What is the Internet

- **Hosts** - end systems, runs network apps like computers and phones
- **Communication Links** - wired(fiber, copper) and wireless(radio, satellite), transmission rate = **bandwidth**
- **Packet Switches** - **Routers** and **Switches**, information organised into packets of data

Hosts uses communication links to talk to packet switches which uses communication links to talk to hosts. Internet is a network of networks.

Network Edge

- **Network Edge** - clients(laptops and phones) and servers, can be shared or dedicated
- **Access Networks** - wired or wireless communication links (home network)
- **Network Core** - interconnected routes, network of networks (essentially the entire internet)

Network Core

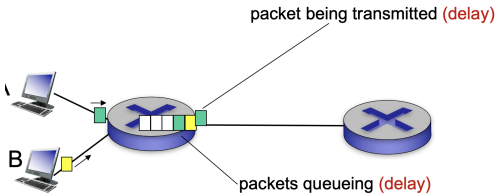
Mesh of interconnected routers.
Packet-Switching - host creates packets (chunks of data), forwards packets from one router to the next, ensure that data send it correct and complete. Packets of size L *bits*, transmission rate of R *bits/sec*. **Link transmission rate = link capacity = link bandwidth**, is the transmission rate for 1 link, may not be the same for others.
$Delay = \frac{L}{R}$

- **Advantage** - Efficient use of links
- **Disadvantage** - Potential congestion

Packet Switching - Store-And-Forward

Entire packet has to arrive before it can be transmitted. Router checks completeness and validity on arrival.

Packet Switching - Queuing delay, loss



Input rate > Output rate, creates traffic jam, packets have to queue and wait to be transmitted. Packets arriving to a full queue will be lost. Lost packets can be re-transmitted or not. Total delay time can be calculated by :
$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$, where

- d_{proc} - time for node to check errors and determine output
- d_{queue} - time waiting at output link for transmission, depends on congestion
- d_{trans} - Time taken to push one packet onto the link
- d_{prop} - Time taken for 1 bit to travel through the link, $\frac{length_{link}}{speed_{propagation}(m/s)}$

Functions

- **Routing** - determine route to take to deliver packets to destination, using routing algorithms
- **Forwarding** - Using a **forwarding table**(maps values to output link number). Table created by routing algorithm

Circuit Switching

Divide link into circuits for data transfer. Something like exclusive lane.

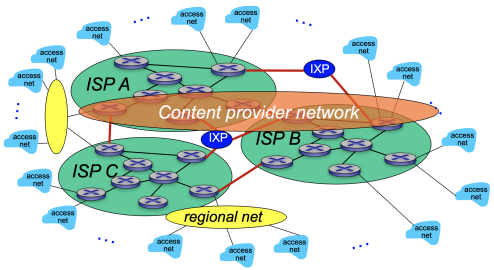
- **Advantage** - Performance guaranteed - no congestion since each packet have own lane
- **Disadvantage** - Reserved resource may not be used

End-to-End Delay

Transmission delay from starting router to end router, **one way only!**

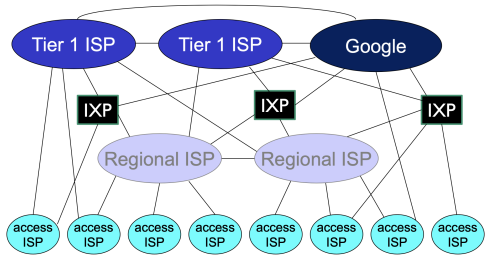
Internet Structure

Network of networks. **ISP** - Internet Service Provider



Connecting local ISPs to each other is **not scalable** - $O(n^2)$. Create **Global ISPs** for them to connect with each other. Competitors will appear in this case (ISP_A, ISP_B, ISP_C). Need to connect these global ISPs together. Use **peering link** or **Internet Exchange Point** (IXP).

- **Peering links** requires bilateral ties and guarantees performance.
- **IXP** cheaper but no speed/performance guarantee and can get congested.
- **Regional Net** connects regional networks together before going to global ISP
- **Content Provider Network** is a private company network for it workers to connect to. Doesn't rely on global ISPs(cheaper, and better performance), direct connection to access points of the same company.



Google (Content Provider Network) can be seen bypassing Tier 1 ISP and Regional ISP to connect to access ISP.

Throughput

Rate at which bits are transferred between sender/receiver

- **Average** - rate over longer period of time, refers to a packet
- **Instantaneous** - assume a stream, throughput would be the smallest rate because of bottleneck. Calculate total time divide by size of 1 packet.

Protocols

Network protocols are communication between machines. Defines format, order of messages sent and received among other network entity and actions taken on message transmission, receipt.

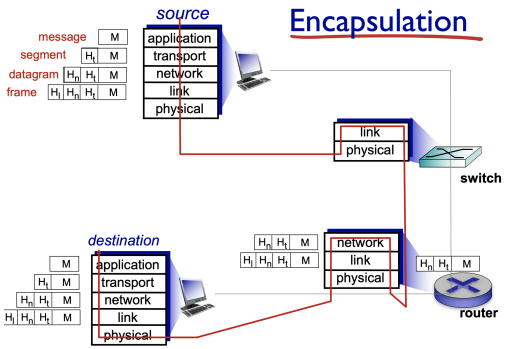
Layers

Similar to airport idea, each layer is a service, travelling between countries is the link. When arriving at each airport, procedures are the same.

- Allows identification, relationship of complex system pieces
- Modularisation for easier maintenence and updating - can update specific layer without changing entire protocol

Layer details (from top to bottom):

- **Application** - support network applications (HTTP)
- **Transport** - interprocess data transfer
- **Network** - routing of data from source to destination
- **Link** - data transfer between neighboring network elements (whats the difference between link and network? - network is like the forwarding in the routers while link has some form of transfer)
- **Physical** - bits on the "wire"



Network Application Architecture

Client-Server

- **Server** - hosts that are always on, permanent IP address and data centers for scaling, sends response.
- **Clients** - communicates with server, may be intermittenly connected, may have dynamic IP address, doen't communicate with another server directly, requests from server. **Don't always have to be alive**

Peer-to-Peer

- Not always on servers
- Everyone can talk to everyone else
- Help one another to serve others - **self-scalability**.
- Complex management, since IP addresses can change.

Process Communication

- **Process** - program running within host. Within host, process communicate through **IPC**. In different hosts, **send messages**.
- **Client Process** - process that **initiates** communication.
- **Server Process** - process that **waits to be** contacted

Application Protocols

- Types of messages - request or response
- Message syntax - format of message
- Message semantics - meaning of message
- Behavior of communication - where and how should process send and respond

Open Protocols - defined in RFCs, allow for inter-operability(as long as speaking same language as server, can communicate already).
Proprietary Protocols - Skype

HTTP

Hypertext Transfer Protocol. Something like the language. Requests for only 1 object at a time.

`www.someschool.edu/someDept/pic.gif`

host name | path name

Web page consists of objects like HTML, JPEG, etc. Each object addressable by URL.

- Client - requests, receives using HTTP and displays web objects
- Server - sends objects using HTTP

Uses TCP :

- Client intiates TCP to server
- Server accepts TCP
- Client and server talk in HTTP via TCP
- Close connection when done
- port 80

Stateless - server doesn't store prior client info. In the shopping cart context, when user exits, given fresh cart.

