When we want to change or make variations we need to know the protocol

- http at 2.0 now keeps changing protocols keep changing
- google protocol – sdp

what is the internet??

- A network of all local network that allows devices to communicate with one another
- Network of networks of interconnected isps
- Infrastructure that provided services to application option C, such as instant messaging social networks etc.
- Provides a programming interface to web simple api with send/receive messages

NETWORKING LINGO MEANINGS

- Host/end systems – a device connected to a network
- Bandwidth – transmission rate of data
- Packet switches – splits data into chunks known as packets
- DSL – digital subscriber line

Internet contains multiple protocols

- Tcp
- Ip
- http
- Skype
- 802.11

Internet standards

- RFC: request for comments
- IETF: internet engineering task force <- publish internet drafts has to go through the stages from IETF long process + rigorous testing

WHAT IS A PROTOCOL

- Protocols define Format/order of how a message is sent/received and actions taken on the message transmission,recipient ← IMPORTANT FOR ASSIGNMENT

Network edge:

- Consists of hosts that are clients and servers
- Access networks – wired/wireless connection type
- Network core – connection of routers think ISP

Dial up DSL

- Voice and data communicate over same line so dedicated to either one only!
- Useful for outback areas where low population cheaper to setup
- Used time multiplexing

ADSL (DSL – digital subscriber line)

Different frequency on same cable

Adsl depends on distance speeds are directly related from distance to supplier, this is because they use copper wiring! In optic fibre everything travels in electric pulses at speed of light so propagation delay is minimal! Also higher noise and attenuation with longer distance.

- Frequency multi-plexing so splits bandwidth for voice and other for network connection

- 0-4khz for the PSTN
- White space for splitting
- Upstream frequency
- Downstream frequency

DSL is a dedicated network no sharing its dedicated

Access net: cable network shared cancer

- Frequency division multiplexing, so channel for control/data/ and white space same as DSL

NBN fiber to the home

- Fully optical fiber from the supplier to house
- Verizon/google/nbn/fios
- 30mbps -> google fibre (1gbs)

Power over ethernet is where power is supplied from power line

- Access point ethernet from cable

Ethernet over power – power line networking

- Connect power outlets to the ethernet directly

Enterprise access networks (ethernet)

- Constantly evolved currently using switch ethernet, every device connected to cable with switch

Wireless access networks (WAN)

- Shared wireless access connects host to router at access point
- AC5300 standard shown for wireless

NETWORK CORE

- Where the routers reside
- Mesh of interconnected routers/switches
- IT IS THE CARRIER OF YOUR TRAFFIC

Two forms

- Circuit switching used in old telephones
- Packet switching current in day

Circuit switching

- Requires path pre-defined from the start--> end of where the data is going, purely circuit
- Establish circuit beforehand before transmission of data
    - FDM
        - Frequency domain multi-plexing
            - Divides the frequency between users so all users
    - TDM->SHIT
        - Time domain multi-plexing
        - 1 user at a time receives ALL the frequency but can only service one user at a time
        - Entire frequency given to user at time for fixed amount of time
    - Circuit establishment source->client has delays
        - Delay to check the source/check destination of message and if a path exists!/check if it has the capacity to reach destination
        - Termination at end
            - PROS OF CIRCUIT SWITCHING
                - Reliable in terms of data transfer -> guaranteed bandwidth!
            - CONS OF CIRCUIT SWITCHING
                - Requires a pre-defined path from source -> destination so very restrictive!
                - Whenever any router connected disconnect can possibly delete some paths
                - Whenever any network standards change has to be re-built
                - Waste of resources, if a certain amount is allocated to a user and they decide to do NOTHING with it, then that bandwidth is just wasted as guaranteed bandwidth! Waste of resources!!! No sharing!
                - Routers participating in circuit switching need to keep state maintenance

FIRST PRINCIPLE of network design KEEP THE CORE SIMPLE NO COMPLEXITY CIRCUIT SWITCHING IS COMPLEX DUE TO LAST CON LISTED. We don't want routers to remember any state or anything just the basics, eg. When we make a linked list in C we make it generic so it can have multiple uses making it complex working with strings only etc. is bad design!

Independent of data from source

Routers only contain routing tables no info from source/data

All network applications deploy on end system rely on host! No network applications go on router all on host!

Circuit switching is not feasible because it is

- Inefficient
    - Communication is very bursty -> load a page, then downtime whilst maintaining page
    - No sharing during down time so wasted resource!

    - Cannot be changed when new networking standards come
- Fixed data rate
    - We don't always want to have the same 10mb/s rate constantly as this can be seen as a waste of resources its not useful
    - Eg. We download want fastest speeds, when reading a web page or browsing web pages don't need much speed
- Connection state maintenance is difficult
    - Not scalable
    - Would waste resources maintaining state

PACKET SWITCHING

- Data is sent as chunks formatted bits as packets
- Packets have header and a payload
    - Header contains information on the destination and entire package! Has the entire control information! Think amazon shipping! Shipping + item! Item is payload, shipping info is the header has destination and information on order
        - Header contains
            - Internet address
            - Age (TTL) time to live <- how old the packet is
            - Checksum to protect header, to see if there are missing packets (sanity check)
            - Your own address!!!! Source address
- Packets travel independently, there is no defined path like in circuit switching, like in the amazon shipping example packets can choose to take different more efficient paths, like in amazon shipping not everything is delivered all at once even if it is purchased at once!

Payload is the data being sent!

Header has the instructions for how to handle the packet sort of like an api instruction set.

Core router utilise header to find the relevant path, the destination uses the header for sanity check + see who the message is from.

Advantage of packet switching is that packet loss isn't as detrimental as message switching, losing 1/5 parts is better than losing the entire message, we can always extrapolate or atleast see from the 4/5 parts delivered. Can re-transmit aswell with sanity check

When a hop receives a packet, it will use it's routing table to check if there is a path to the destination, if it does it will forward the packet to the appropriate path, else would terminate.

Packet switching timing, routers only check header for destination. Almost negligible time in processing the packet at switch can consider that overhead cost as 0. Only information needed is header. For router to forward to next hop.

Store and forward switching is when the router will wait for the entire packet to arrive before forwadding it to the next hop, so it will not only wait for the header (only information it actually needs) but it will also wait for the payload which causes a larger delay because now we are waiting for information the router is not requiring. However, this is used in public internet as it is more SAFE

and secure, if we don't wait for the entire packet this can cause an issue with the checksum, the header may corrupt/destination corrupt and we are no longer able to do our sanity check as our header/payload are separate.

Cut through switching is a newer technique where routers will only wait for the header to arrive to the router before forwarding the data to the next hop, this causes significantly less delay because the router no longer has to wait for the data to arrive in the payload, it can already send it out on its path as the only information the router uses is in the header! However, this can arise some problems and is not used in the public internet as it is a lot more dangerous store and forward is more safe as we can do sanity check at each hop, with cut through the checksum cannot be accurately calculated as its dependant on the payload too and if the payload is lagging behind the header this can cause issues!

Packets can take any route (most efficient!) we have no control on what the path for the route is for the entire message! Routers maintain NO STATE all they do is see a packet coming, read the header, and set it out to destination path based on routing table! Will not handshake with a bad header!

Statistically multiplexing

- Relies on the assumption that not all flows will burst at the same time causing overload
- Not guaranteed and has some fail cases

When we overload we want to store packets in the buffer queue to avoid overloading this is to absorb transient overload, the overflow will die down. Buffer will absorb transient overload

If the overload is not transient, the queue will be full, we cannot have infinite buffer (unfeasible) extremely costly and physically not possible. To handle persistent overload we block the queue when it is full, and we drop packets ← not so good. RTT could be different on the packets. If queue is empty, no delays quick RTT, but with congestion longer RTT

Pros of packet switching

- Doesn't maintain any state! Scalable
- We don't have to pre-define any path from source to destination so multiple paths
- Delay depends on state of router
- Overhead costs increase as the routers have to process header however, this is also true in circuit switching as the ROUTER has to process the message.
- Supports bandwidth sharing allows more users to use network no guaranteed bandwidth depends on activity and congestion on router, delay is dependant on router

Cons of packet switching

- Can drop packets
- Comes with an overhead cost

Packet switching allows more people to use network suppose we have 1mb/s link

Each user is active 10% of time -> 100kb/s split between users

Circuit switching

- Max capacity 10 users to guarantee 100kb/s

Packet switching

- Can hold infinite amount of users
- With 35 users on network, the probability that more than 10 users are active at the same time is less than .0004 -> Bernoulli trials and binomial distribution (statistical multiplexing gain)

Global internet → connecting the network of networks

Internet Structure

- Global ISP like Telstra etc that connects to access nets and all users joining are connected to the users.
- Multiple global ISPS since there is economic competition
- To connect global ISPS together we can set up
    - Peering links high speed link between ISPS private, or we can use
    - IXPS → internet exchange point
        - Saves money on infrastructure
        - Requires less hop considerabely since we don't need to hop to the peering link
    - Content providers can run their network on the IXP providing faster connection for their services.
- Regional ISP
- AARNET
    - Network for australia's academic and research

Key Properties of Links

- Bandwidth is the width of a link (like the width of pipe flowing water) instead width of link flowing data
- The longer the pipe longer delay between when pipe is opened and the water is received, similar to networking, propagation delay
- Keep efficient flow through pipe, we need to constantly pump the volume of the pipe to have an efficient link
- Bandwidth delay product gives the volume of the link, it is the bandwidth * propagation delay

Packets queue in router.

- When the buffer is full packets are dropped

Delays in the correct order

- Nodal processing delay
    - Checks bit errors - checksum
    - Determined destination
    - Normally <millisecond
- Queueing delay → difficult to predict
    - Time waiting at router before transmission
    - Dependant on router congestion
    - If we know the size of the buffer we can calculate delays in the queue
- Transmission delay
    - Packet length (size of packet) = L
    - Bandwidth of link (bps) = R
    - = L/R
- Propagation delay
    - Length of the link = d
    - Propogation speed -> how fast it moves through the link = s ← typically 2*10^8m/s
    - = d/s

For N routers in the hops, if the queueing delay is the same total delay = n * sum of all delays

Queueing delay

- If a router can process 1 packet per second
- Packets arrive at → a packets per second
- The packet length → L
- Bandwidth of link → R
- Total delay = AL/R

If the router can process 1 packet per second, and we send 1 packet per second, our queueing delay would be 0 as queue would be empty!

If the router can process 1 packet per second, and we send 10 packets in a burst every 10 seconds, the arrival rate would be the same however, there would be a queueing delay since 9 packets wait in queue while first is being processed.

THE PATTERN OF THE TRAFFIC MATTERS FOR QUEUEING DELAY!!

If aL > R → queue will full up and packets will drop

aL/R is the traffic intensity typically when aL/R = 1 → queueing delay == 0

end-to-end delay is the sum of all delays across all hops on path

throughput

the rate at which bits are transferred between sender and receiver.

Instantenous = rate given at a point in time

Average = rate over whole period of time

Bottle neck occurs when one link is smaller than others

PROTOCOL LAYERING

Keep core simple

Layering

Three design steps

- Break down the problem into tasks → divide and conquer
- Organize the tasks
- Decide who does what

What does it take to send packets across.

