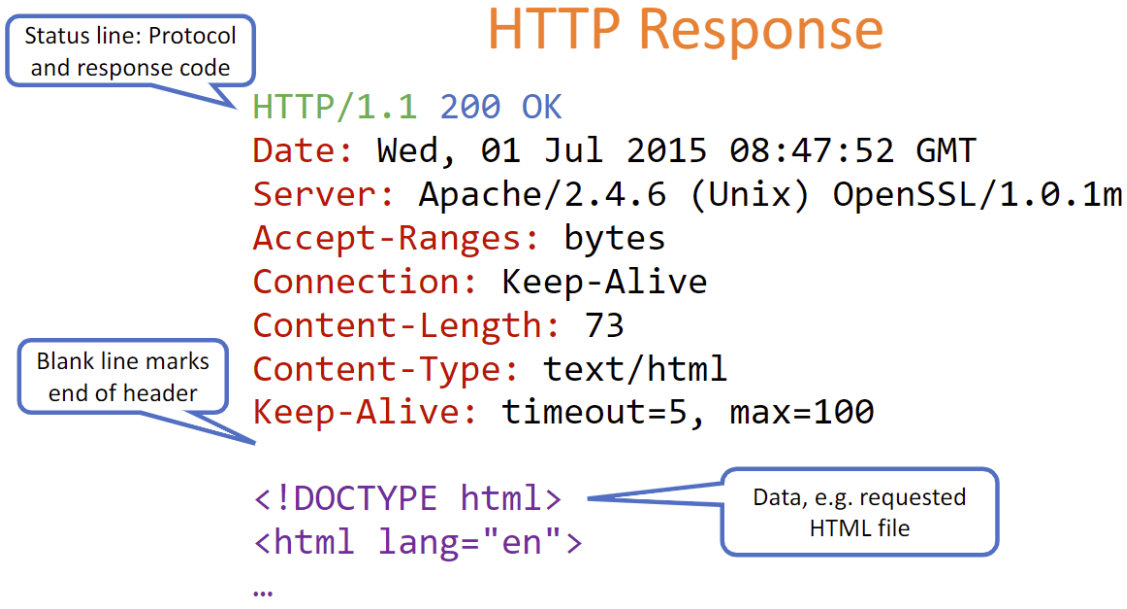
Multiplexing: Multiplexing is the process of combining multiple data streams into a single stream for transmission over a communication channel. It allows multiple signals or data streams to share the same transmission medium efficiently.

Demultiplexing: Demultiplexing is the reverse process of multiplexing. It involves separating a combined data stream into its individual constituent streams upon reception at the destination. Demultiplexing allows the receiver to extract and process each data stream separately, ensuring that the intended recipients receive the correct data.

HTTP Request



HTTP Response



STATUS CODES

200 - Request ok

301 - Moved permanently

304 - Resource unmodified since a certain time

403 - Server declines to show the webpage

404 = Resource not found

500 - Unspecified error

Resource caching is sometimes employed so we do not need to keep redownloading resources if they are unaltered. Specify the date where we want to check if the resource is last altered in the HTTP request, and the response should be different based on whether the object has indeed been modified.

Modified: Send **new object** with response code **200**

Unmodified: Send **response** with response code **304**

telnet: Telnet is a network protocol used to establish a text-based interactive communication session between two devices over a network. It allows a user to

log in remotely to a server or device and execute commands as if they were physically present at the device's terminal. Telnet operates primarily over the Telnet protocol (TCP port 23). Use case: testing network service. telnet causes a **NS lookup to the IP address** and **TCP SYN to be sent (attempt to send and establish connection**). However, no HTTP request is sent, and if the port is not open telnet will not open it.

cURL: cURL (Client URL) is a command-line tool and library for transferring data with URLs. Use case: Making HTTP requests and debugging/testing web applications and APIs.

DNS

Hosts can be identified using 2 methods: the first is the host name, and the second is the IP address. The DNS (domain name service) translates between the two.