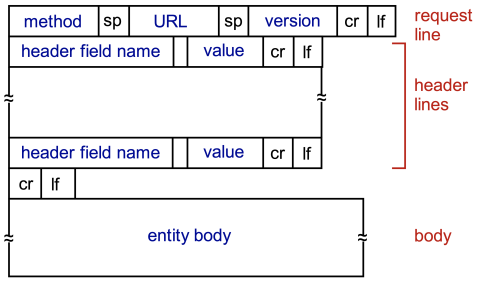
HTTP Persistency and Response Time

RTT - time for packet to travel from client to server and back

- Non-Persistent - Connection will close, downloading multiple content require multiple connections since closes after every response. Response time = 2RTT + file transmission time
- Persistent - Connection timeout, then closes. Allows for multiple objects to be sent over 1 connection. Response time ≥ RTT

Persistency doesn't determine whether a connection is using TCP or UDP. TCP can be persistent, and UDP can be non-persistent.
Independent Concepts!!!

HTTP Request



- Request line - version(HTTP/1.1 or HTTP/1.0), carriage return character, line-feed character
- Header line - header field-header value, basically key-value pairs
- Body - optional

Uploading inputs

- **POST**(keyword) - input uploaded as part of the body
- **URL** - uses GET and inputs are put in URL instead

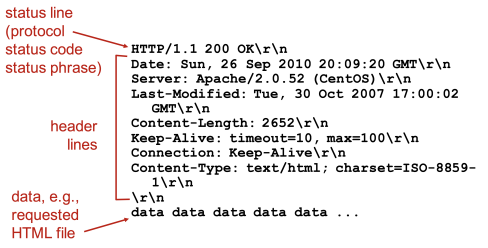
HTTP Methods

- GET - requests data
- POST - send data to server to create or update resource; input uploaded as part of body
- PUT - similar to POST, but calling same PUT produce same result
- HEAD - test if server has content
- DELETE - delete and modify file

Method Types

- **HTTP/1.0** - GET, POST, HEAD(test if server has content), uses TCP
- **HTTP/1.1** - All of the above , PUT(web uploads) and DELETE(delete and modify file at URL), default is persisten

HTTP Response



HTTP Response Codes

- 200 OK - request success, object in body
- 301 Moved Permanently - object has moved eleswhere, location in body
- 400 Bad Request - request not understood
- 404 Not Found - request not Found
- 505 HTTP Version Not Supported

Performing HTTP as a client

- telnet <domain name> 80 - open TCP connection to port 80 (default HTTP server port). Anything typed sent header
- GET <path> HTTP/1.1 Host : <domain name> - sends GET request to HTTP server
- Response message by HTTP server

Cookies

Components :

- Header line of response
- Header line in next request
- File kept on host, managed by browser
- database of server

Serves as an identifier when accessing a website.

- Create entry in database
- Set cookie-id
- Create new entry in cookie file
- Subsequent request, just check database and perform cookie-specific action

On deletion, doesn't remove entry from database, but instead removes id from cookie-file. Information kept at the endpoints. Protocol endpoints - maintains state at sender/receiver over multiple transactions. Cookies : http messages carry state.

Web caches(proxy server)

Goal: satisfy client request without involving origin server. Something like a cache for HTTP requests. Faster response and reduces traffic. **Problem** : content in original server more updated than cache.

Conditional GET

Proxy queries server to see if its updated or not. Not modified before date - proxy send query with info of cached copy, server response whether its modified since then. Otherwise, updates cached copy with data from response.

Domain Name System(DNS)

Mapping between IP address and host name. Uses:

- Distributed database - implemented as a tree structure of servers
- Application-layer protocol - hosts and servers communicate to resolve names

Can go down hierarchical tree - Root to servers. DNS queries are not the same as HTTP. HTTP response is a file object. Doing completely different things.

- DNS listens to UDP port 53
- Failure to contact DNS servers can cause disruption in access to Internet services.
- A hostname may be mapped to multiple IP addresses

Local DNS

Acts as a cache before accessing the database. Contains recent name-to-address pairs but may be outdated. Each ISP has 1.

Top-Level Domain (TLD), Authoritative DNS

- TLD - responsible for com, org... and all top-level country domains like sg
- Authoritative DNS - organisation's own DNS server(s), provides name to IP mapping for organisation's hosts.

Queries

Iterated - host requests and local DNS handles request. Local DNS asks all the different kinds of servers(in hierarchical order, root, then others) until answer found. **Recursive** - goes down tree from root until found, then backtracks with the answer. Root requests on behalf of local.

Caching

Timeouts on the cached data, have to constantly look for updates once timed out.

Sockets

Process communciates by using sockets. Something like a mailbox. Sending process relies on **transport layer** on receiving end to deliver message. **Abstraction** between Application and Transport layers.

Addressing processes

Identifier for each message, includes :

- IP address - not enought because there can be many processes with same IP address.
- Port number with process on host - well known service, PID may change when host restarts.

Transport services

- Data Integrity - some apps require 100% (file transfer), others not so much
- Timing - some need low delay to be effective, usually for real time
- Throughput - per unit time, how much data transfer. Some require minimum to be effective
- Security - encryption and data Integrity

Transport Protocols

TCP properties :

- Guarantees data integrity
- Reliable
- Flow control - sender won't overwhelm receiver
- Congestion control - throttle sender on overload
- Doesn't provide - timing, throughput guarantee and Security
- Need a connection between client and server

UDP properties :

- No data Integrity
- Unreliable transfer
- Doesn't provide anything

UDP aimed at services who don't require data integrity like multimedia services and its cheaper to do so because there's minimal overhead.

Socket Programming

Predefined port for service. Client can use random port number. But must be different from service side. Reasons :

- Running different process
- Assymetry between server and client (assume server always on and client always initiate)

Protocols depends on socket programming. Port number identifies which socket to use.

Services

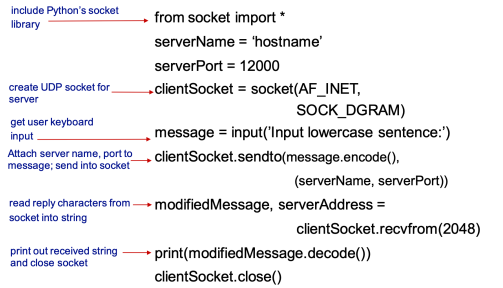
- **UDP** - Unreliable datagram (no guarantee that data reaches destination, either dropped or arrive out of order), don't need connection.
- **TCP** - Reliable data stream, requires connection.
- Sockets identified by tuple (source IP, source port, dest IP, dest port)

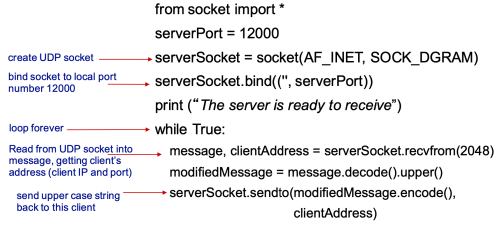
UDP

- No connection means no handshake required.
- Implies sender need to specify which server
- Server also need to identify which client to reply together
- Client can send data to non-existing UDP server without observing server is offline.
- Sender explicitly attaches destination IP address and port number to **every packet**.

Logic for Socket Programming UDP :

- Server run on server ip address
- Server create server side socket
- Read datagram from socket
- Client can contact at any time. Client creates socket and port number
- Client send datagram via own socket and then read from own socket
- Internet sends the packet back and forth
- On receive, extract, process and reply via server socket (specify client IP and port number)





sendto - send message (in bytes) to destination (tuple pair of hostname/ip address and port number). **receivefrom** - buffer of modified message its willing to receive. Takes in actual message and server address (ip address and port number). Server need to specify port number before using socket - bind port number with socket (tuple of hostname and port number). Cannot rely on OS because it may be random then clients can't bind to it.

TCP

Difference - TCP need to build connection for each client and each need their own sockets. So create 'new sockets' for each client. In reality its just multi process/threading. TCP server needs to be running first. **TCP servers already have 1 socket to begin with which is the welcome socket.**

- Server create welcome socket
- Wait for requests
- On connection, the server and client have their own dedicated sockets for communicating.