DECOMPISITION

- Task 1: send along a single wire
- Task 2: connect task 1 N times till it reaches end systems

Bits/packets on a wire ->physical layer

Deliver packets within local network ->datalink layer

Deliver packets across global network -> network layer

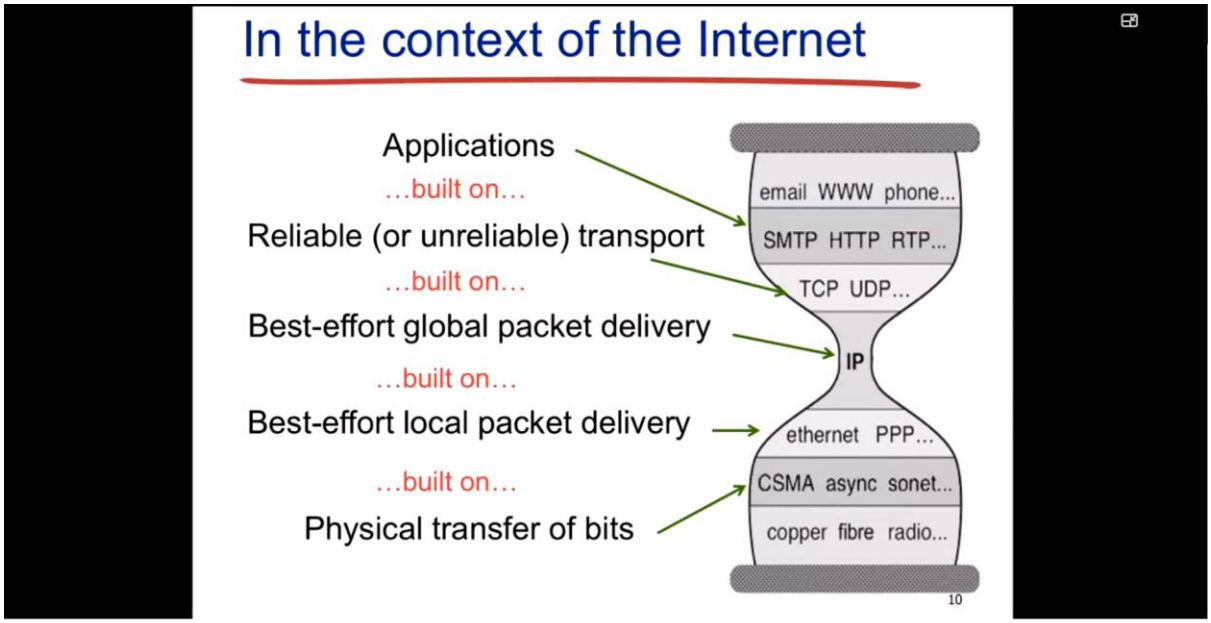
Make sure packets get to the destination -> transport layer

Process the data -> application layer

Abstraction

➔ Hiding the contents whilst providing the service

Applications are built on reliable or unreliable transport built on global packet delivery built on local packet delivery built on physical transfer of bits.



Application

- Network applications

Transport

- TCP

ONE IP LAYER unifying protocol

The routers only work in the network layer no application layers

Benefits of layering

- Abstraction ← security
- Tasks can be split up
- Modularity – easy to tell where errors are in.
- Easy to integrate different systems, existing applications don't need modifications with layering

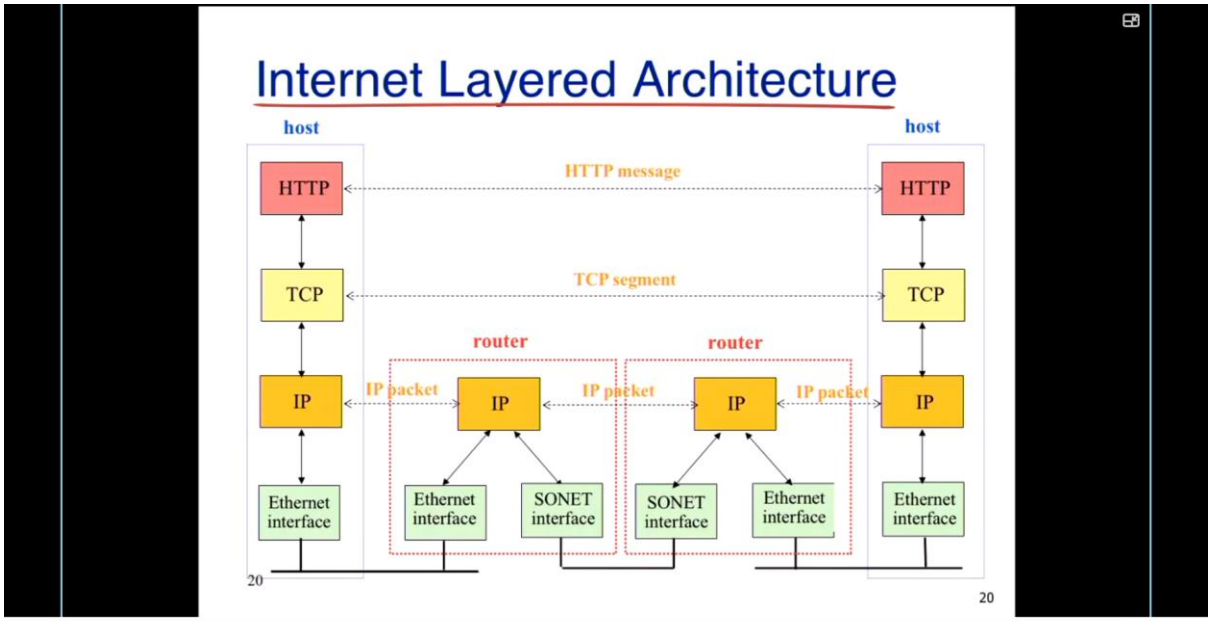
Disadvantages of layering

- Larger headers, TXP + IP + Ethernet headers are up to 54 bytes
- Worse performance overall → information hiding
- Checksum being used by multiple layers, duplication of efforts
- Layer violations when the gains too great to resists or when the network doesn't trust ends eg. Firewalls

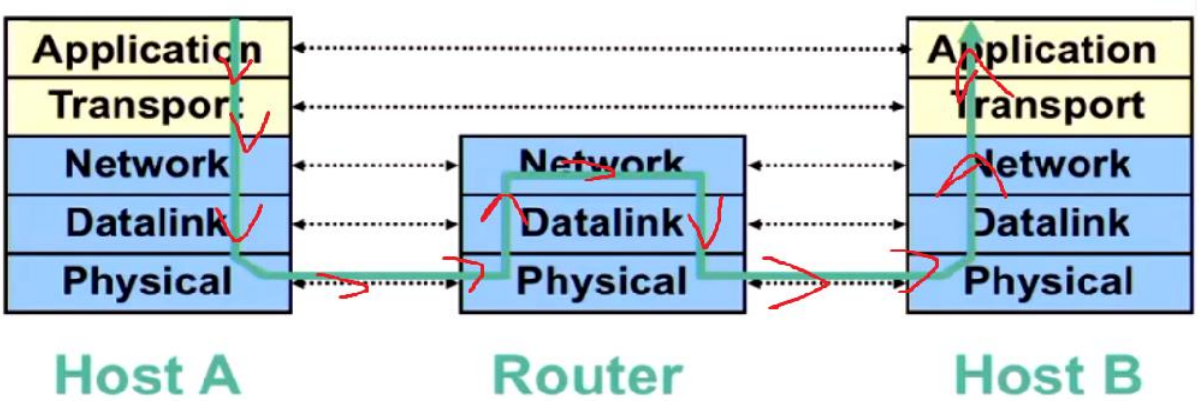
Bits arriving on the wire must make it to the application

All layers must exist on the host

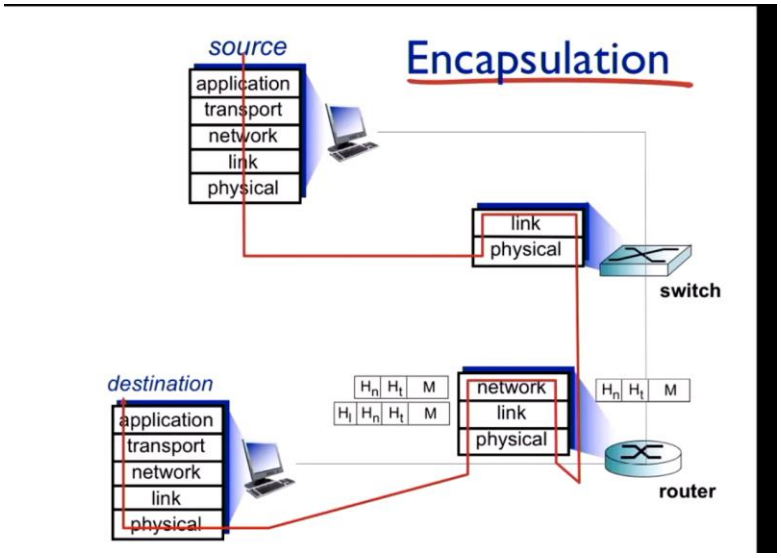
Bits arrive on wire at router, physical layer, packets must be delivered to next hop, data link layer, routers participate in global delivery, networking layers, routers don't support reliable delivery, transport layer and above are not supported



Layers interact with peer's corresponding layer



Down stack when transmitting, and up the stack when receiving



Encapsulation → LARGER OVERHEAD AS YOU GO DOWN THE STACK YOU ENCAPSULATE LEADS TO LARGER HEADERS, PDU -> concatenating to the end of the PDU

Application layer – message → M

Transport layer – segment → Ht + M adding header of the transport layer to the message

Network layer – datagram → Hn + Ht + M adding header of network layer to segment

Link Layer – frame → Hl + Hn + Ht + M adding header of link layer to the datagram

Switch then transports the frame through the link layer and physical layer to the router, note the switch has no networking layer

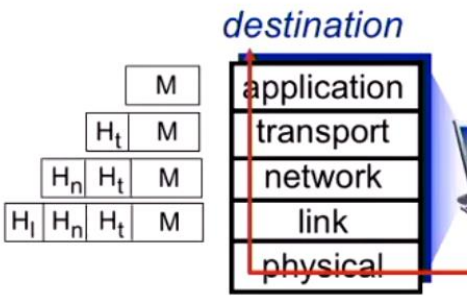
At the router we process the incoming frame

Travels to the link layer → frame → Hl + Hn + Ht + M processing the header at the link layer of the router.

Travels up to the network layer for transmission → datagram → Hn + Ht + M processing frame

Travels back down to the link layer as a new frame → Hl + Hn + Ht + M

While processing at the destination we reverse this process returning the message



Transport layer header – 20 bytes

Network header – 20 bytes

Datalink header – 18 bytes -> 14-byte for header 4 bytes for tailer

58 bytes of extra information just to carry 1-byte message

We have to add 58 bytes to the packet to send a message just for encapsulation

Application Layer

TCP → reliable

UDP → unreliable

When we create a network application, it can only be deployed on a host or end system, NOT A ROUTER OR IN NETWORK CORE → KEEP CORE SIMPLE. Test on own machine, if it works on own machine work, it will work anywhere else

IPC interprocess communication

Processes talk to each other via IPC

- We need abstractions to communicate to different machines

SOCKETS -> NEED IP ADDRESS + PORT NUMBER

Processes send/receive messages to/from its socket

Sockets can be thought of like a door that connects the application layer and the transport layer

- Sending process → pushing message out door abstraction, push it out and other part will handle for you

To identify messages processes must have an identifier, host devices have a unique 32- bit ip address

IP address only provides access to networking layer of machine, we must know which process the packet is being delivered to. This is done using port number.

We can uniquely identify a socket by knowing it's ip address and its port number to know which process to address on the machine.

UDP → SEND TO ip address + port number

TCP → SEND, already established connection

Identifier for a process consists of an ip address and a port number assosciated to the process on the host. HTTP server → port 80, mail server → port 25

To send a http message to cse.unsw.edu.au we use the ip address and port 80

Client server architecture

Server

- Exports well-defined response/request interface
- Process that sits listening for requests
- When receives a request it will carry it out
- Always on the host
- Permanent IP address
- Static port conventions, http:80 ssh: 22 email:25
- Data centres for scalability
- Communication available between other servers

Clients

- Short lived processes that make a request
- The users of the application
- Starts communication with server
- May be intermittently connected
- May have dynamic ip addresses
- Do no communicate with each other all done through medium of the server

Peer to Peer (P2P) architecture

- No always on server
- Arbitrary end systems communicate directly
- Often used for
    - File sharing (BitTorrent)
    - Games
    - Video chat/distribution
    - Distributed systems
- User downloads file off site and then they themselves become a server hosting for others once they have it

Peer to peer pros and cons

- Pros
    - Clients who discover each other can communicate without server, in client server no communication if server goes down
    - Self-stability, new peers bring new service capacity (bit torrent when you host) as well as new service demands, ← bit torrent
    - Speed: parallelism
    - Reliable communication
- Cons
    - State uncertainty, there is no shared memory or clock
    - Action uncertainty: users are in control or if mutually conflicting decisions
    - Users have free will at anyway they can walk away from the system

Application layer protocol

- Types of messages, message syntax, message semantics, rules need to be provided

There are open protocols eg. HTTP, SMTP or properiatery protcols like skype or private ones

Transport service an app requires

- Data integrity, throughput, timing and security.
- Data integrity
    - Some applications require 100% reliable data transfer
    - Other applications eg. Audio can tolerate some less
- Timing
    - Some apps require low delay to be effective eg. Games or movies
- Throughput
    - Some apps require a minimum amount of throughput (bandwidth rate) to be effective
    - Other apps can dynamically change based on throughput
- Security
    - Encryption and data integrity

Transport service requirements: common apps

application	data loss	throughput	time sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 50kbps-1Mbps video:100kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few msecs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
Chat/messaging	no loss	elastic	yes and no

38

Internet apps: application, transport protocols

application	application layer protocol	underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

40

Employ TCP through their own application layer protocol dependant on the application

Btw socket address = ip address + port address

Hyper text transfer protocol (HTTP) uses TCP

Webpage consists of objects

Objects are different file types

Web page consists of a html file which consists of several referenced objects

Each object can be addressed by a url

Request index.html as soon as establishment of connection is setup (incurs some delay making connection)

For each object on different server, another connection must be established to retrieve the object.