DNS Resource Record

Mapping between host names and IP addresses (and others) are stored as Resouce Records (RR)

RR Format: <name, value, type, TTL>

Type	Name	Value
A (adress)	Hostname	IP Address
NS (name server)	Domain, e.g nus.edu.sg	Hostname of authoritative name server for domain
CNAME (canonical name)	Alias for real name, e.g. www.comp.nus.edu.sg	The real name, e.g. www0.comp.nus.edu.sg
MX (mail exchange)	Domain of email address	Name of mail server managing the domain

nslookup and dig: Query DNS servers, nslookup is the more basic version while dig provides more information

DNS records are stored in distributed databases implemented in a hierarchy of many name servers - From the root server, has many smaller servers, there are a total of 13 root servers.

Top level domain servers are responsible for: .com, .org, .edu, .net and top level country domains (.sg, .eu). Authoritative servers are organizations' own DNS, and they provide authoritative hostname to ip mappings for internal named hosts (webmail etc.), and are self maintained by organizations or service providers. Local DNS servers are found in every organization, and when a host makes a query, it is sent to the local DNS which acts as a proxy and forwards the request up the hierarchy if it cannot resolve the request.

DNS runs over UDP and can make use of recursive queries or iterative queries (but typically use iterative). DNS caching occurs, for as long as a **TTL**.

Recursive query means i.e. local host ask local DNS, which asks A. If A doesn't know, A will ask B, which will ask C and so on. Iterative means i.e. local host ask local DNS, which asks A. If A doesn't know, it lets local DNS know. Local DNS will then query B and so on.

Chapter 3 - Socket Programming

Applications run in hosts as processes, and within a single host, two processes communicate using IPC. Processes in different hosts communicate by exchanging messages. An **IP address can identify the host** but it is **not** sufficient to identify a process.

Port numbers identify a process → Coordinated by AINA

Socket is the software interface between processes and transport layer protocols. Each **process sends out and receives messages to and from its socket**. A socket consists of an **IP address and a port number** and is used to locate a process within a host.

A **process** can however create **multiple sockets** and it typically uses a **new socket for each new connection/HTTP request** it receives from a browser.

There are two types of sockets - stream socket and datagram socket

Datagram socket: A datagram socket uses UDP as its transport layer protocol

Stream socket: A stream socket uses TCP as its transport layer protocol.

UDP

In UDP, data is sent as packets (datagrams).

Only one socket is required for communication, and communication is done with anyone through the same socket.

In a UDP communication, the packet is created by the application, which specifies the recipient's IP address + port. The return info (source IP and port) is automatically attached to the packet by the OS., and this allows the receiver to identify the sender.

TCP

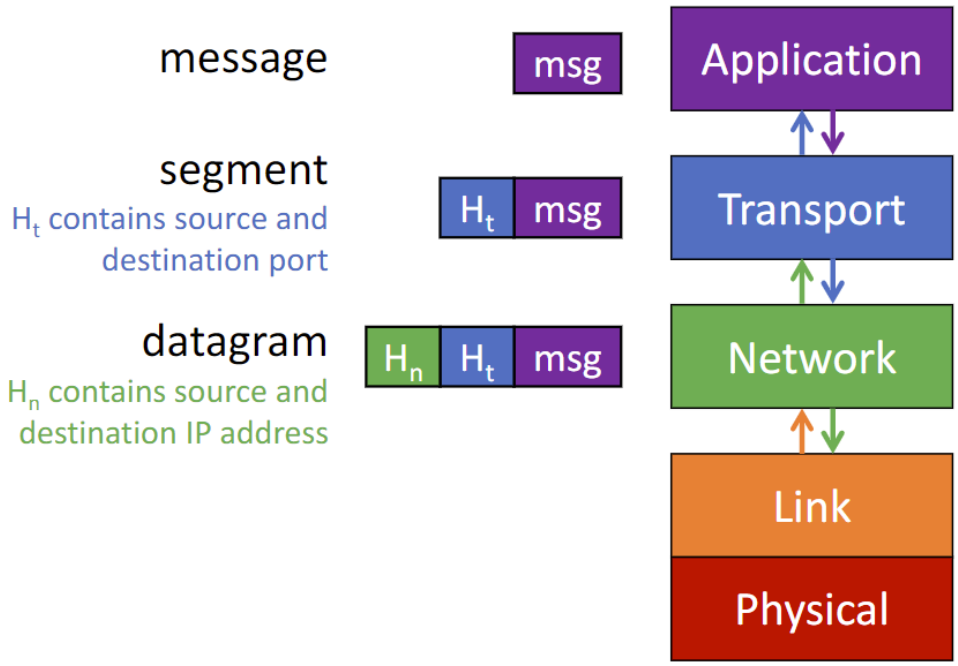
In TCP, data flows as a continuous stream.