Client

- Need built connection
- Calls connection method with no data
- Just use a method called send
- Dont need to specify like UDP because there is already a connection

Server

- Need to accept the connection request from client
- Listens to connection requests, sets upper bound on number of connections allowed.

```
from socket import *
serverName = 'servername'
serverPort = 12000
clientSocket = socket(AF_INET,SOCK_STREAM)
clientSocket.connect((serverName,serverPort))
sentence = input('Input lowercase sentence:')
clientSocket.send(sentence.encode())
modifiedSentence = clientSocket.recv(1024)
print('From Server:', modifiedSentence.decode())
clientSocket.close()
```

create TCP socket for server, remote port 12000

Establish a TCP connection with the server process

No need to attach server name, port

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print('The server is ready to receive')
while True:
    connectionSocket, addr = serverSocket.accept()
    sentence = connectionSocket.recv(1024).decode()
    capitalizedSentence = sentence.upper()
    connectionSocket.send(capitalizedSentence.
                          encode())
    connectionSocket.close()
```

create TCP welcoming socket

server begins listening for incoming TCP requests

loop forever

server waits on accept() for incoming requests, new socket created on return

read bytes from socket (but not address as in UDP)

close connection to this client (but *not* welcoming socket)

Transport Layer

- Provide logical communication between app processes running on different hosts
- Sender - breaks app message into smaller packets (segments) and passes to network Layer
- Receiver - reassumbles the message and pass back to app layer
- Protocols only implemented at end host.

Network Layer

Communication between hosts - versus **Transport layer** which is the communication between processes.

UDP - User Datagram Protocol

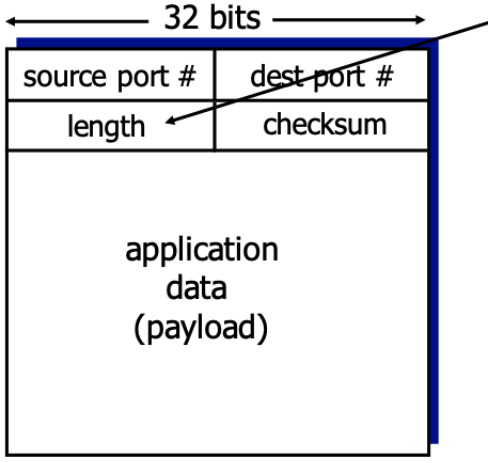
Adds little to network protocol - unreliable, to make it reliable, add other forms of checking. No connection establishment which can add delay, small header size, no congestion control.

Connectionless De-multiplexing

UDP uses the same socket destination socket number for every host. OS check the port numbers.

Segments

Contains information of the segment. Format :



UDP segment format

Source and destination ports are 16 bits each. The length of the segment including the header is at least 8 bytes(64 bits) because of the size of the header. Checksum component is for the receiver to detect errors - check if bits are flipped or not. **Header value counts number of bytes and represents it as a 16bit integer.**

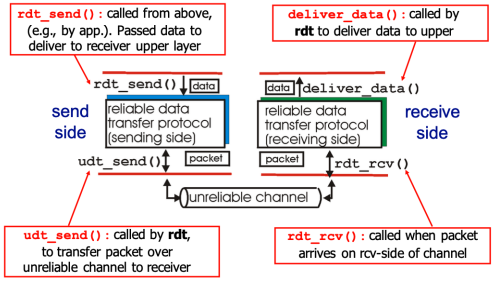
UDP Checksum

Detect errors in received segments. Add numbers together, **carry from most significant added to result**. Get 1s complement.

Principles of reliable data transfer

Initially, between applications, we know they use either TCP or UDP to get information across. Now - implementing the reliable channel makes use of an unreliable chnalle from the network layer.

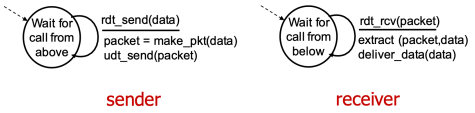
Reliable Data Transfer



`rdt_send()` is called by any generic layer to provide reliability. Then packets are sent via the unreliable channel using `udt_send()`. Method called by **rdt** to transfer packet over. On the receiver side, `rdt_rcv()` called when packet arrives on the receive side of the channel. Then it is reassembled into the data and deliverd to upper layers via `deliver_data()`

rdt1.0

Assuming underlying channel is perfect (no errors and no loss of packets) - then state diagram will be similar to that above. No checking needed. If event happens, then make transition.

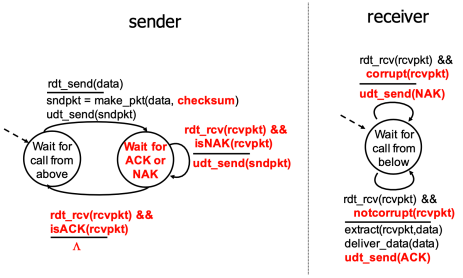


rdt2.0

Suppose now underlying channel may flip bits in packet - use checksum to detect. Once detected, recover by asking sender to send again.

- acknowledgements (ACKs) - explicit **OK** by receiver
- negative acknowledgements (NAKs) - explicitly say got error, sender will retransmit

On **ACK**, sender will send more packets. On **NAK**, sender will retransmit the last packet. **Stop and wait protocol (sending one at a time on ACK, then wait for receiver response)**.



Need to include checksum when sending so that receiver can check to see if corrupted or not.

ACK/NAK Corruption

On corrupted acknowledgements, just resend the same packet. Solves the issue for **NAK** since packet wasn't received anyway. But duplicate packet won't be detected in the case of **corrupted ACK** - receiver see duplicate packet as a new packet and will just take it anyway.

Solution

Add sequence number to each packet and receiver discards duplicate - receiver can now detect duplicates using identifier number. On detection, send back **ACK**

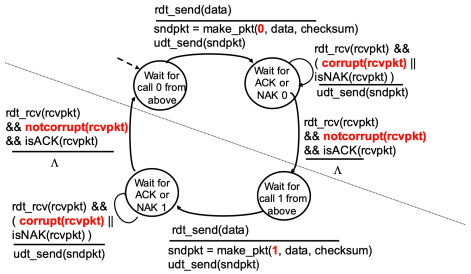


Figure 1: Sender

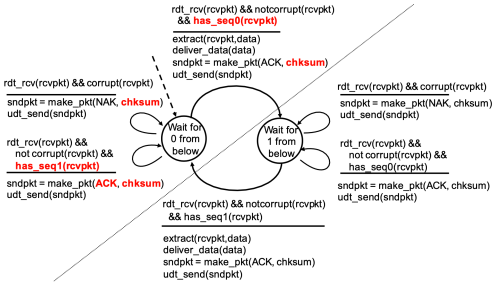
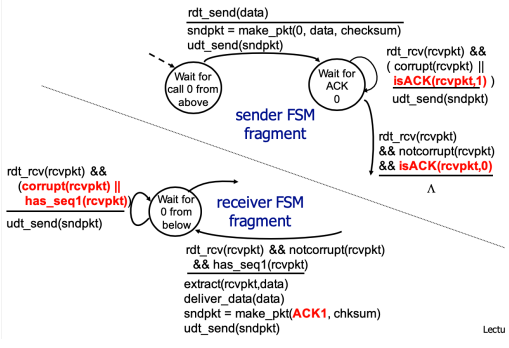


Figure 2: Receiver

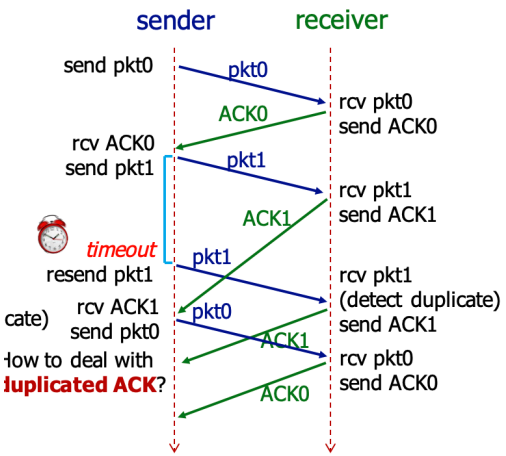
NAK-Free

Instead of **NAK**, receiver send **ACK** with sequence number of last packet received. Now, **NAK** = **ACK**n, where n is the sequence number of the last packet received.



Channel Loss & Errors (rdt3.0)

Now, underlying channel can also lose packets (ie network layer) but won't reorder packets.. Sender waits some time for **ACK**, requires timer. After some time if no **ACK**, then retransmit. But packet may not have been loss, so there will be duplicates. Resolved already using unique sequence number (on receiver side). Sender may receive duplicate acknowledgements because of this.



Previously, because of the lack of timer, the sender get stuck. With timer, sender can just idle and let the timer timeout before transmitting again.

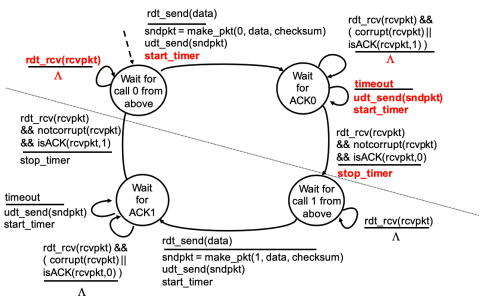


Figure 3: Sender

Diagram for receiver is the same as Figure 2. Looking at diagram, on corrupt, sender does nothing. And just waits for ack0.

Performance

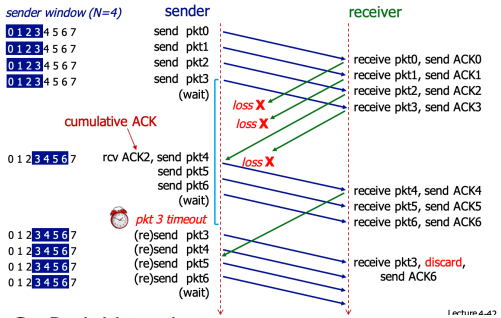
Network protocol limites physical resource. Effective utilisation by sender is very little. Actual utilisation refers to transmission time ($\frac{L}{R}$), if RTT much greater than transmission time, then effective utilisation would be $\frac{L/R}{RTT+L/R}$.

Pipelining