Uniform resource locator (URL)

Protocol://host-name[:port]/directory-path

Directory path – reflects the file system

http overview

- Request -> response system client, server paradigm

- Index.html

http is stateless

- Server maintains no information about past client requests

Protcols that maintain state are complex! Keep the core simple.

HTTP messages has two types

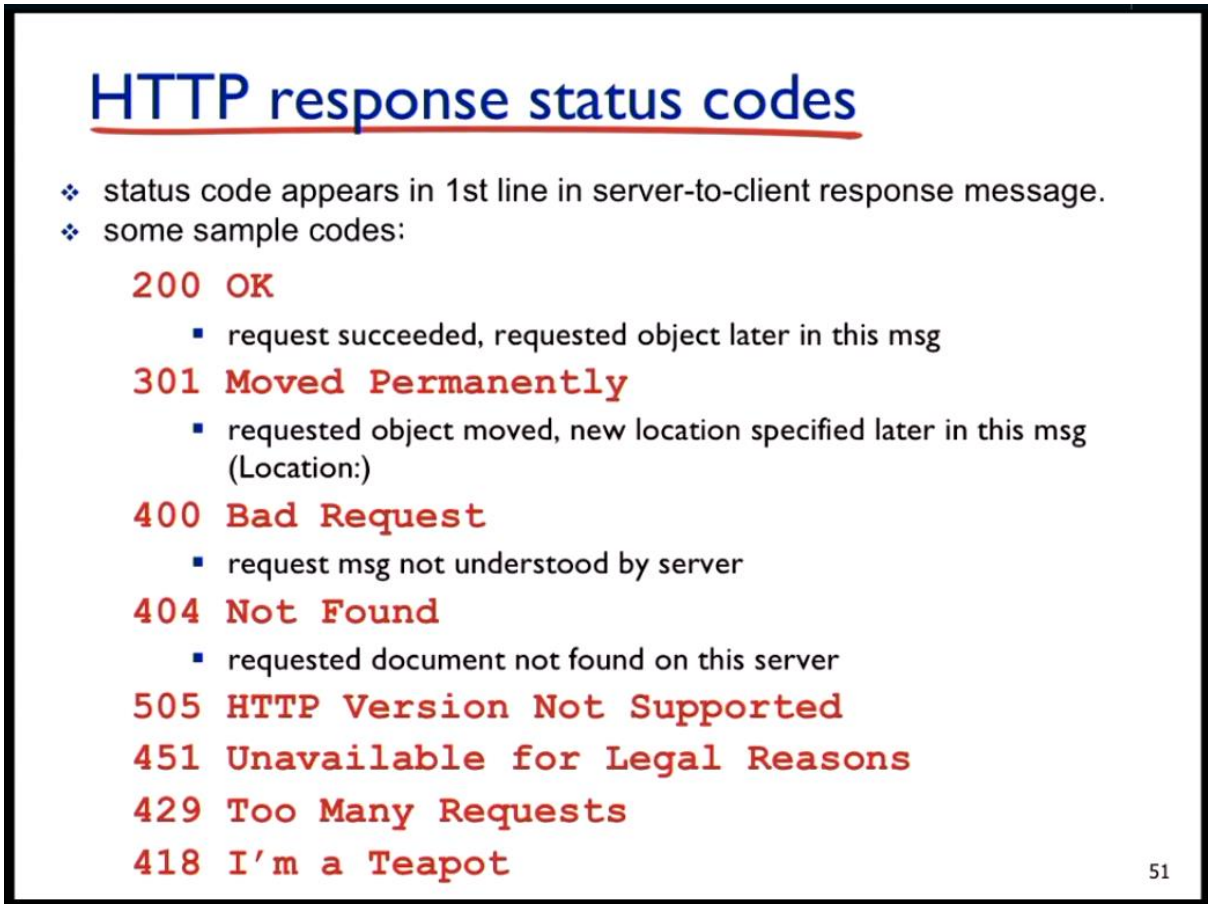
Requests/response

Request -> GET/POST

Carriage return character + line feed character (\r\n)

HTTP response

Has the response 200 ok means everything is okay, multiple different types of responses however, comes with a status code + status phrase.



LECTURE 3

Rquest Method types

HTTP/1.0:

GET → request page

POST → uploads user response to form

HEAD → ask server to leave requested object from response

HTTP/1.1

GET,POST,HEAD

PUT → upload file in entity body to path specified

Uploading form input

POST method

- Web page includes form input
- Input is uploaded to server in entity body, method="post" in url

GET method:

- Uses GET method
- Input is uploaded from url field

HTTP is a stateless protocol → doesn't remember you

Many websites use cookies to remember who you are so that you don't need to constantly authenticate user, saves your state! Eg. If you go online shopping needs to remember your cart

COOKIESS

Four components

1) Cookie header line HTTP response message
2) Cookie header line in next HTTP request

Set cookie sent by server → client

Cookie file is then saved onto client somewhere

Next time client contacts server, it will use that cookie in the header line. Server will recognise the cookie and then retrieve the information associated to that cookie.

Cookies allow for targeted advertisements

Cookies come with disadvantages too

- Sites learn a lot about you personal details included
- 3rd party cookies can follow you across multiple sites, you can turn them off but its unrealistic

3rd party cookies

- Accessing a web server
- Whenever you load a page all objects on different servers are requested → creates a cookie there, third party cookie.
- Double clicks create a cookie that can then track your browser usage
- 2007 google acquired double click → 3.1 billion dollars
- Double click allows targeted advertisements for a user
- DoubleClick → 1 pixel of unrecognisable data to create a cookie sneak 100

Performance of HTTP

- Page load time (PLT)
    - from click until user sees page
    - key measure of web performance
- depends on factors such as
    - page content/structure
    - protocols involved
    - network bandwidth and RTT

performance goals

User requires

- fast downloads
- high availability

Content provider requires

- happy users
- cost effectiveness and infrastructure

Network requires to

- avoid overload

caching and replication helps achieve these goals

How to improve PLT

Most web pages have multiple objects, eg. Images embedded

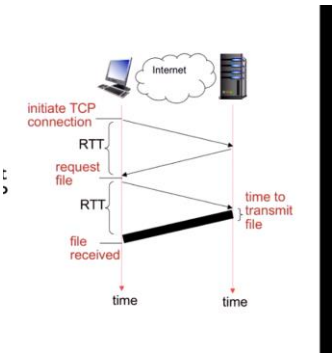
Non persistent HTTP(HTTP 1.0)

Connection fetches a single object, non persistent connection!

Done shitly

One object at a time

This creates multiple delays



HTTP response time

- ONE RTT to initiate TCP connection

- One RTT for HTTP request and response
- File transmission time
- Non-persistent HTTP response time = 2 RTT + file transmission time

HTTP 1.0 OR NON PERSISTANT HTTP

Poor PLT

Needs to set a connection for EACH object → higher delays because requires multiple TCP connections to the SAME SERVER

Sequential requests/responses even when resources are located on DIFFERENT SERVERS no parallel requesting

MULTIPLE TCP connections slow-start phases

Everytime you have to create new connection, lose momentum

Downsides of parallel HTTP connections

- Can lead to network overload
- Congestion in network, since its all bursty → sequentially queue = 0
- Assume 40 objects on one server, with primitive approach will create 40 requests on same server sequentially 1 at a time. Parallel will create 40 requests at one time, will increase network load, can be harmful

Persistent HTTP

- Server leaves TCP connection open after sending response
- Subsequent HTTP messages between same client/server use the same TCP connection → much more efficient than creating a TCP connection each time
- Allow TCP to learn more accurate RTT estimate
- Allow TCP congestion window

Persistent without pipelining

- New requests only when previous response has been received
- One RTT for each referenced object
- More efficient than non-persistant as it keeps connections open that will be re-used

With pipelining

- Default in HTTP/1.1
- Sends out multiple requests at once sort of like multi-threading
- As soon as it encouters a referenced object → send a request, as low as one RTT for all objects

Response time of one page with three embedded objects

- Initiate TCP connection → 1 RTT
- Request file -> time to transmit file → 1 RTT

How to improve PLT

- Reduce content size for transfer → compression
- Change HTTP to make better use of bandwidth → persistent connections and pipelining

- Change HTTP to avoid repeated transfer of same content → caching
- Move content closer to the client → closer servers

Caching

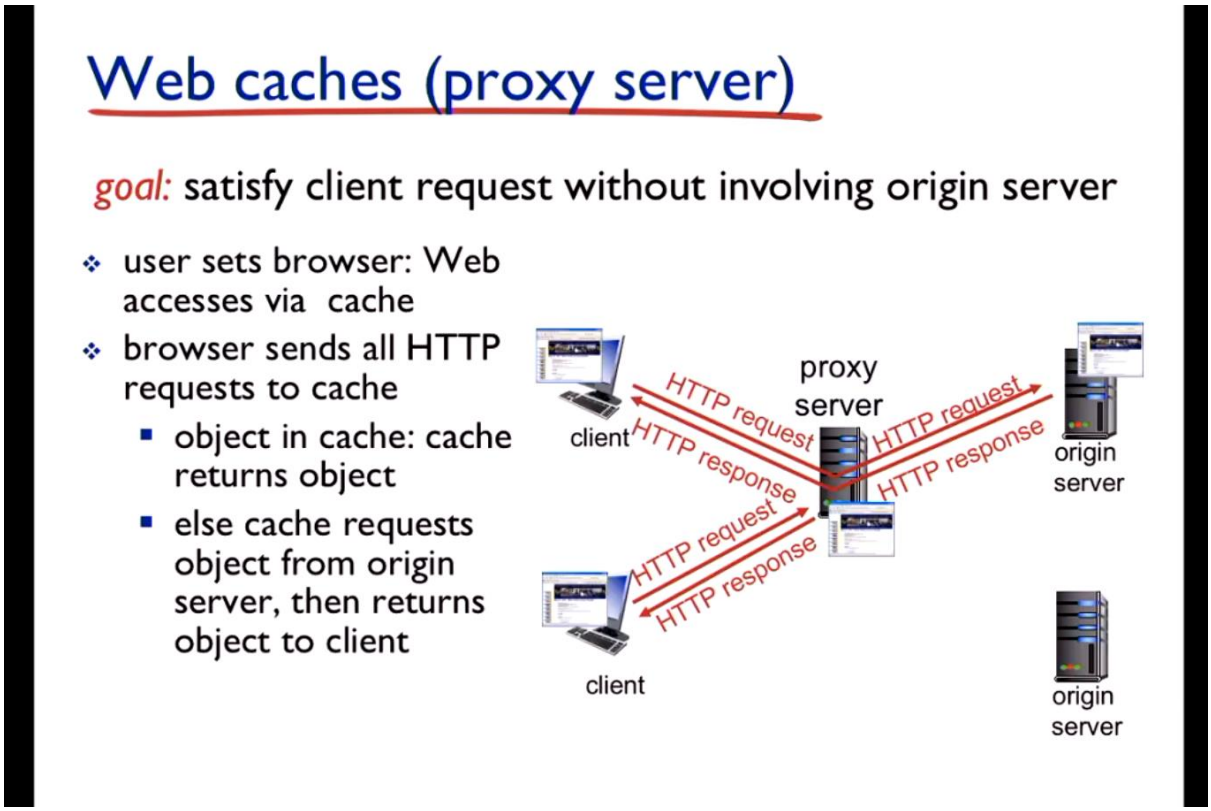
- Allows you to load a state much faster by using a reference
- Works well up to a point, however many unique request. Large overlap in content

Web caches (proxy server)

To attempt to satisfy client request without involving the original server

Client sends HTTP request, response is received through server, proxy server works through origin server, and caches the response

All new objects are cached, and existing objects in cache returns the object



Cache acts as both client and server,

When the first request is made, cache acts as a client requesting data, and then when it is received it can then act as a server for the data.

Cache is installed by ISP typically

Caching reduces response time for client requests, reduces traffic on access link and enables poor content providers to deliver content

Caching example

Assumption

Avg object size → 100K bits

Avg request rate from browsers to origin server → 15/sec

Avg data rate to browser → 1.5mbps

RTT access to origin server → 2 seconds

Access link rate: 1.54Mbps

aL/R ~= 1 → 1.5/1.54

LAN has 1gbps speed → utilisation = 1.5/1000 → 0.15%

Access link utiliszation = 99% → insane!