As previously mentioned, processes establish a connection to another process first, and only when the communication in place does the data flow between the processes in streams.

Server has a **welcome/listening socket**, and when it is contacted by client, a **new socket is forked** for the server process to **communicate with the client process** (establish connection). Client creates socket to establish a **TCP connection** with the server.

Every **connection has its own socket instance**, and a socket instance **can have the same port number as another socket instance**! (i.e. multiple instances of ip:80 to support multiple http connections)

Layering



TCP process communication simulates having a ‘pipe’ between two communicating processes which enables data to keep flowing through unless this connection is closed by either the client or the server. Sends data in segments.

UDP datagram packets need to explicitly attach the **destination IP**/port number to every packet.

Chapter 4 - Reliable Protocols

Transport layer: Process - Process

Transport layer protocols run in hosts - at the sender’s end, they segment messages and pass them to the network layer. At the receiving end, the transport layer reassembles segments into a full message and passes it into the application layer.

Network layer is ‘best effort’ and unreliable. It provides host-host communication.

Reliable data transfer over unreliable channel

Possible issues include:

- Packets being dropped (lost)
- Packets bring corrupted
- Packets being re-ordered
- Packets taking a long time to be delivered

A reliable transport service should eliminate all the above problems, and guarantee packet delivery and correctness, and deliver packets in the same order they are sent.

The reliability of the protocol increases with its complexity.

rdt1.0 - Assume that the channel is perfectly reliable

rdt2.0 - Assume that the channel has bit errors (bit flips can potentially occur)

How can we detect errors: Checksum, to determine if the data sent is identical to the data received

How can we recover from errors: Acknowledgements, telling the sender if the packet is OK/NOT OK. If the packet is not OK. then the sender retransmits the packet upon receiving the NAK.

This makes use of the stop-and-wait protocol, which means one packet is sent at at a time and only when the receiver response is obtained then another packet/the same packet is transmitted.

FLAW: What if the feedback is corrupted? (i.e. corrupted ACK/NAK) - retransmission is not the best solution because if the feedback was actually an ACK, then the duplicate packet will be sent! This problem is handled in rdt2.1

rdt2.1 - Duplicate handling from rdt2.0, with the addition of sequence numbers

At the receiver end, when the packet is received, if the duplicate is identified (same sequence number), then the packet will not be forwarded up to the application layer and discarded instead.

rdt2.2 - Instead of having ACK/NAK messages, only use ACK with the sequence number of the last correctly received packet. Now without NAK, if the sender receives duplicated ACKs (i.e. ACK4 ACK4 then it knows that packet 5 has not been received at the receiver end, and retransmission of that packet occurs)

rdt3.0 - Assumes that packet can be either **lost** or **corrupted**

How can packet loss be detected and handled: Sender waits a certain amount of time for ACK, and if timeout occurs then retransmit. This can handle both lost packet, and lost ACK. Duplicates, like in rdt2.1, can be detected by the sequence number

Protocol name	Summary	Error Handling
rdt1.0	Assume everything is as per normal and the channel is perfectly reliable	-
rdt2.0	Assume that bit flips can occur. Stop and wait, meaning every packet has to be ACK/NAK before sending next/resending.	Acknowledgement of reception using AK/NAK
rdt2.1	Assume the ACK/NAK messages can be corrupted	Addition of sequence numbers to packets to detect duplicates
rdt2.2	Same assumption as rdt2.1	Using ACKs only, with the ACK being for the last packet received correctly
rdt3.0	Assume that packets can be lost or corrupted	Re-send after waiting for some time. This can handle both lost packet and lost ACK.