To better utilise, increase number of packets sent during RTT. Then $\frac{xL/R}{RTT+L/R}$, where x is the number of packets in the pipeline

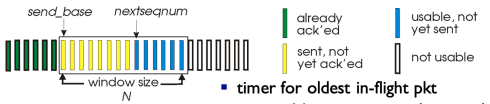
Go-Back-N

- Sender : window size of N, sequence number in packet header. Timer for oldest in-flight packet. On timeout, retransmit packet and all higher sequence packet numbers in window
- Receiver : **ACK** for correctly received packet in order. Possible duplicates handled using **NAK-free** scheme. Remembers **expectedseqnum**. If packet received out of order, ignore.



Go-Back-N: sender

- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed



- timer for oldest in-flight pkt
- *timeout(n):* retransmit packet n and all higher seq # pkts in window

Go-Back-N: receiver

- always ACK for correctly-received pkt with highest *in-order* seq # : *cumulative ACK*
  - possible duplicate ACKs
  - need only remember **expectedseqnum**
- out-of-order pkt:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK pkt with highest in-order seq #

Selective Repeat

Selectively retransmit packets that were not acknowledged. Receiver creates buffer for out-of-order packets. When the missing packets are delivered, then send everything inorder to the upper layer. Maintain timer for each unACKed packet.

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-1]:
- mark pkt n as received
- if n is the 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

TCP Buffer & Segment

- Dual direction - both receiver and sender buffer
- Byte stream (not a packet stream), TCP don't see it as packets. Byte stream is a sequence of bytes
- Keeps track of certain info to achieve reliability
- TCP segment is about 1460 bytes instead of fully using 1500 bytes for Ethernet. 40 bytes used for header, 20 for TCP, 20 for packet.
- **MSS only inludes data**

Connection-oriented demux

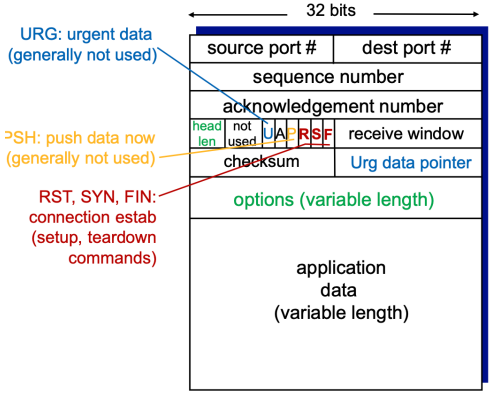
Segments all go to same IP address. **How to differentiate?** - use source IP and source port number. Incoming segments demultiplexed to different sockets. **TCP socket identified by (source IP, source port, dest IP, dest port)**. Initial sequence number chosen randomly.

Alternate sockets created using multi-threaded programs which is

- handled by OS
- For parallelism

TCP segment structure

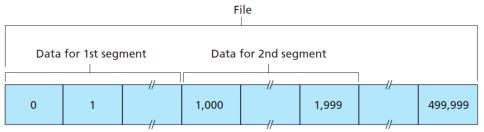
source port number, dest port number and checksum same size and location as UDP.



- 32 bit sequence number - which packet i am current sending
- 32 bit acknowledgment number - which packet i want to acknowledge next.
- 1 bit acknowledgment - 1 (packet includes acknowledgment), 0 (no acknowledgment, pure data)
- head len - length of packet header
- options - can put more data (optional)
- psh - pushes data to upper layers
- urg - specify which part of data is important, referred to by pointer.

TCP sequence number

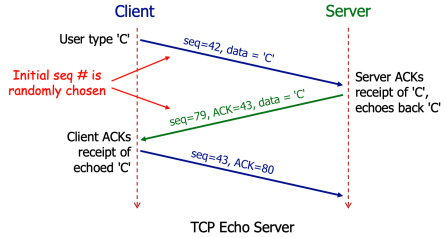
Byte number of first byte of data in segment. File in buffer packed into segments. Sequence number of segment is index of 1st byte of packet.



1st segment has sequence number 0, 2nd segment has sequence number 1000, etc. **How receiver handles out-of-order segments, depends on implementor**

TCP ACKs

Sequence number of next byte expected from other side. **Cumulative** - implicit that everything before is correct. Receiver doesn't handle out-of-order segments(up to implementor).



Data sent can be empty. Server acknowledge received and set the bit as well as put **ack number of next expected segment**.

Flow Control

Controls sender so sender doesn't overflow receiver's buffer. Inform sender that buffer almost full, pls stop sending(no packet dropped). If application is very slow, then buffer will fill up and may overflow.

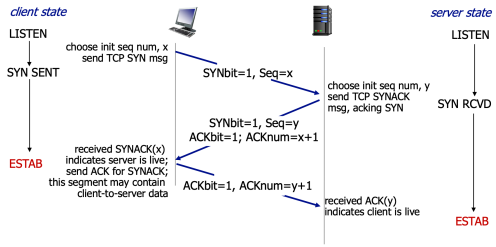
- Receiver advertise free buffer space by including receive window value in header
- Size of window set via socket options
- Sender limits number of unacked data according to the value received. Value changes as buffer gets filled or empty
- Guarantee no overflow

Each TCP socket has their own buffer.

Connection Management

Handshake on agreements

- establish connection (each knowing the other willing to establish)
- agree on connection parameters

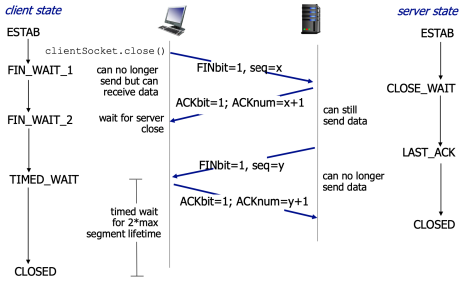


- Listen - nothing established
- TCP need sync bit and seq num.

- Client send sync and seq to server
- Server send back with sync bit and ACKs
- Sender become **ESTAB** - receive info that server is live and send back with ACK bit = 1, to acknowledge the syncing.
- client send back server's ack num + 1 and ackbit. At this point, client can add data already. Syncing done, no more sync bit here
- Server received ack indicating client is live too.

Closing Connection

Whoever sends FIN, cannot send data, only ACKs.



- Client closes connection
- FIN_WAIT (immediate), can no longer send data, only acknowledgments
- Set packet FIN bit to 1
- Server acknowledge and close wait
- Now client FIN_WAIT 2 (waiting for server to close)
- Client can start cleaning up
- If client doesn't send anything, server sends FINbit to client
- Client waits some time in case of corruption (ie repeated send or sth)
- Client send ack(might get lost)
- Server close

Reliable Data Transfer

Sender keeps 1 timer and retransmit only oldest unACKed packet.

- similar to Go-Back-N
- On timeout, retransmit 1 packet and not everything in window. Results in duplicates. Restart timer.

Scenarios of retransmission

- Lost ACK scenario - on loss, sender timer will timeout, so sender just retransmit last.
- On Premature timeout - receiver will send an acknowledgment for the packet that it wants to receive next.
- Cumulative ACK - Because ACKs are cumulative, when one is missing, just continue to transmit the next one. Since the last ack will be asking for the next packet.

TCP ACK generation

event at receiver	*TCP receiver action*	3 4 5
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK	
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments	
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send *duplicate ACK*, indicating seq. # of next expected byte	
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap	