Total delay = internet delay + access delay + LAN delay

= 2 sec + minutes + usecs msecs

If we increase access link to 154 mbps we are reducing access link utilisation to 0.99%! reduces the delay from minutes to micro seconds, however this is a much more expensive approach!

Better option, local web caching → using 1.54 MBPS access link rate

All subsequent requests do not use access link → use LAN (local web cache)

Assume cache hit rate is 0.4

40% of requests satisfied at cache → LAN speed

60% of requests satisfied at origin → access link

Access link utilization is reduced to 60% now (add in the 40% from cache (very small)

This is even faster than 154MBPS access link as it utilises the lan speed, access link requirement down to 60%!

Likelihood of cache hits

- Distribution fo web objects requests follow a zipf-like distribution
- The probability that a document will be referenced k requests after it was last referenced is roughly 1/k. web traces exhibit excellent temporal locality

SIMILAIR TO VIDEO CONTENT CREATERS → 10% OF THE TOP POPULAR VIDEOS ACCOUNT FOR NEARLY 80% OF VIEWS WHILE REMAINING 90% ACCOUNT FOR THE REMAINING 20% OF VIEWS

More popular content is → more likely is to be cached!

To check if the cached version is still valid we can use CONDITIONAL GET REQUESTS

If modified since will make sure a resource has been modified since a given date, otherwise will send error 304 response

To make a resource uncacheable, we can use the expires header line to specify a date where we no longer cache!

Replication → clones' website to create another web server for global distribution similar to google hq in Australia. This is done with network providers CDN (content distribution network) servers. Or how league created server in Sydney for Australian players → lower RTT

CDN to improve HTTP performance

Caching and replication used as a service

CDN combination of pull caching and push replication

- Pull → direct result of clients requesting
- PUSH → expectation of high access rate

HTTPS!!!

HTTP is insecure

Basic authentication with HTTP uses base64 encoding → easily converted into plain text

HTTPS: HTTP over a connection encrypted by transport layer security

Provides

- Authentication
- Bidirectional encryption

Server push content method

Servers can push content and reduce overhead, they provide index.html and give you the 4 objects to complete the page rather than client creating multiple requests for other objects, so there is less overhead on the client, server can process the 4 objects rather than client.

Can lead to wasted resource if user requests and then closes browser

Application layer → EMAIL AND DNS

Electronic mail has three major components

- User agent for composing mail with address of mail server → normally browser/installed application on PC
- Mail servers receives mail in queue. Once it is your turn it will create a TCP connection and push out your mail to recipient mail server.
- Simple mail transfer protocol (SMTP) runs ontop of TCP (reliable as TCP is reliable)

Mail server contains

- Mailbox contains incoming messages for users
- A queue for messages (like buffer queue)
- SMTP protocol between mail servers to send email messages between different mail servers
    - Client → sender of mail
    - Server → receiver of mail
- NO INTERMEDIATE MAIL SERVERS (DIRECT COMMUNICATION BETWEEN SENDER AND RECEIVER).

- This allows for easy one way communication between sender and receiver

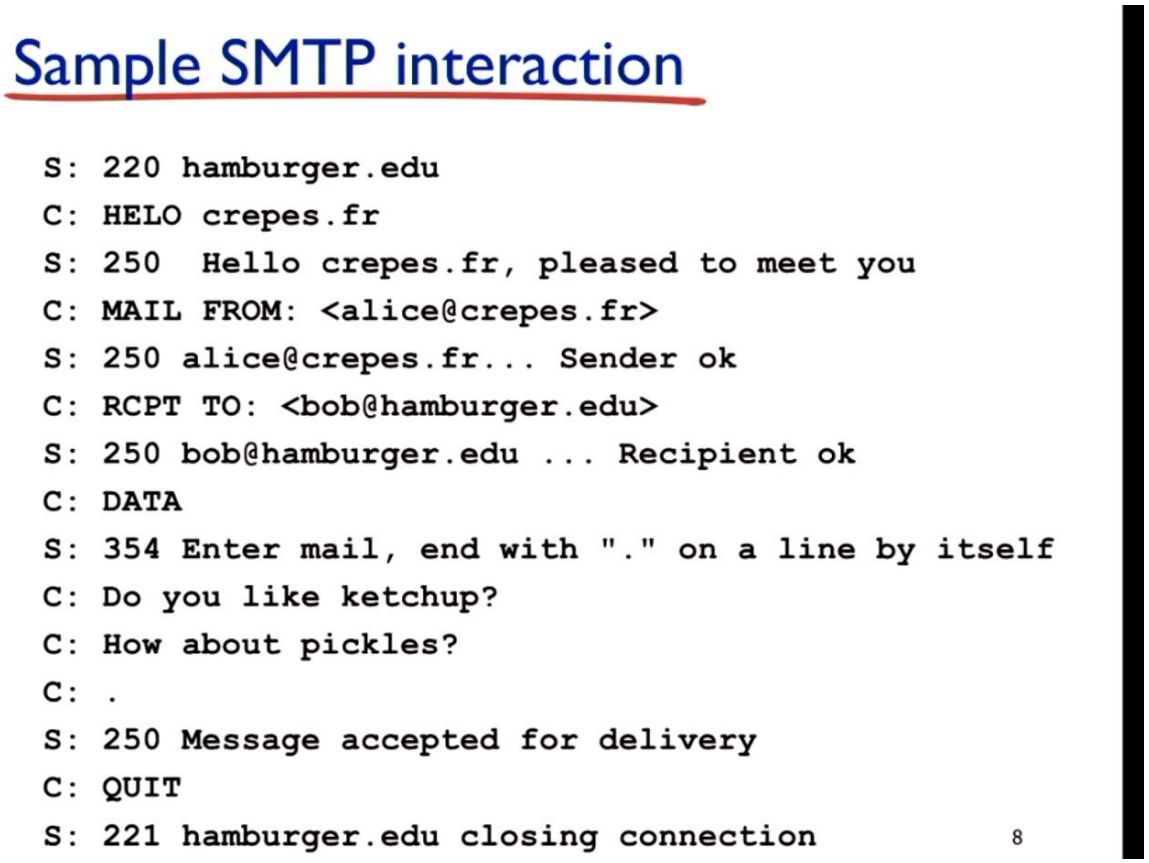
Email SMTP

- Uses sTCP for reliable transfer email message from client to server, port 25
- Direct transfer
- Three phases
    - Handshake (greeting)
    - Transfer of message
    - Closure
- Command response interaction
    - Commands: ASCII text
    - Response: status code and phrase
- ALL MESSAGES MUST BE IN 7-bit ASCII

SMTP is used to get message from user agent and transfer it to the mail box. To transfer it across we do NOT use SMTP, we use TCP to transfer message across!

S = server

C = client



PHISHING

Spear phishing

- Directed at specific individuals or companies, attackers may gather personal information
- Nigerian_prince_email.exe

Clone phishing

- Runescape phishing,
- Attempt to make a legitmate email to make you click on a link to go to their version of the wbesite to steal ur credentials

SMTP

- Uses persistent connections
- Requires message to be in 7 bit -ascii
- CRLF to determine end of message

Comparison with http

- http → PULL
- smtp → Push
- both have ascii command/response interaction/status codes
- each object in HTTP is encapsulated in its own response message
- SMTP → multiple objects sent in multipart message

Mail message format

SMTP protocol for exchanging email msgs

RFC standard for text message format

Header lines → to+from+subject, different from SMTP mail from RCP to: commands!!

Body: the "message" → only ascii

- Why do we have sender's mail server/why do we have a separate receiver's mail server

  - mail servers must be constantly on!

  - we need to transfer responsibility of delivering emails to the mail server instead

  - if we do it on ours we will have to make sure 100% of time connection is on

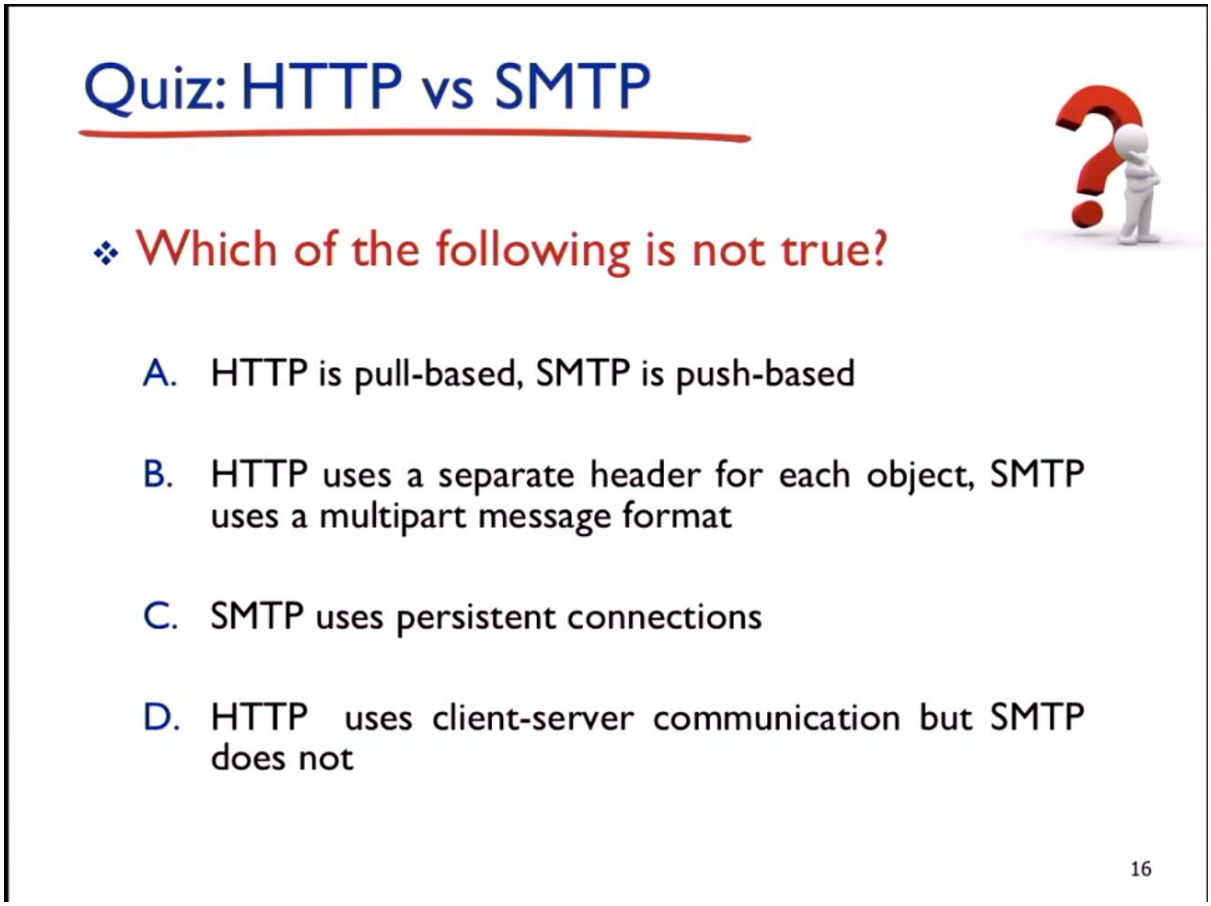
If SMTP only allows 7-bit ascii how do we send pictures/videos/files via email

- We encode these objects as 7-bit ASCII

Mail access protocols

- SMTP -> delivery/storage to receiver's server
- Mail access protocol retrieval from server
  - POP: post office protocol: authorization download → cannot create folders and sort emails
  - IMAP: Internet Mail Access Protocol: more features including manipulation of stores messages on server

- HTTP(S): Gmail,Yahoo! Mail etc.



D SMTP is still client server based

DNS – Domain Name System

We have name + student id, all records use student id and all backend work is done with student id however our name is an alias!

In web communication is based on IP addresses, 32 bit

If we were asked to get info from google, we use google's alias rather than its ip address

DNS

- Distributed database implemented in hierarchy of many name servers
- Application layer protocol

DNS takes a name and will do a query to find the corresponding IP address either via recursive or iterative query

Any protocol which requires naming will refer to DNS! Nslookup + dig will allow us to access DNS directly

Hosts.txt beginning of DNS

Maintained by the Stanford research institute

Changes were submitted to SRI by email

New versions of hosts.txt periodically FTP'd from SRI

An adiminstrator could pick names at their discretion

As the internet grew this system broke down as SRI couldn't handle the load

Names were not unique and hosts had inaccurate copies of hosts.txt

DNS CAME TO THE RESCUE DADDY DNS

Why not centralize DNS?