Utilization refers to the fraction of time that the link is being used: **Time spend sending message / Total time.**

Stop-and-wait utilization: dTrans / dTrans + RTT

Stop-and-wait throughput: Number of bits / dTrans + RTT

Pipelining: Sender allows several packets to be “in-flight” at the same time - unacknowledged. This can be done by increasing the range of sequence number and some buffer at one/both ends. Same assumptions as rdt3.0

Pipelined protocols:

1. Go-Back-N

   Acknowledgement ‘n’ means that any packet with sequence number smaller than or equals to ‘n; has arrived.

   Timer set for oldest unACKed packet, and upon timeout, retransmit all packets.

GBN Sender

- can have up to N unACKed packets in pipeline.
- insert k-bits sequence number in packet header.
- use a “sliding window” to keep track of unACKed packets.
- keep a timer for the oldest unACKed packet.
- timeout(n): retransmit packet n and all subsequent packets in the window.

GBN Receiver

- only ACK packets that arrive in order.
  - simple receiver: need only remember expectedSeqNum
- discard out-of-order packets and ACK the last in-order seq. #.
  - Cumulative ACK: “ACK m” means all packets up to m are received.

2. Selective Repeat

Selective Repeat: Key Features

Receiver individually acknowledges all correctly received packets.

- Buffers out-of-order packets, as needed, for eventual in-order delivery to upper layer.

Sender maintains timer for each unACKed packet.

- When timer expires, retransmit only that unACKed packet.

This means that less retransmission is needed because packets are not discarded

Selective Repeat: Behaviors

Sender

data from above:

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase, sendbase+N]

- mark pkt n as received
- if n is smallest unACKed pkt, advance window base to next unACKed seq. #

Receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

pkt n in [rcvbase-N, rcvbase-1]

- ACK(n)

otherwise:

- ignore

	Go Back N	Selective Repeat
Number of UnACK	N packets in pipeline	N packets in pipeline
ACK Style	Cumulative	Selective
Out of Order	Discard	Buffer