- All data up to expected seq num already acked. Arrival of in-order segment with expected seq num. Receiver sends delayed ack (assume gonna receive next one soon). Send ack if no segment
- Arrival of inorder, but one other segment has pending ack. Receiver immediate send cumulative Ack, ack both in-order segments. Possible packet loss/stuck. Send earlier to resolve
- Arrival of out-of-order, gap detected. Receiver immediately send back same ack as before. Must have sent ack for the lower numbers before.

TCP RTT Timeout

- RTT varies
- timeout too long - reaction will be slow
- timeout too short - premature timeout (too impatient)

Estimated by recording time and calculate RTT for 1 packet. Ignoring retransmission. This time can vary greatly, need to smoothen. Apply weights over samples. Estimated RTT = $(1-\alpha)$ * Estimated RTT + α * sample. Timeoout interval need to include safety margin - **Why need safety margin?** : average will still vary, give buffer for average with large deviations.

TCP Fast Retransmit

- Timeout period often relatively long (long delay before resending).
- Detect lost segments via duplicate ACKs - since sender sends many acks, if something is loss, there will be many duplicates.
- If sender receives triple duplicate ACKs, resend unack with smallest seq (likely that that segment is lost) immediately.

TCP	GBN
Uses Cumulative ACKs	
Resends unacked / next segment	Resends entire window

TCP	SR
Resends unacked segment/pkt	
ACKs num will be last acked segmnt, possible duplicate ACKs	ACKs num can still change

Network Layer

Network layer protocols are in **every** host, router - connects everything together. Sender encapsulates segments into datagrams. Receiver delivers segments to transport layer. Routers don't have transport layer.

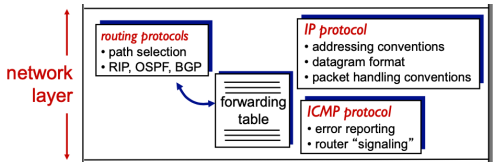
Network Layer Functions

Forwarding tables formed by routing algorithm. Think of routing like the entire path and forwarding just containing adjacency lists.

- Forwarding - move packets from router to router
- Routing - determine the route that packet need to go from source to destination

Modularised Network Layer

- Data Plane - in charge of forwarding
- Control Plane - determine how data is routed from source to destination (end-to-end).
  - Traditional Routing Algorithm - talking to routers to understand how to run
  - Software defined(not covered)

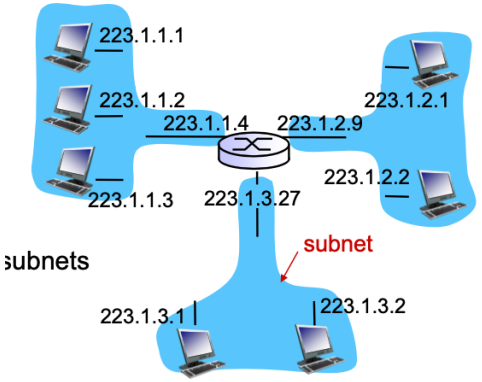


Routing protocols (no single routing protocol) determine routing tables. IP protocols needed for communication (must have). ICMP protocols are optional.

IP addressing

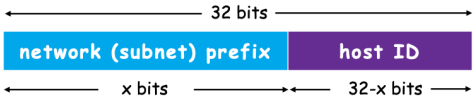
IP address - 32 bit identifer for hsot, router interface. Unique and not assigned to host router, assigned to network interface. IPv4 - 32bits. Interface is the physical boundary between host/router and physical link. Example : wired and wireless.
Interfaces connected by switches. Represented as dotted decimals, which is then converted to binary.

Subnets



Subnets is refers to the 'blue cloud'. Hosts in the same subnet can reach each other directly, without help of router. Connect to other subnets through a router. **Hosts in the same subnet have same network prefix of IP address**
To count number of subnets - disconnect each interface from its host or router, creating isolated islands of networks.

IP addressing: CIDR



Prefix to identify subnet and host id. Total length is 32bits. Network prefix length is not fixed. Address format : **a.b.c.d/x**, where x is the number of bits used in the prefix. In IPv4, each a,b,c,d is 8 bits long.

Subnet Mask

Made by setting all the digits in prefix to 1 and everything else to 0. How to do? - perform bitwise AND. Used to determine which network an IP address belongs to.

IP addresses

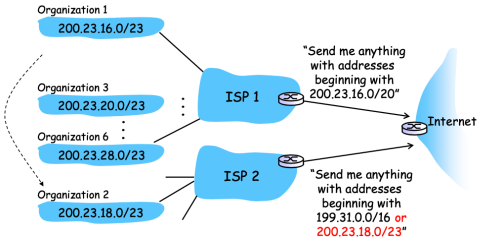
How ISP get blocks of addresses? - Some international cooperation that handles the uniqueness of IP address. Allocate x most significant bits of hostID to each organisation.

Route Aggregation

Multiple organisations under same ISP, ISP then advertises its own IP address to the internet to receive information. **Why does it advertise own and not individual organisations?** - organisation IP address falls under ISP, ISP can request on behalf of organisation and then just re-route it later.

More Specific Routes

Suppose an organisation switches ISP, the new ISP has different network prefix compared to organisation. So ISP has to specifically request for organisation IP.



Will ISP 1 receive contents for Organisation 2?

Longest Prefix Matching

Can have multiple matching prefixes. Choose the one which match for the longest length. Chooses most specific destination address that matches up to subnet mask.

- Packet with destination IP 200.23.20.2 ⇨ R1
  - (Binary: 11001000 00010111 00010100 00000010)
- Packet with destination IP 200.23.19.3 ⇨ R2
  - (Binary: 11001000 00010111 00010011 00000011)

match the longest prefix

Forwarding Table at R3

Destination Address Range	Destination Address Range in Binary	Next Hop
200.23.16.0/20	11001000 00010111 00010000 00000000	R1
200.23.18.0/23	11001000 00010111 00010010 00000000	R2
199.31.0.0/16	11000111 00011111 00000000 00000000	R2

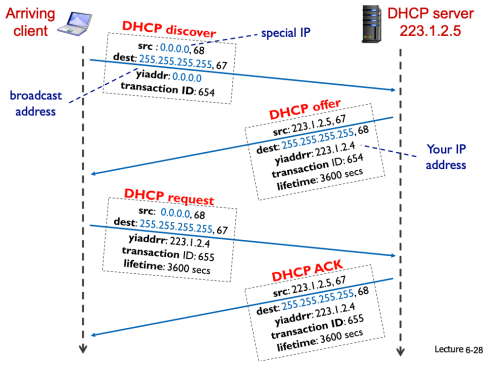
Dynamic Host Configuration Protocol

Dynamically get IP address from server. IP addresses are 'borrowed' by host, on expiration, its returned. Host can use protocol to renew the IP address it was assigned. Think library of books. Runs over **UDP** (at application layer). Server port - 67. Client port - 68. On subnet, can return :

- address of first-hop router for client
- name and ip of dns server
- network mask (indicating network vs host portion of address)

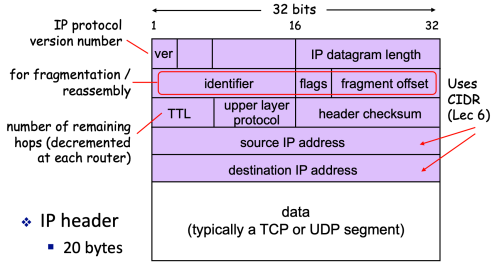
Mechanics Actually quite similar to job application process.

- discover - hosts broadcasts, try to find some server to use (optional)
- offer - server offers host an ip address (optional)
- request - host request to use ip address offered by server
- ack - server acknowledges and let host use ip address



y(our)i(p)addr - server puts allocated address here. Initially host ip is all 0 because it doesn't have one yet. **Why client request src ip still all 0?** - tells others DHCP you're accepting this particular ip from this server.

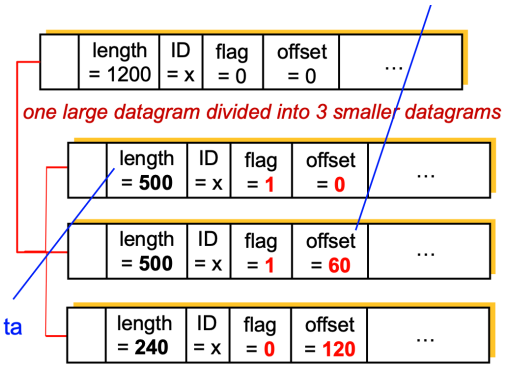
IPv4 Datagram Format