- Single point of failure
- Traffic volume
- Maintenance
- Distant centralized database
- ITS NOT SCALEABLE

What is DNS?

- Hostname to ip address translation
- Aliasiang for host → canonical, alias names
- Mail server aliasing
- Load distribution
    - Replicated web servers → many ip addresses correspond to one name
    - Content distribution networks (CDN): use ip address of requesting host to find best suitable server, closest least loded etc

GOALS

- No naming conflicts → UNIQUENESS → SET
- Scaleable
    - Many names
    - Frequently upodated
- Distributed and autonomous administration
    - Ability to update my own machines names
    - Don't have to track everybody's updates
- Highly available
- Fast lookups

This is achieved through hierarchy

- Three intertwined heirarchies
    - Namespace
    - Administrated
    - Storage

Heirarchial namespace

- Top is root → NEVER TO BE NAMED
- Below that TLD → top level domain names, eg. Edu com gov mil org
- Also CSTLD → country specific TLDS , eu au us fr uk
- Intermediate name servers → extras
- When writing urls go from bottom up no root included

Domains are sub trees

Name is leaf->root path

Depth of tree is arbitrary, limit 128

Naming conflicts only rise within own domainds!

Heirarchial administration

Authoritative name server (NS) berkley is administrator

All sub domains managed by berkeley

Below that eecs has its own zone where it is the authority

EACH NAME SPACE HAS AUTHORITY OVER ITS OWN ZONE!!!

A zone corresponds to an administrative authority that is responsible for that portion of the hierarchy

UCB controls names: *.berkley.edu and *.sims.berkeley.edu

EECS controls names *.eecs.berkeley.edu

Each server stores a small subset of the total DNS database

An authorative DNS server stores resource records for all DNS names in the domain it has authority for

Each server needs to know other servers that are responsible for the other portions of the hierarchy

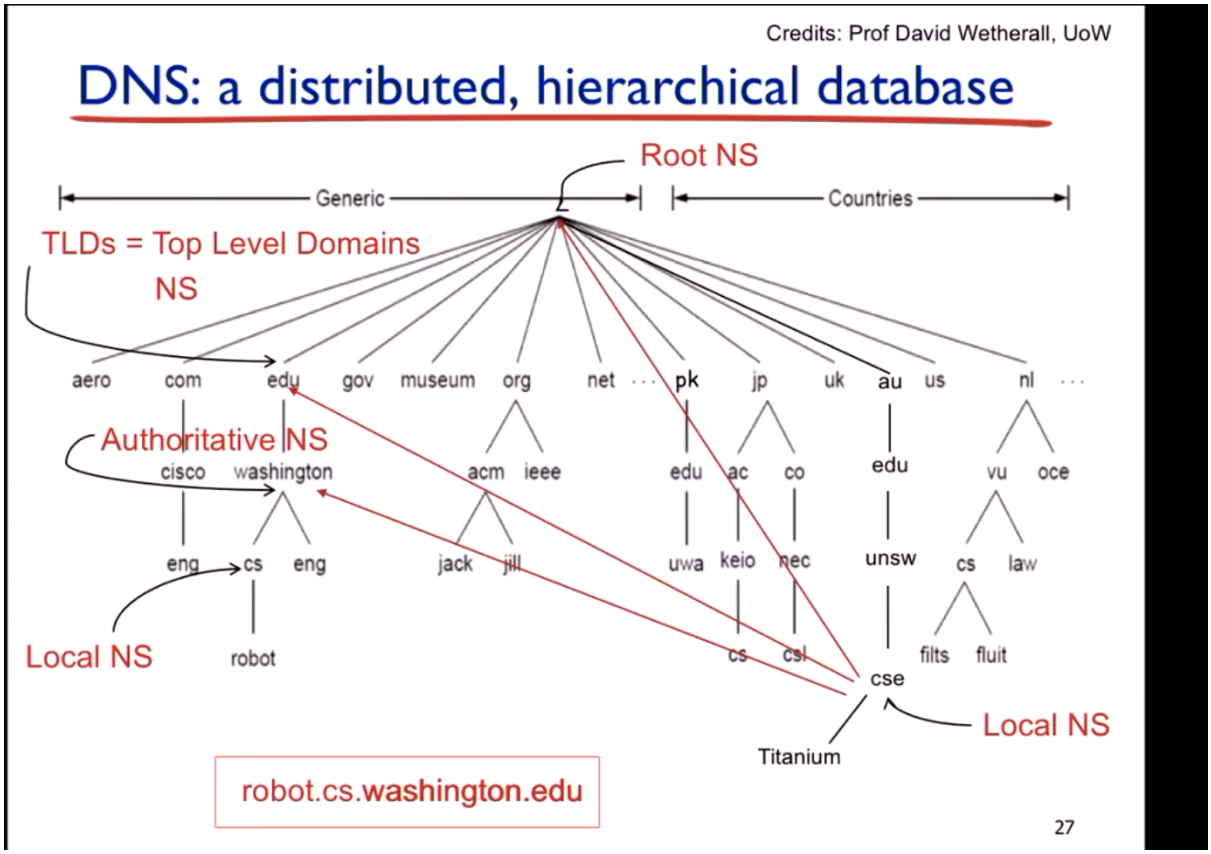
- Every server knows the root
- Root server knows about all top-level domains

First on dns query for any name server, we have name we need IP eg. Robot.cs.washington.edu

- ➔ Will ask local name server to check cache for which ip address we looking for
- ➔ If no reply we jump straight to root we do not go up the tree
- ➔ Root will check the full url, it will start sending it on the right path
- ➔ Root will send it to .edu which is the highest level under root with right extension
- ➔ Cache the entry so next time can jump straight to edu
- ➔ Cache on the way down!!
- ➔ Now .edu will send you to .washington
- ➔ Cache the entry so next time can jump straight to Washington
- ➔ Washington is the authoritative Name server
- ➔ Washington will have the mapping of all IP's under Washington and will return ip

Caching is important, it will cache robot.cs.washington.edu and now any other service requesting it will have result cached

THIS IS AN ITERATIVE QUERY WE RECOMMEND JOB IS ON CLIENT



In iterative search, only we have the cached answer since the answer travels through us

In a recursive search, the root server and .edu has the answer cached!!

Is there a single root name server? → NOOO

13 root servers!! Managed by 12 organisations geographically distributed

In reality we are distributed all across the globe

Unicasting/broadcasting, 1 source many users = broadcaster

Multicasting, send message to a subrgroup of entire course

Any casting, get hold of any member of a specific group! <← used in DNS finds the closest root server for you

All addresses mapped hard coded so it will find the first one closest to you

TLD authoritative servers

Must know how to differentiate between local name servers and authoritative servers or root name server and top level domain service and know their roles!

Top-level domain (TLD) servers:

- Responsible for
  - .com
  - .net

    - .edu
    - .aero
    - .jobs
    - .mesuems
    - All country domains
        - Uk
        - Fr
        - Ca
        - Jp
        - Au
        - Network solutions maintains servers for .com TLD
    - Educause for .edu TLD
- Authoritative DNS
    - Organization's own DNS server(s) providing authoritative host name to ip mappings for organization eg. Google
    - Can be maintained by organization or service provider
- Advantage vs disadvantage of using our own local name server vs google name server

DNS records what is stored in a DNS record

DNS – distributed database storing resource records (RR)

RR format: name, value, type, ttl

TTL → time to live refers to how long to keep a packet alive before revalidating, this is to stop a data packet from circulating indefinitely.

Type = a → ipv4 address/aaaa → ipv6 address

- Name is host name
- Value is ip address

Type = NS → name server

- Name is domain
- Value is hostname of authoritative name server for domain

Type = CNAME → canonical name

- Name is alias for some canonical name
- Value is the canonical name

Type = MX → mail server

- Value is name of mailserver associated with name

Type – PTR → reverse query

- Stores reverse DNS entries

DNS protocol and messages

- Query and reply messages both with same format

    - Message header
        - Identification, a 16-bit number for query, reply to query uses same number
        - Flags
            - Query or reply
            - Recursion desired
            - Recursion available
            - Authoritative reply

Inserting records into DNS

Example new start-up networkland

Register the domain name at DNS registrar e.g. Network solutions

Provide names, ip addresses of authoritative NS both primary and secondary

Registrar inserts two RRs into .com TLD server

- Networkutopia.com, dns1.network.utopia.com, NS
- Dns1.networkutopia.com, 212.212.212.1, A

Create authoritative server type A record for networklank and type MX record for networkload.com

Where do we insert the type A and type MX Records??

- Store in our own name server!!
- Since we are authority now

Reliability in DNS

- DNS servers are replicated
    - Name service available if atleast one version is up
    - Queries can be load-balanced between versions for performance
- Usually UDP used for queries (sometimes TCP not likely)
    - Need reliability: must implement this on top of UDP
    - Spec supports TCP but its now likely implemented
    - We try alternate servers on time out, however there is exponential backoff when retrying same servers
- Same identifier for all queries
    - We don't care which server responds or which version we just want response
    - All replies assumed same

DNS provides indirection

Addresses can change underneath

- Move www.abc.com to 4.125.91.21
- Humans/applications/users should be unaffected
- When changed we ask root, and it may have a cached copy of old ip, so we apply TTL to make sure new cached copies are validated, when cache empty we go down the chain of authority to the host we are tying to reach

Name could map to multiple ip addresses

- This allows for better load balancing
- Reduces the latency by picking closest servers

Multiple names for same address

E.g. Many services use mail, web, file transfer on same machine

And aliases allow for easy identification

Reverse DNS

- Given ip address we can look for domain name, -x option
- Special PTR record types used for reverse DNS look ups
- Reverse DNS is used in tools such as
    - Traceroute//ping
    - Receiving trace header field in SMTP
    - SMTP servers for validating ip addresses of originating servers
    - Internet forums tracking users
    - System logging/monitoring tools
    - Used in load balancing servers/content distribution to determine location of requester
    - Firewall, when a packet comes in we do reverse DNS to check name of app

Do you trust your DNS ServeR?

Alternate DNS → can be redirected unwillingly to a different website through dns this is flat out censorship

NX domain = not accessible domain

- Censorship
- Logging
    - IP addresses, websited visited, geolocation data
    - Google DNS

DNS is powerful but least protected, google DNS can log everything

ATTACKING DNS

- Ddos attacks
    - Bombard root servers with traffic
    - Not successful to date
    - Traffic filtering
    - Local DNS servers cache ips of TLD servers allowing root servers to be bypassed
- Bombard TLD server then
    - A lot more dangerous, there are a lot and less protected
- Redirect attacks
    - Man in the middle intercepting queries
    - DNS poisoning which sends bogus replies to dns server which caches
- Explot DNS for DDoS

    - Send queries with spoofed source address, the target IP
    - Requires amplification
- Look up DNS tunnelling and exfiltration

DNS cache poisoning is when you force fraudulent cache for another service, suppose you own www.trolled.com and when you perform your query one of your authority sections is google.com, google.com will be caches under your machines ip address!

The solution would be to not allow DNS servers to cache IP address mappings they have no authority over!