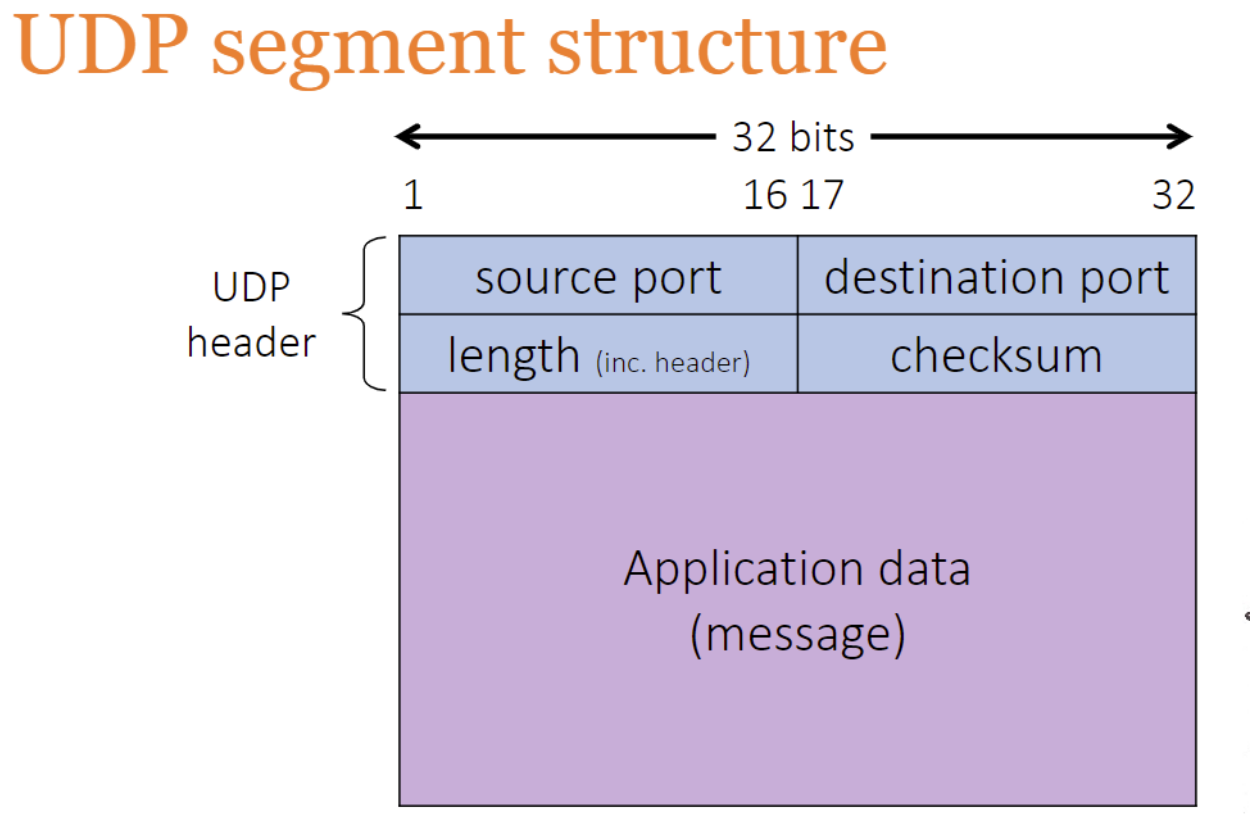
Timer	Oldest UnACK packet	Each UnACK
Retransmit	All UnACK packets	One UnACK

Chapter 5 - UDP and TCP

Internet protocols are described in Request for comments (RFC)

UDP

UDP adds connectionless mulitiplexing (more than one signal to be sent from a single channel) and checksum on top of the IP, and its transmissions as mentioned above are typically **unreliable**. To achieve reliable transmission, application implements error detection and recovery mechanisms.



A UDP header is 8 bytes in length - first 2 bytes for source port, 2nd 2 byes for destination port, 3rd 2 bytes for length and finally 2 bytes for checksum (to detect bitflips). Then the message follows.

Pros of UDP:

1. No communication setup delay
2. No connection state at sender or receiver, need less resources
3. Smaller header size
4. No congestion control

The purpose of the checksum is to detect errors. The sender, after coming up with the message, will compute the checksum value and include the checksum value into the checksum field. Receiver when opening the message will then compute the checksum on its end, then compare to the checksum from the header. If match, the message is uncorrupted.

Checksum algorithms include: SHA-1/SHA-2, CRC, MD5, UDP/TCP Checksum

Steps:

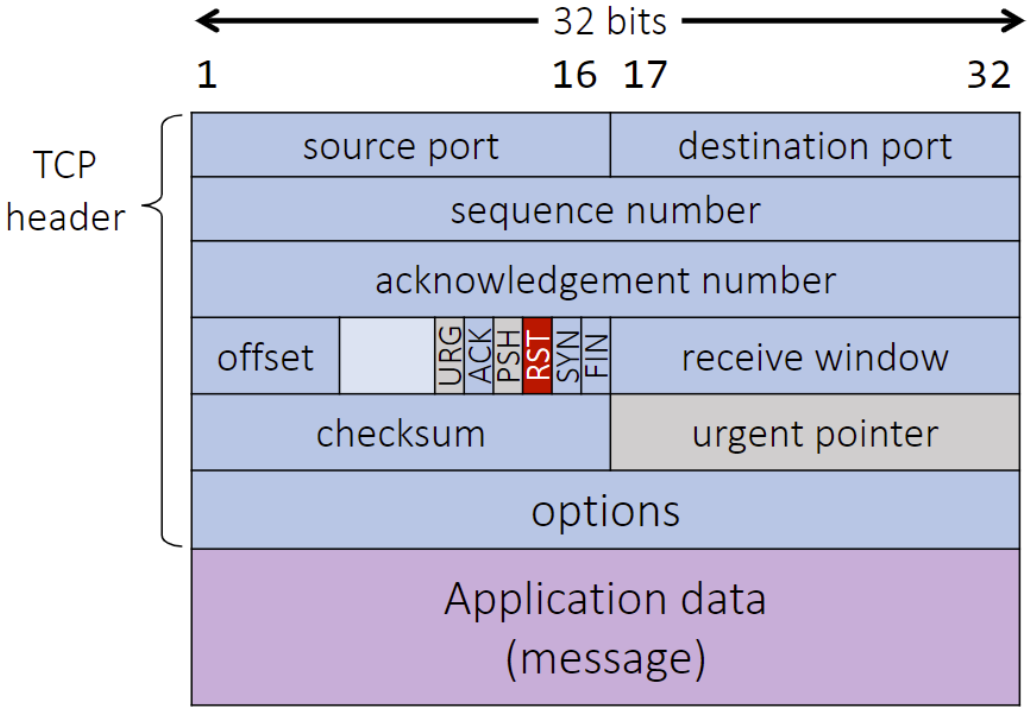
1. Segment into 16-bit integers
2. Add integers with wrap around carry
3. Ones complement (bit flip everything)

- If more than 2 segment, and combining 2 of them gives extra bit, then ones complement it first before adding the next

TCP

Connection oriented and reliable, with flow control. Application passes data to TCP, which forms segments in view of the maximum segment size (which does not include the size of the header).

TCP segment structure



First 2 bytes: Source port

Second 2 bytes: Destination Port

Checksum is 2 bytes, offset is ~6 bits and refers to the size of the header (number of **rows** of the header, **in 32 bit words**)

Sequence number: Byte number of the first bye of data in the segment. Counts bytes, not segment numbers

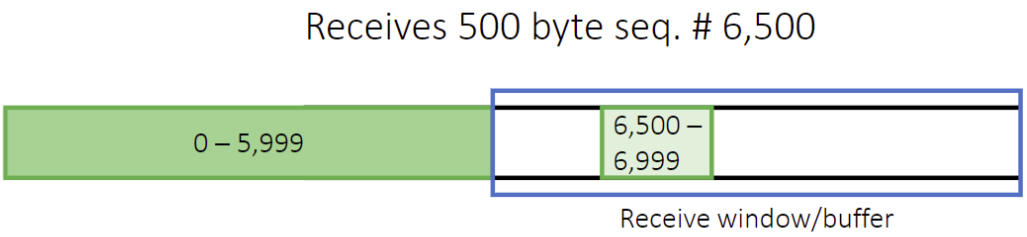
Acknowledgement number: Sequence number of the next byte of data expected. ACKs up to the missing byte (Cumulative)

SYN/ACK/FIN: For synchronization and termination

TCP Acknowledgement Number

Seq. # of the next byte of data expected

TCP only ACKs up to the missing byte (cumulative ACK)



Note: TCP specs does not say how out-of-order segments should be handled

Only when byte 6000-6499 is received then ACK #7000 is sent.

ACKs are 'piggybacked' on a data segment, i.e. no specific message needed for a ack, but if no more data can send on its own.

Cumulative ACK prevents retransmission of data should there be a lost ACK.

Timeout value is important because too short a timeout could lead to too many retransmissions, but too long a timeout value could lead to slow reaction to loss.

Timeout should be > RTT. If 3 duplicate ACKs received, then resend the segment immediately.