IP header is always 20 bytes. IP header builts on top of Transport layer segment. Transport layer headers will still be present.

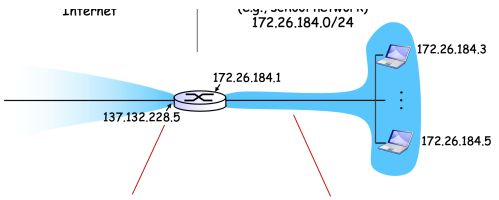
IP Fragmentation

Different links have differnt **MTU** - Max Transfer Unit, amount of data link can carry (includes header). If too large for outgoing link, break it up into smaller fragments. New IP headers are given to the separated smaller fragments.



The headers are actually the same as the original Datagram format. Notice that the large datagram has **flag** and **offset** set to 0. **flag** is a boolean condition for : is there a next fragment from the same segment? **Remember to subtract 20 from length** - and then divide by 8 to get offset.

Network Address Translation



Within the same local network, hosts uses their own private IP to communicate. However, all datagrams leaving the local network have same source IP address. Implementation :

- Translate source IP and port number of every outgoing to NAT IP and new port number (belongs to router??)
- Remember the mappings in a table (table is in the router)
- Translate back upon receiving incoming datagrams.

Table is basically saying - if i receive this from WAN, it is for this address and port in LAN.

Benefits

- No need range of **public** ip addresses from ISP - just 1 for NAT
- All hosts uses private. Can change the IP addresses locally without informing outside. NAT serves as some form of abstraction.
- Change ISP without changing address of hosts - ie just change the router.
- Security - hosts not visible to outside world.

Challenges

- Peer-to-peer won't work directly.

Routing

Autonomous Systems have their own routers and links. Routing is done hierarchically.

Intra-AS

Find a good path between 2 routers **within** an AS. Single admin, no policy decisions needed. Routing focus on performance.

Intra-AS as a weighted undirected graph. Vertices are routers, links are edges. Weights can be some arbitrary number. Goal : **Find least cost path between 2 vertices**

Distance-vector Algorithms

Routers know neighbors and cost of going to neighbors. Exchange views amongst themselves to get better views.

- Swap local view with direct neighbors
- Update own local view
- Repeat till no more change

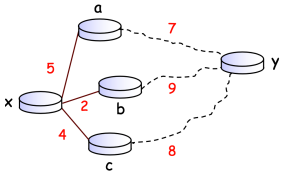
The above appears to be similar to Bellman-Ford.

Bellman-Ford Equation

To obtain the least-cost path from x to y (from x view),

$$d_x(y) = min_v c(x,v) + d_v(y)$$

The least cost from x to y is the minimum sum of each neighbor to y and cost of going to each neighbor.



$$d_x(y) = min_v \{ c(x,a) + d_a(y), c(x,b) + d_b(y), c(x,c) + d_c(y) \}$$

$$= min\{12, 11, 12\} = 11$$

The neighbors know their cost because their neighbors forwarded such information to them. It is actually forwarding information backwards.

ICMP

Internet Control Message Protocol - used by hosts and routers to communicate network-level information. Error reporting and echoing requests/replies. These messages carried in IP datagrams (after IP header).

Type	Code	Description
8	0	echo request (ping)
0	0	echo reply (ping)
3	1	dest host unreachable
3	3	dest port unreachable
11	0	TTL expired
12	0	bad IP header

Ping and Traceroute

ping - sees if remote host responds. **traceroute** - displays the route that packet would take

Link Layer

Focus on single link between routers. IP datagrams in linked layer frames, used for transmission. Link layer closer to hardware, must provide specific protocols and some bonuses. Switches (like power strips) not visible to individual hosts. Implemented in adaptor or chip.

- Communication between 2 hosts
- send datagram between adjacent nodes over single link

Services

- Framing - Encapsulate datagrams into frames with header and trailer.
- Link Access Control - Coordination of nodes to control who sends what and when.
- Reliable Delivery - used on more error-proned links
- Error Detection - errors caused by noise. Receiver tells sender if theres error
- Error Correction - receiver correct the errors if possible without asking for retransmission

Error Detection

Not 100% reliable

Parity

- Count number of 1s in data
- even and odd parity resolved by adding extra bit if needed.
- Less overhead, but less reliable (only 1 bit to detect errors)
- Receiver checks data and parity bit.

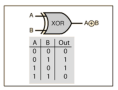
Unable to detect multiple bits flipped if the overall number of 1s remain unchange. Use 2D bit parity instead! Detect and **corrects single bit errors**. Overhead would be the total number of parity bits in all the rows and columns. Steps to check :

- logically arrange data
- compute parity for each row and column
- corner result will be the sum of the row parity and column parity

Higher overhead because more parity bits involved. Able to detect 2 bit errors but cannot correct. Multiple combinations can lead to same error, hence cannot correct. If there are more than 1 intersection between rows and columns, there can be multiple combinations.

Cyclic Redundancy Check (CRC)

- Powerful error-detection coding that is widely used in practice (e.g., Ethernet, Wi-Fi)
  - D: data bits, viewed as a binary number.
  - G: generator of $r+1$ bits, agreed by sender and receiver beforehand.
  - R: will generate CRC of r bits.



Example: $r = 3$

$$1001 \overline{)\,101110000}$$
$$1001$$
$$10$$
$$1001$$
$$1100$$
$$1001$$
$$1010$$
$$1001$$
$$011$$

G D R

Doesn't identify where the problem is but just indicates that theres a problem. Generator bits are agreed by sender and receiver. $D\%G = R$. Send $D + R$ to receiver. To perform division, do **bitwise XOR with no carry or borrows**.

Network Links

- Point-to-point - sender and receiver connected by a physical link
- Broadcast link - like wifi, multiple devices connected to wifi. Leads to collision if all devices simulateneously transmit information.

Multiple Access Protocols

Ways to deal with collisions. Must use the channel itself, cannot use help from external channels.

Channel Partitioning

Divide channel into smaller pieces based on some unit (time, frequency) for each node to use. Dedicated allocation of service to nodes. Need to repartition if new node is added.

- Time Division Multiple Access (TDMA) : division by time. Channels get **fixed** length slot. Unused slots just idle. Guaranteed no collisions.