PEER to PEER + CDNS

Pure P2P architecture

- No always on server
- Arbitrary end systems directly communicate
- Peers are intermittently connected and change ip addresses

VOIP -> coice over IP

File dirtribution: client server cs P2P

How much time is needed to distribute file size F from one server to N peers

Peer upload/download capacity is a limited resource

From client/server

Server transmission

- Must send (upload) N file copies
- Time to send one copy: F/Us (Us = upload speed)
- Time to send N copies: NF/Us
- Client each client must download a file copy
- Dmin = min client download rate
- Client download time = F/dmin

Time to distribute file size F to N clients using client server approach

DownloadC-s >= max{NF/Us,F/dmin}

If N increases, delay increases linearly in N → more client = more delay

In case of P2P

Server transmission

- Must upload atleast one copy
- Time to send one copy : F/Us

Client

- Each client must download file copy
- Time to download = F/dmin

Clients as aggregate must download NF bits

- Max upload rate (limiting max download rate) = Us + sum of all users who have uploaded

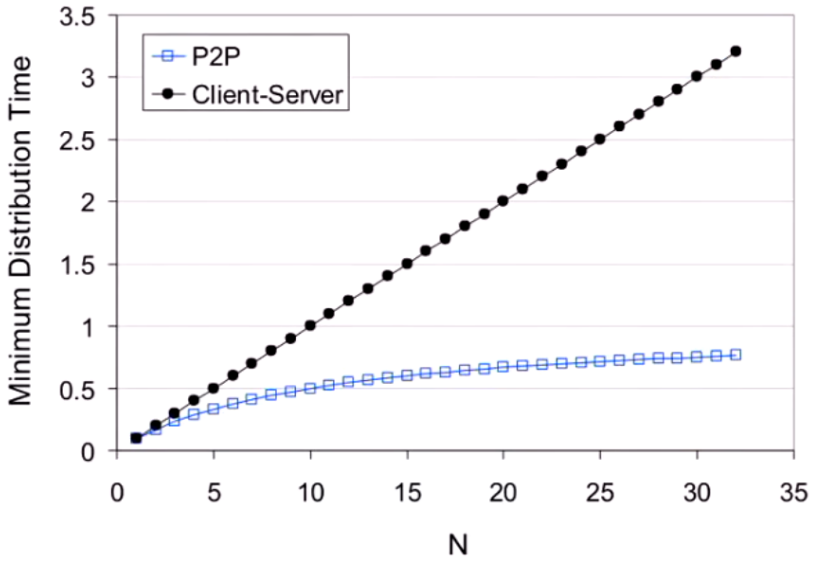
So now time to distribute file size F to N clients using peer to peer approach

DownloadP2P >= max{F/Us,F/dmin,NF/(Us + upload of all users involved)

It is still increasing linearly in N, however the more clients we add in, the more server capacity we add

Client-server vs. P2P: example

client upload rate = u, F/u = 1 hour, $u_s = 10u$



7

More users = more upload capacity means we can download faster

BitTorrent example

- File divided into 256kb chunks
- Peers in torrents send/receive file chunks

Tracker – tracks peers participating in torrent

Torrent – group of peers exchanging chunks of a file

.torrent files

Contains the address of trackers for the file, where you can find other peers

Contains a list of file chunks and their cryptographic hashes, this ensured that chunks are not modified

Peers joining torrent initially has no chunks, however accumulate them over time

Peer joining registers with tracker to get list of peers and connects to a subset of peers (neighbours)

While downloading peers upload chunks to other peers

Peer may change peers with whom it exchange chunks

Churn – peers come and go

Once a peer has entire file it may selfishly leave or altruistically remain in torrent, lets be real who the fuck seeds

Requesting, sending file chunks

Requesting chunks

- At any given time, different peers have different subsets of file chunks
- Periodically, alice asks each peer for list of chunks that they have
- Alice requests missing chunks from peers, rarest first, as it ensures you get all the packets necessary download entire file, and allows you to seed out rarest chunk

Sending chunks: tit-for-tat

- Alice sends chunks to those four peers currently sending her chunks at highest rate
    - Other peers are choked by alice
    - Re-evaluate top 4 every 10s
    - Whoever gives more gets more only top 4 get
- Every 30 seconds randomly select another peer, start sending chunks
- Optimistically unchoke this peer
- Newly chosen peer may join top 4

Free-riding quiz

Suppose todd joins a bittorrent but he doesn't want to upload any data to other peers, and todd says he can receive a complete copy of the file that is shared by the swarm, is his claim possible?

Answer: possible however it would take a very long time, he is just waiting for optimistical unchoke, since he is not giving anything he will only be receiving through the optimitical unchoke

Getting rid of the server/tracker

Distribute the tracker information using a distributed Hash Table

A DHT is a lookup structure

- Maps keys to an arbitrary value
- Works like a hash table

Distributed Hash Table (DHT)

- DHT a distributed P2P database
- Database has key,value pairs
- Distribute the pairs over the millions of peers
- A peer queries DHT with key and DHT returns the values that match the key
- Peers can also inert their key/value pairs

Challenges

- How do we assign key/value pairs to nodes
    - Convert each key to an integer
    - Assign an integer to each peer
    - Put key/value pair in the peer that is closest to the key
    - Assign key to the peer that has closest id
    - Closest is the immediate successor of the key
- CIRCULAR DHT
    - Each peer only aware of immediate successor and predecessor
    - Overlay network
    - Can only move clockwise
    - Create shortcuts
        - Each peer keeps track of ip addresses of predecessor, successor and shortcuts
        - Reduces load from 6 to 2 messages
        - Possible to design shortcuts so we get O(logN) neighbours with O(logN) messages sent in query
- How do we find them again quickly?
- What happens when nodes join/leave
    - Peer churn,
    - When a peer leaves abruptly we need to reset the DHT

We need to know how a DHT works, we need to know how does it maintain randomness and that if there are N trackers chance of you chosen is 1/N

DHT makes it so that it is very hard to pinpoint a service/torrent to a specific tracker, instead of making it based on a certain user in control of management, each peer can now be the tracker all random

Video streaming and CDNS (content distributor network)

Video traffic: major consumer of internet bandwidth

Youtube has ~1 billion users,

Netflix has ~75 million Netflix users

Our main challenge is scalability with that many users, note a megaserver is not scalable a single point of failure is never scalable

Another challenge is heterogenety, making products suited to the users

Solution is having a distributed application-level infrastructure

Multimedia video

Video sequency of images displayed at constant rate

24 frames a second

Digitial image is an array of pixels each pixel represented by bits

Coding: use redundancy within and between images to decrease the number of bits used to encode image

- Spatial is within image
- Temporal, from one image to next

CBR – constant bit rate

Video encoding at fixed rate

VBR – variable bit rate

Video encoding rates as amount of spatial, and or temporal coding changes

Examples

MPEG 1, 1.5mbps

- MPEG2 3-6mvps
- MPEG4 often used in internet <1mbps

Streaming stored video:

A video server has a server of stored videos.

Streaming multimedia: DASH

DASH: Dynamic, Adaptive Streaming over HTTP

Server:

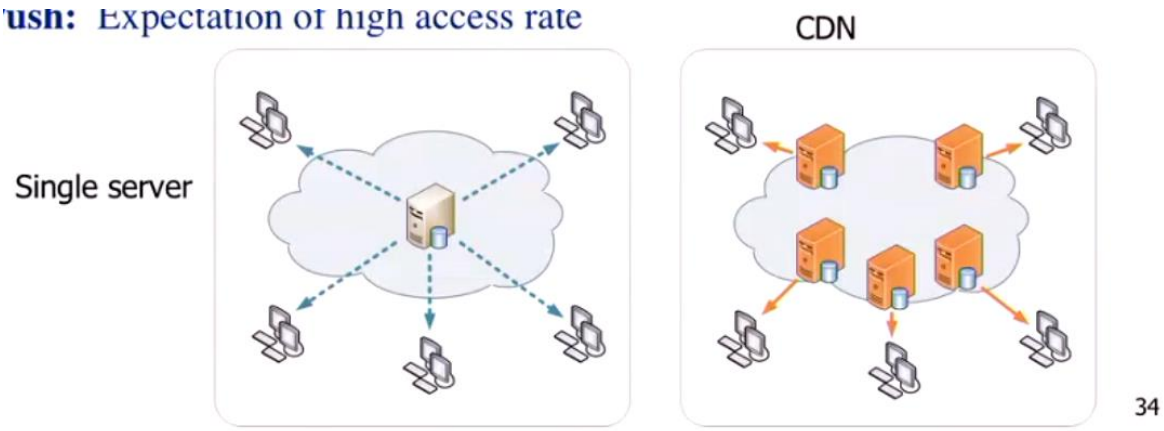
- Divides video file into multiple chunks
- Each chunk stored, encoded at different rates
- Manifest the file, providing urls for different chunks

Client:

- Periodically measures server-to-client bandwidth
- Consulting manifest, requests one chunk at a time
    - Chooses maximum coding rate suistainable given at current bandwidth
    - Can choose different coding rates at different points in time (depending on available bandwidth)
    - Intelligence at client determines when to request chunk so that buffer starvation or overflow doesn't occur, determines what encoding rate to request at (the higher the quality the more bandwith needed), and where to request chunk. Can request from url server that is close to client or has highest available bandwidth

CDN – content distribution network

A single server will never scale up



Caching and replication as a service amortise cost of infrastructure,

Goal is to bring the content closer to the user

To do this we can have large-scale distributed storage infrastructure (Servers) administed by one entity

Combination of pull (caching) and push (replication)

- Pull directs result of clients request
- Push expectation of high access rate (despacito needs to be worldwide while a content creator only seen in France can stick in France)

To check if a service are hosted by CDNs use dig and check for akami which is used for content distribution

2 options for CDN

Enter deep: put the CDN node within networks isp

- Close to users
- Used by akami at thousands of locations

Bring home

- Smaller number (10's) of larger clusters in POPs (point of presence) near access network (at internet exchange point IXP)
- Used by limelight

Enter deep

➔ Deploy node at isp

Bring home

➔ Deploy node at ixp

Enter deep gives least network delay since its already within ISP.

Google uses a mix of bring home + enter deep

Isp is 1 hop away

IXP might be multiple hops

CDNS store copies of content at CDN nodes.

Eg. Netflix stores copies of madmen

Subscriber requests content from CDN

- Directed to nearby copy retrieves content

This is an over the top approach where we disregard the underlying network that supports this network

Transport Layer

We are now moving down a layer

Network layer provides path from one end host to another, however comes with

- No guaranteed path
- No reliablility
- No guaranteed transfer rates
- Only guarantees to give best effort to move network across

Transport layer only works from an end to end mechanism, only comes to play in end system, routers do not contain transport layer, only end system acknowledges

Transport services and protocols

- Provides logical communication between app processes running on different hosts
- Transport protocols run in end systems
    - Send side: breaks app messages into segmenets that passes to network layer
    - Receive side reassembles segments into messages, passes to application layer
    - Exports services to application that network layer does nto provide

Why a transport layer?

- Reliability needed!
- Multiplexing/demultiplexing
    - Makes communication between processes at hosts
    - Transport layer can identify the correct process for request through sockets

Multiplexing/demultiplexing

Multiplexing at sender:

- Handle data from multiple sockets, add transport header (later used for demultiplexing)
- Allows for identification of which process is running

Demultiplexing at receiver

- Use header info to deliver received segments to correct socket
- Allows mapping for application/hosts

Connectionless demultiplexing

- Recall socket has host qand port number
- When creating datagram to send to UDP socket, must specify destination ip address and destination port number,
- UDP only cares for destination

-When host receives UDP segment

- Checks destination port # and directs segment to socket with that port number

Create the sockets, based on UDP.

- Only differentiate socket by port number not IP
- Multiple machines can communicate to same socket
- Server only looks at destination port number, this is only in UDP

Connection oriented demux

- TCP socket identified by a 4-tuple
    - Source IP
    - Source port number
    - Destination IP
    - Destination port number
- Demultiplex
    - Receiver uses all four values to direct segment to appropriate socket
- Server host may support many simultaneous tcp sockets
    - Each socket identified by its own 4-tuple
- Web server shave different sockets for each connecting client

In UDP identification based only on port number

In TCP identification is based on the 4-tuple

As soon as TCP handshake is over, server will create its own connection under the server

Connect call will create a separate connection for you under server process

All process go to same ip address, however each service has different port number. Destination port number = 80 are demultiplexed to different sockets

Tcp creates different sockets to communicate with each client. And the different sockets have the same port #

Servers wait at open ports for client requests

Hackers often perform port scans to determine open, closed and unreachable ports on candidate victims

Several ports are well known

- <1024 are reserved for well known apps
- Apps also use known ports
    - MSQL uses prot 1434 UDP

    - Sun Network File System (NFS) 2049 (tcp/udp)
- Hackers can exploit known flaws with these known apps