We can estimate RTT by first taking a sample RTT once every RTT. Then, keep updating the RTT based on the following formula:

Take SampleRTT (RTT_s)

- *Once every RTT*

Compute EstimatedRTT (RTT_ε)

- $RTT_\varepsilon = (1-\alpha)\cdot RTT_\varepsilon + \alpha \cdot RTT_s$
- typical value of $\alpha = \frac{1}{8}$

Setting Retransmisson Time Out (RTO)

Compute Deviation of RTT

- $RTT_{dev} = (1-\beta)\cdot RTT_{dev} + \beta \cdot |RTT_s - RTT_\varepsilon|$
- typical value of $\beta = \frac{1}{4}$

RTO Interval is set to

$$RTT_\varepsilon + 4 \times RTT_{dev}$$

estimated RTT

"safety margin"

Establishing a TCP connection:

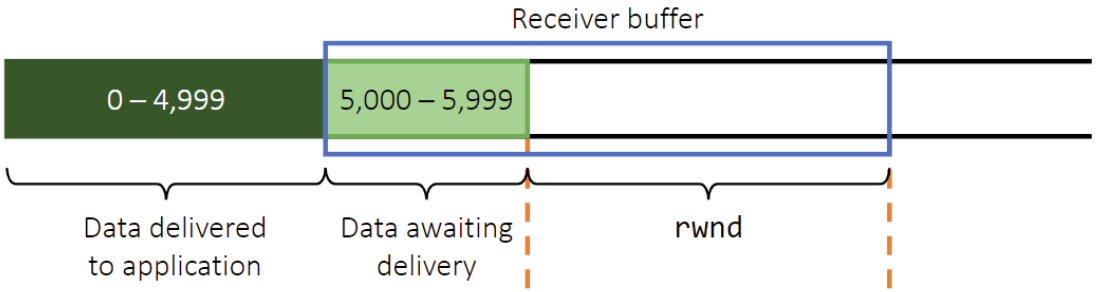
1. SYN request: SYN = 1 + seq
2. Response: SYN/ACK + seq/acknum
3. ACK + seq/acknum

Closing a TCP connection: FIN = 1

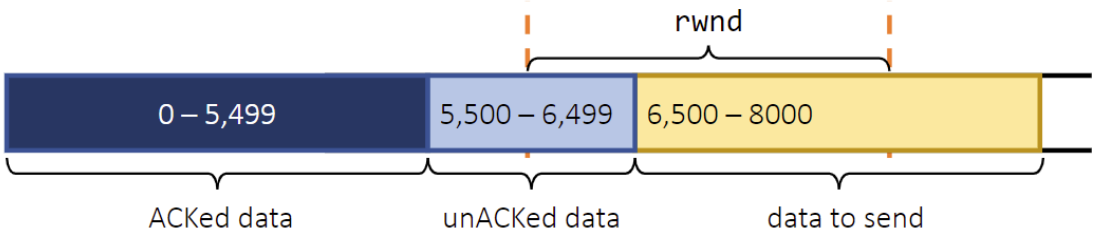
Each side has to send a closing segment with FIN bit =1, which must be acknowledged with an ACK = 1. After closing the connection, no more data can be sent, but data still can be received (and ACKed). (no more seq)

TCP Flow Control

Receiver buffers data to application



Tells sender how much data it can send



If rwnd is 0, then the sender sends a 0-data segment.

Midterm Review

Question Number	Topic/ Question	Score / 3
1	HTTP Browser Experience	2
2	Purpose of TCP Handshake	2
3	Messages, what could possibly cause this?	2
4	Send RCV for GBN/SR	3
5	Sending packet, delayed	2
6	Calculations	2
7	Reading dig lookup	3
8	IP Address required for email	3
9	GBN sending	0

10	Checksum	3
11	SR Sending	3
12	Reading HTML Request	3
13	Things based on HTML Request	2
14	Properties of Reordering Req	1
15	Transmitting Bits to Router	2