- Frequency Division Multiple Access (FDMA) : division by frequency. Channels get specific bandwidth frequency to transmit at.

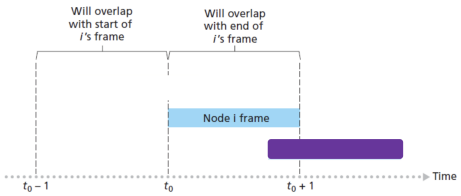
Taking Turns

Polling - Nodes take turns to use the **full** channel to transmit. Master creates a polling system. Sequentially (round robin) asks every node if they have something to send. If nothing, just go to the next one, resolves idling issue. Issue arises if master dies, then entire network idles since slaves can't anything till they get signal from master.
Token passing - something like passing a microphone around, whoever has the mic can transmit. Token fail, everything fails. And theres overhead from token as well.

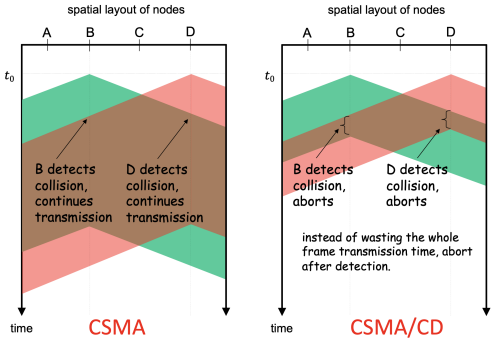
Random Access Protocol

How to recover from collisions? How to detect collisions? **SLOTTED ALOHA** - listens to collisions. On collision, decide whether to retransmit with a probability p. Retransmission solved with a probability of p. Can result in wasted slots and more collisions. But guaranteed to be retransmit. Choose p wisely, higher value results in lower waster, but more collisions. Assume same frame size, time division same and nodes transmit at the start.**UNSLOTTED ALOHA** - if node has data, just send. Chance of collision high between frames if not aligned.

- Chance of collision increases:
  - frame sent at t_0 collides with other frames sent in $(t_0 - 1, t_0 + 1)$



Carrier Sense Multiple Access (CSMA) - sense channel to determine next action. Don't interrupt others when they using channel. Still can result in collisions because of **propogation delays**. On detection, finish transmission then retransmit whole frame later. **CSMA/CD (Collision Detection)** - on collision, immediately stop transmission. **Frame size affects whether collision detected**. If frame size too small, collision go undetected (host finish transmitting already, but collision actually happens elsewhere. ie collisions not detected by sending nodes.)



CSMA/CA (Collision Avoidance) - on broadcast link, router tells hosts whether there was collision or not.

MAC address

By manufacturer. Used to send and receive link layer frames. On receiving a frame, check if its mine. If it is, I process it. Otherwise I throw it away. 48 bits. First 3 bytes identify vendo of adaptor. Comparison between IP address and MAC address :

IP address	MAC address
32 bits in length	48 bits in length
network-layer address used to move datagrams from source to dest.	link-layer address used to move frames over every single link.
Dynamically assigned; hierarchical (to facilitate routing)	Permanent, to identify the hardware (adapter)
Analogy: postal address	Analogy: NRIC number

Use **ARP** table to derive MAC address. Stores mapping of IP and MAC address of **other nodes in same subnet**. Has a timeout after which mapping is forgotten.

Sending frames

Same Subnet

- If A knows B MAC address, create frame and send it to B. Other nodes that may receive it will drop the frame. Only B will process this.
- Else, broadcase to everyone in subnet with B ip address (possible because every host in subnet can talk to each other). Only B replies with MAC address. A caches it and sends the frame.

Different Subnet

Use router to route to other subnet. Use router's MAC address, then router can send to B. Set destination MAC address as router's and destination IP as B. When R receives the frame, construct a new frame with B's MAC address and with source MAC address as router's. Cannot specify MAC of destination directly. Since address not in subnet, every host will ignore.

LAN

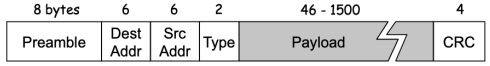
Computer network that interconnects computer within geographical area.

Ethernet

Wired LAN technology. Topologies :

- Bus - all nodes collide with each other
- Star - switch in the center, no collisions

Encapsulates IP datagram in Ethernet frame. Connectionless and unreliable. Uses CSMA/CD with exponential backoff.



Structure :

- Preamble : 8 bytes, used to synchronize receiver and sender clock rates
- Src and dest MAC address : if dest MAC matches current or is broadcast address, pass to higher layer protocols. Else discard.
- Type : higher layer protocol mostly ip
- CRC : checks for corruption, drop frame on corruption.
- Payload length : minimum 46 bytes (frame is 64 without preamble, ie 18 + payload ¿= 64, but cannot exceed 1500 because that is MTU)

Collision for bus topology. Transmission of frames in opposite(?) direction while there are frames in the link. Algorithm :

- If channel idle, send frame. Otherwise, wait till channel idle.
- If no secondary transmission detected, we are done.
- On secondary transmission detected, abort and jam.
- Enter binary back off. Rolldice to get how long to wait : up to $2^m - 1$ where m is the collision number. Wait time would be value times 512bit(64 bytes) time. Value of m increases as more collision, exponential backoff - longer backoff for more collisions.

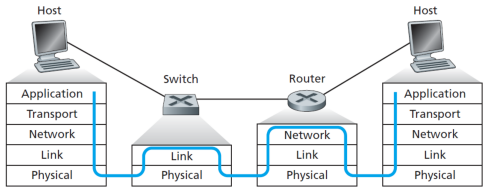
Ethernet Switch

Device used in LAN, store and forward Ethernet frames. Examine incoming and selectively forward to one or more links. Hosts **don't know** about this. In star topo, hosts have dedicated connection to switch. Buffers frames and is bidirectional. **No collisions** and uses ethernet protocol.

Self-learning

Switch has switch table that maps MAC address of host, interface (link) to reach host and TTL. Similar to sending frames in subnets, when switch receive from a host, saves its info in table. If destination not in table, broadcast, and can save all.

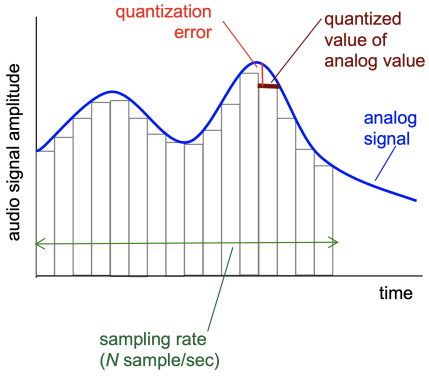
Routers	Switches
Check IP address	Check MAC address
Store-and-forward	Store-and-forward
Compute routes to destination	Forward frame to outgoing link or broadcast



Switches for within subnet, routers for inter-subnet.
Bus topology advantages are easy to extend, simple and reliable, cheaper and less cables. Disadvantages are collision prone, not isolated and performance depends on number of nodes in network.
Star topology advantages are isolation and handles heavy load well. Disadvantages are expensive and that the switch is the single point of failure.

Audio Streaming