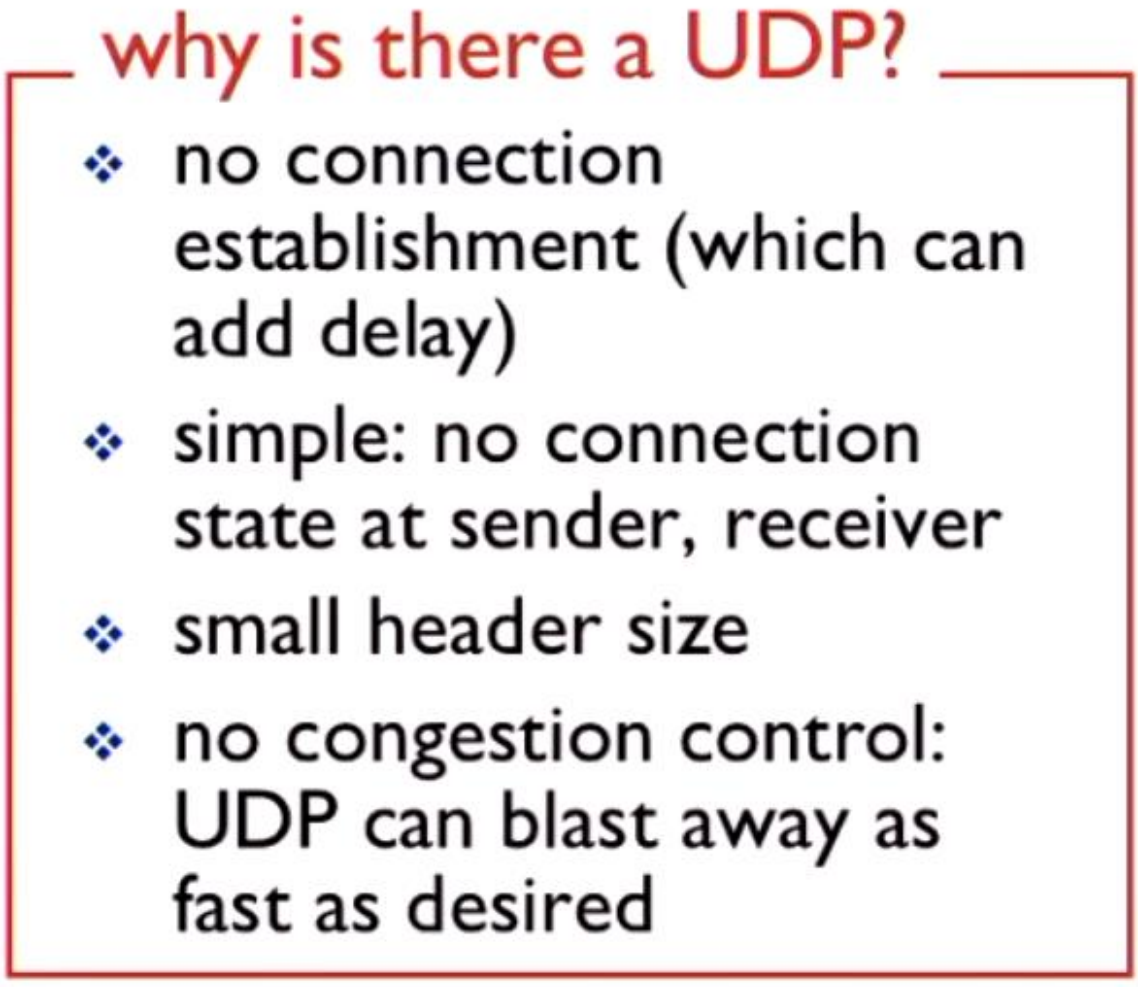
NMAp,superscan can perform port scans

UDP -> user datagram protocol

- No frills, bare bones internet protocol
- Best effort service

UDP header only requires

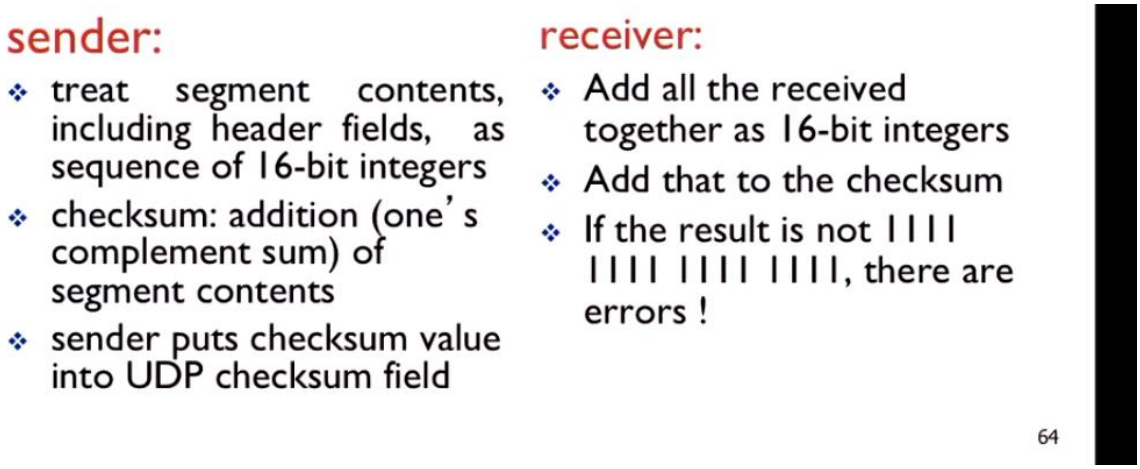
- Source port – 2 bytes
- Dest port – 2 bytes
- Length – 2 bytes
- Checksum – 2 bytes
- 8 byte total for header in UDP



UDP has a lot lower overhead than TCP, lower delay

UDP checksum

- Detect bit errors in transmitted segment
    - Router memory errors
    - Driver bugs
    - Electromagnetic interference



-
- Udp steals ip checksum method → layer violation
- But header contains the checksum itself

Mid sem 25 question, all MCQ 1 answer only

Internet checksum example

Whenever we get overflow, we take it and add it to the least significant bit, then at the end we take the one's complement of our number to get the check sum

UDP applications 8 byte header only allows multiplexing + de-multiplexing with port value

- Quick request/response (DNS, DHCP)
- Network management (SNMP)
- Routing updates (RIP)
- Voice/video chat
- Gaming especially with FPS

If udp receives a header and check-sum fails, it will simply just drop the packet. UDP offers no reliability and is a stateless protocol

Error correction is unnecessary in UDP (periodic messages)

Udp allows us to send out data even if server is down, just no reply/no error

Principles of reliable data transfer

Reliable transport

All the bad things best effort can do is

- A packet is corrupted (bit errors)
- A packet is lost
- A packet is delayed
- Packets are re-ordered
- A packet is duplicated

Reasons bit errors can occur in a packet

- Network interface card is corrupted -> send out rubbish data
- Or maybe router stores packet in buffer in queue, and when it is being read leads to bit errors

Reasons packets can be re-ordered

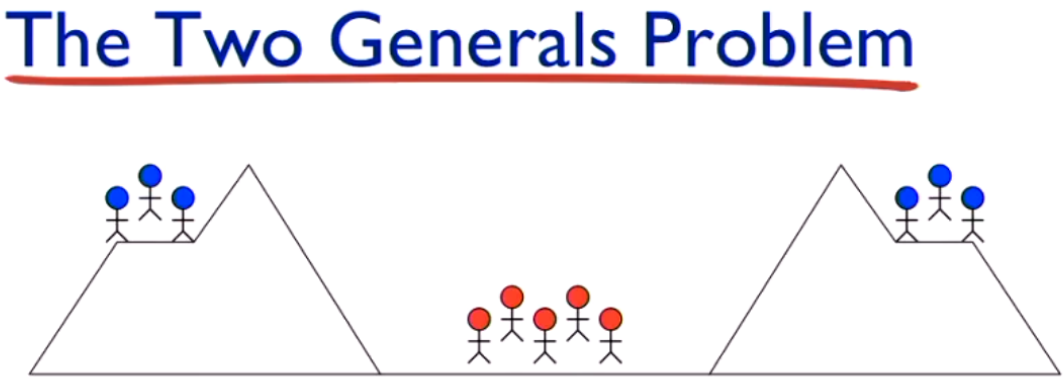
- This is dependant on the path the packet takes to reach the host, if paths are congested leads to this

Reasons a packet can be duplicated in network

- Following reliable protocol, if ack doesn't come back in time and packet is re-transmitted, however it comes back later, we get both packets

The two general problems

- Two army divions surround an enemy
- Each army is led by a single general
- Both must agree when to simultaneously attack
- If either side attacks alone there is defeat
- Generals can only communicate via messangers, and messangers may be captured



Transport layer is built on IP layer, which is unreliable, a best effort service, transport layer can provide reliability on its own building on top of the unreliable ip layer, we can implement our own reliability. We are using UDP but implementing the reliability of TCP.

tCP in a nutshell

- Hey bro I got this folder I wana transfer to XXX.com

- TCP goes, bro give it to me and don't worry about a thing
- All your packets arrive in order, with no bit corruption and will be guaranteed to arrive
- TCP is a reliable protocol

TCP is built on IP which is an unreliable best effort service

When application sends data it will assume it is a reliable protocol, so it sends an RDT (reliable data transfer), when we deal with ip then it assumes unreliable so does unreliable sends

Reliable data transfer basics

- Incrementally develop sender, receiver sides of reliable data transfer protocol (RDT)
- - consider only unidirectional data transfer
- Control info flows in both directions
- Channel will not effect ordering of packets

1 guy sends, other guy only allowed to send back acknowledgements.

RDT 1.0 reliable transfer over a reliable channel

For a channel to be perfectly reliable we require

- No bit errors
- No loss of packets

However the transport layer does nothing since the channel is assumed reliable

Rdt 2.0 channel with bit errors

- Underlying channel may flip bits in packet
    - Checksum to detect bit errors
- Receiver has to provide feedback ACKNOWLEDGEMENT BICH
- We need feedback to know current status of state
- To recover from this error we can simply re-transmit the packet, in human nature if we mess up in conversation we repeat ourselves
- Negative acknowledgements (NAKs) receiver tells sender the packet had errors
- Acknowledgement assumes packet is ok receiver says it's a okay!
- Retransmit on naks

Now we have 2 extra measures for reliability

1. Set checksum to check for errors
2. Re-transmission of packet

RDT 2.0 has a fatal flaw which can be really bad

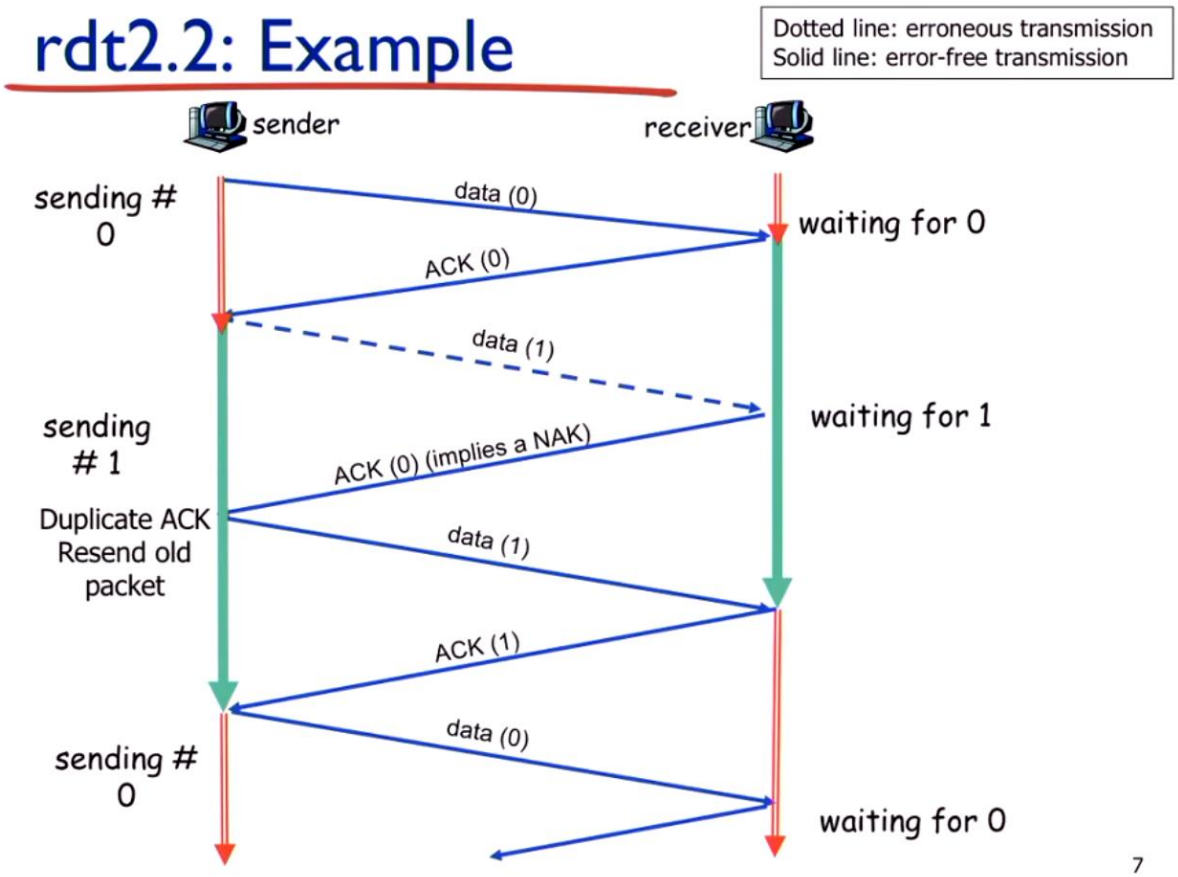
- Ack/nack response can also corrupt as above! Same chance it will corrupt!
    - Sender won't know what happened at receiver
    - Can't just retransmit possibly a duplicate, a corrupted ack could be re-transmit!
    - Put a checksum on the ack/nack, now the sender can know if the feedback is corrupted!

- Now what happens if the checksum on ack/nack fails?
    - Sender re-transmit packet if any corruption to feedback
    - Sender adds sequence number to each packet
    - Receiver discared duplicates, you can make a set where u do not add any duplicate sequence number! → RDT2.1
- This is the stop and wait protocol or alternating bit protocol
- We send a packet
- Wait for ack/nack
- And continue
- Keep transmitting till ack before we go to next packet
- Sequential protocol, we cant send bursts of packets!

RDT2.2 a nak free protocol

- Same functionality as rdt2.1 however only uses ACKS
- Instead of nak, receiver sends ack for last packet received okay
    - Receiver must explicity include seq# of packet being acked
- Duplicate ack at sender results in same action as nak, retransmit current packet

Basically now the receiver puts a sequence number,



Now if we send packet 1 and we get back ack0 we know that it was a nack since we didn't get ack1, so we transmit

All current versions of rdt assume no packet loss only assuming bit error

Rdt3.0 channels with errors AND loss

New assumption:

Underlying channel can also loose packets

- Both data/acknowledgements
- Checksum, seq#, retransmissions are helpful but we need more!!

Packets can be lost through

- Queue being full in router
- Link failure
- Network failure
- Routier failure → ip checksum failure

We need a timer!

Sender will wait a reasonable amount of time for acknowledgement

- Retransmits if no ACK within timeout period
- If packet is just delayed, retransmission is simply a dupliocate who cares
- Receiver must specifically specify seq# of packet being ack'd
- Requires a countdown timer that resets each send

Rdt 3.0 doesn't cater for packet re-ordering

Stop and wait – best care t = RTT + L/R

Rdt 3.0 has horrible performance

Performance of rdt3.0

- rdt3.0 is correct, but performance stinks
- e.g.: 1 Gbps link, 8000 bit packet and 30msec RTT:

$$D_{trans} = \frac{L}{R} = \frac{8000\ bits}{10^9\ bits/sec} = 8\ microsecs$$

- U_{sender}: *utilization* – fraction of time sender busy sending

$$U_{sender} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

- RTT=30 msec, 1KB pkt every 30.008 msec: 33kB/sec thruput over 1 Gbps link

- Network protocol limits use of physical resources!

Pipelined protocols

Pipelining: sender allows multiple, “in-flight”, yet to be acknowledge packets, send n-packets back to back, and wait for acknowledge for all N packets.

- Range of sequence numbers must be increased
- Buffering at sender and/or received
- Two generic forms of pipelined protocols: go-back-n, selective repeat

Stop and wait, 1 packet at a time super bad efficiency imagine 50000 packets

Pipelining can increase efficiency by massive amount N depending on burst of packets

We want (L/R)/(RTT+(L/R)) as close to 1 as possible 100% efficiency

Go-Back-N

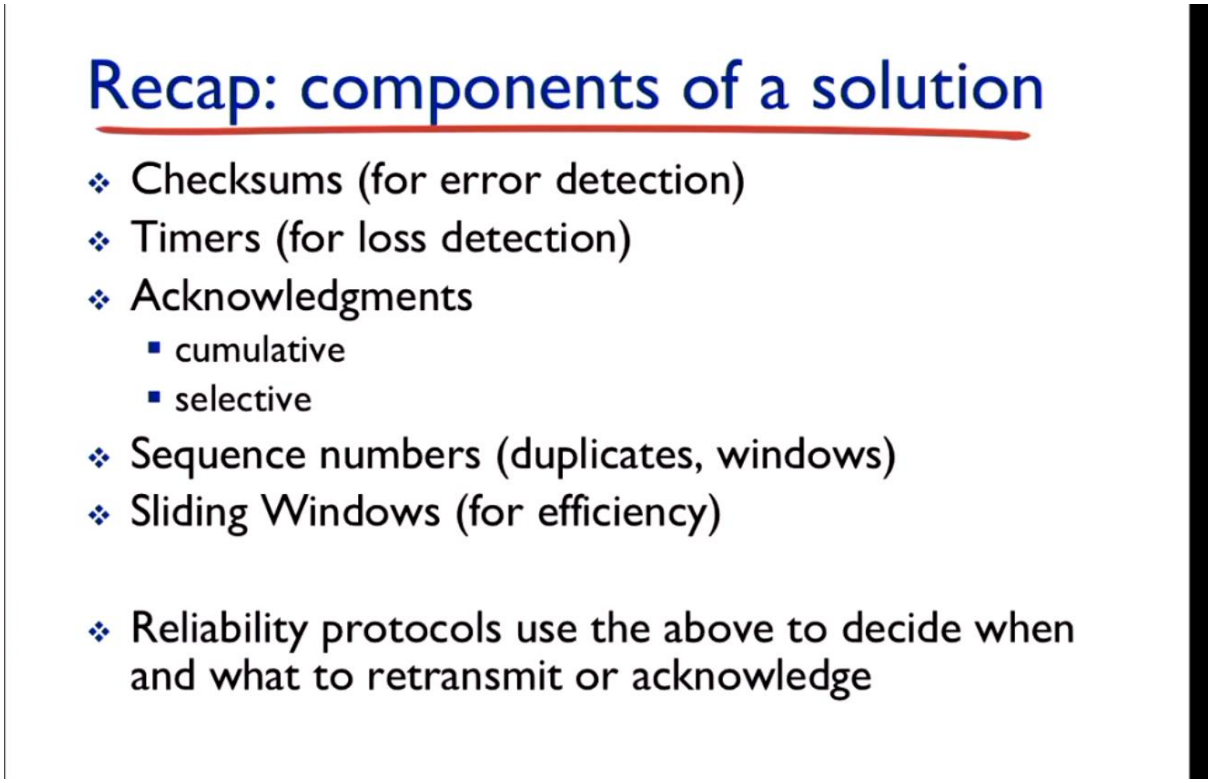
- Sender can have up to N unacked packets in pipeline
- Sender has only a single timer for oldest unacked packet, when timer expires it will re-transmit all unacked packets, timer will be on first packet in window, so packets are sent in order guaranteed! Drops all out of order packets
- No buffer at receiver, out of order packets are dropped
- Receiver only sends cumulative ack, doesn’t ack new packet if there’s a gap

Selective repeat:

- Sender can have up to N unacked packets in pipeline
- Sender has a timer on each unacked packet, when timer expires, retransmit only that unacked packet, → leads to mis-ordering!!
- Buffer at receiver, can accept the out of order packets, and will insert in order
- Receiver sends individual ack for each packet

GBN → sliding window of size N super ez

For selective repeat – window size must be less than or equal to half the size of the sequence number space



Components of a solution for reliable transport

- Checksum for bit error detection
- Timers for loss detection
- Acknowledgements in two forms
    - Cumulative GBN
    - Selective
- Sequence number for duplicates and windows
- Sliding windows for efficiency
    - GBN
    - Selective repeat

What does my boi TCP do

- Checksum exact same! Uses ip checksum. Same as udp, checksum bi-directional for acks aswell
- Sequence numbers are byte offsets!
    - Tcp numbers based on the size of packets and offset, similar to array in memory,
    - Eg. Packet size 100 10 packets,
    - Sequence 1 byte 0, sequence 2 byte 100,
    - We refer to our segmenet by the starting byte
- Segment sent when
    - Segment is full → max MSS (maximum segment size)
    - Not full, but time out while adding data to segment
- Receiver sends cumulative acks just like GBN opop

TCP segment size

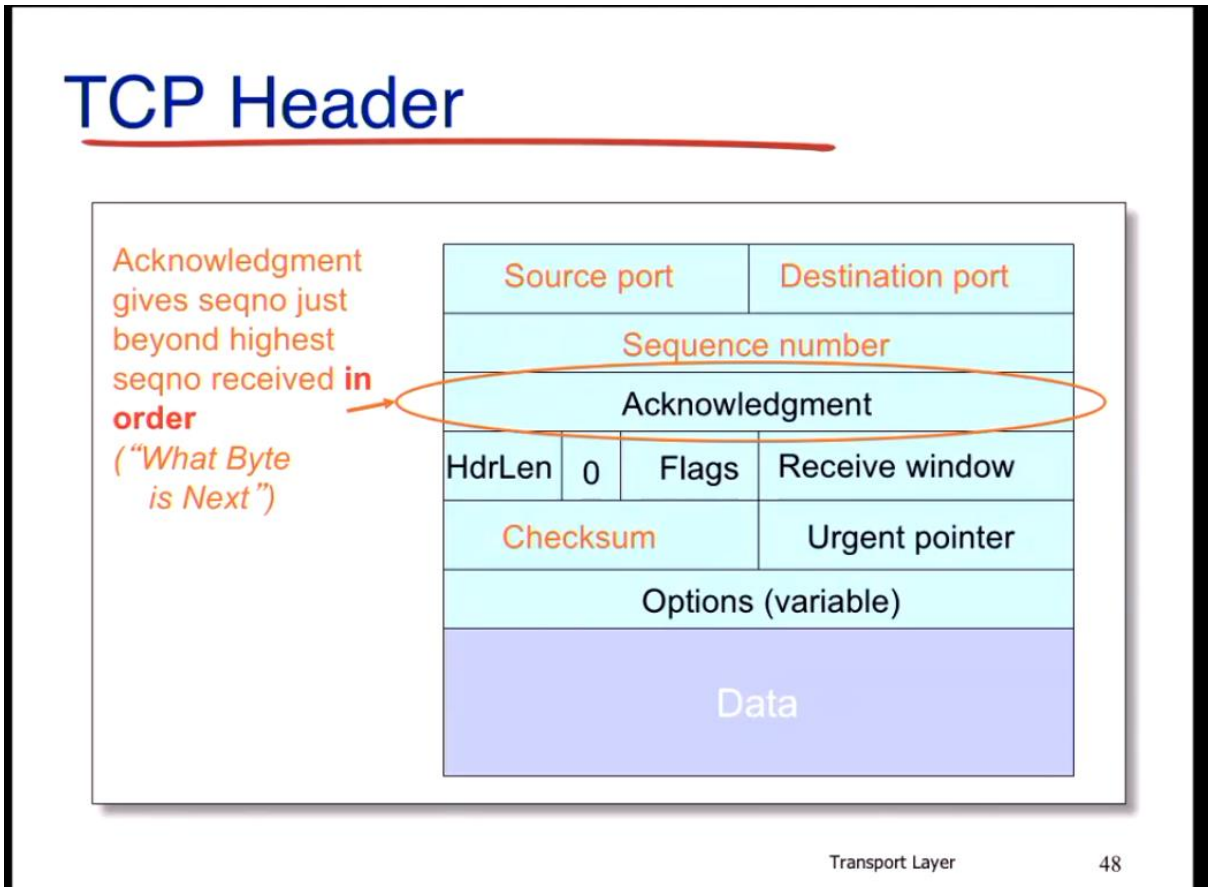
MTU (maximum transmission unit) is the total amount of data the ip layer can hand over to ethernet

(IP sitting ontop of datalink layer). If MTU is 1500 bytes, we need to take off amount for 20 byte ip header which has TCP data, then we also need to take 20 bytes off from the TCP header, then maximum segment size = 1500-20-20 = 1460 bytes

Sequence numbers

- Byte stream number off first byte in segment's data
- Sequence number = ISN+k, where ISN = initial sequence number and k = size of segment
- Now ack sequence number = next expected byte = sequence number + length of data

ACK in tcp just like GBN is the last byte received correctly



TCP Header

Piggybacking

- With piggybacking we can send a response + ack in the same message
- In M4 think of two speeding cars 1 is response 1 is ack

Receiver can buffer out of sequence packets like selective repeat

TCP is a hybrid of GBN (cumulative acks) and selective repeat (buffer out of order sequence)

Sender maintains a single retransmission timer just like GBN, and retransmits on timeout doesn't re-transmit things in the buffer

TCP round trip time, timeout

We need to adapt our timer to the network conditions, if it's too high we get long delays, if its too short, too many re-transmtis, we don't want a static timeout, we want a dynamic timeout system.

Longer than RTT which varies

estimatedRTT = (1-a)*EstimatedRTT + a*sampleRTT

- A has typical value of 0.125
- Our estimated RTT is already calculated
- Sample RTT is one sample sent out

Exponentially average, weight is assigned to history really similar to raising wam cant just raise in bursts

Estimated RTT is not enough we need more, a safety margin

DEVRTT = (1-B)*DEVrtt + B *(sampleRTT-estimatedRTT)

Typically B = 0.25

Timeout interval = estimatedRTT + 4*DevRTT ← safety margin