Sample audio to convert it into a file. Constant rate of sampling.



Each quantised value represented by bits. Nyquist-Shannon sampling : sampling frequency should be more than double of highest signal frequency. Conversion rate would be sample rate multiplied by number of bits to represent quantised value. Converting bits back to analog signal can lead to quality reduction.

- ADC - analog to digital converter
- DAC - digital to analog converter

Video Streaming

Videos are a series of images. Exploit localities to reduce number of bits used in encoding.

- Spatial encoding - instead of sending all the pixels of same colour, send 2 values instead; colour and number of pixel.
- Temporal encoding - instead of sending entire frame, send only the difference between current and next.

Encoding rates may differ, depending on requirements

- Constant bit rate - frames that require more bits will be lower in quality (optimised for space)
- Constant quality - varied bit rate because each frame may require different number of bits (optimised for quality)

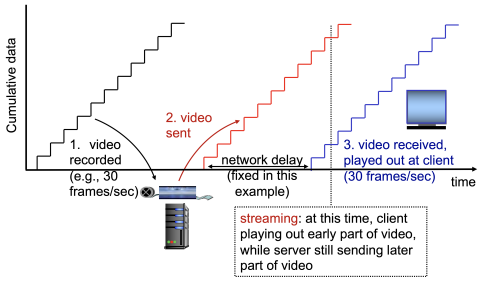
Motion Compensation

Load the difference between current and next frame. If there are no difference, pixel is blacked. If have then there is some colour, implying motion. Analysis direction of frame to reduce the difference. Ideally aim to minimise total difference, because less bits used and better compression.

Multimedia Networking

- Streaming stored audio, video - data stored on server somewhere, just playing from server. Not live
- Conversational - 2 way bidirectional transmission, short latency
- Live Stream - 1 way with limited interaction. Latency doesn't matter

Streaming Stored Video



Some delay when content is received by client. Client start playing frame 1, server sending frame 10. In reality, network delay not fix. Challenge : **client staircase same as server staircase**. Varied client video reception does not guarentee this. Use buffer to compensate for varied arrival times.

- press play to start buffering
- once buffer filled up to certain point, start to output content from buffer
- **playout rate is constant**
- **fill rate varies**
- if fill rate less than playout rate, alot of buffering, video freezing on playout
- if fill rate greater than playout rate, buffer overflow

Possible Solution : Use a bigger buffer and let the user wait longer. Solves overflow with size. Longer wait time absorbs variation in fill rate. Provides more room for error. **Goal : average fill rate = playout rate**

How Client Get From Server

Server Push

- **Uses UDP** - server just keep pushing to client
- Not aware of congestions, client may receive errors
- End-to-end playout is very short, but can be blocked by firewall
- **Real Time Protocol**

Client Pull

- **Uses HTTP** - client requests from server instead. Underlying transport layer protocol is **TCP**
- TCP handles send and receiver buffer.
- Server doesn't do anything. Client just requests from server
- Because of TCP, lead to large playout delays
- Passes through firewalls

Voice Over IP

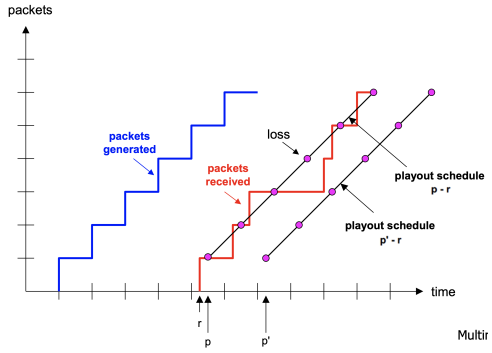
Similar to how there is a lag between reporters and newsroom when they call. Application layer header added to chunk. Chunk and header encapsulated in transport layer protocol. Audio is alternating talk spurts with silent periods. Quality depends on :

- Network Loss - IP datagram lost by router
- Delay Loss - packet arrive too late, dropped
- Loss Tolerance - how much packet loss can be tolerated?

Fixed Playout Delay

Client try and playout each chunk at some fixed delay time. If arrived, can play. Once time is passed, data cannot be played.

- Large delay - less packet loss but not as interactive
- Small delay - more interactive, but more packet losses



By shifting the playout delay from p to p', we can ensure that packets are not loss. But this incurrs longer delay (distance between blue line and black line increase).

Adaptive Playout Delay

Goal : Low playout delay, low late loss rate. Use exponentially weighted moving average to estimate packet delay. On more congestion, make the delay longer.

Packet Loss Recovery

Using NAK/ACK takes one RTT, too slow. Use **Forward Error Correction**. A simple FEC scheme is :

- Create redundant chunk by XOR-ing n original chunks
- send n + 1 chunks, increasing bandwidth by factor of 1/n
- reconstruct original n chunks if at most 1 lost chunk from n+1 with playout delay
- if the loss exceeds a certain limit, cannot recover for the grp

A possible FEC scheme is :

- Add more data to prevent retransmission, like parity.
- Ideally, include 2 versions of the original, high and low quality.
- Contents of current packet contains high quality of current and low quality of next.
- If next packet is loss, at least still have the low quality one to work with.

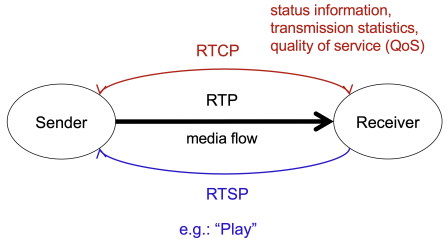
Can also use **Interleaving**

- Divide audio chunks into smaller units
- Scramble the units up with those from other audio chunks
- Packet now contains chunks from different parts of the original streaming
- On packet loss, still have most of the content
- Exploit fact that human can recover from small errors

Real Time Protocol

Specifies packet structure for packets carrying mulitmedia

- Push type (UDP)
- On top of UDP, usually in transport layer
- Payload type id, packet sequence numbering, time stamps
- Come in set of 3



multimedia chunk with RTP header makes RTP packet, encapsulated in UDP segment. **RTP doesn't guarentee quality of service**. Only seen at ends, not routers.

- payload type(7 bits) - indicates type of encoding used
- sequence (16 bits) - use to detect packet loss, cumulative
- timestamp (32 bits) - sampling instance of first byte in current packet. Clock increases by one for each sampling period. If have x, then clock increases by x.
- SSRC field (32 bits) - identifies source of RTP stream. Use to differentiate between different client instance.

RTP used in WebRTC, present in most browsers. Benefits :

- Short end-to-end latency

Challenges

- Special purpose server for media
- Protocols use TCP and UDP, may not pass through firewalls if UDP used
- Difficult to cache data

Session Initiation Protocol

Goal : reach callee independent of the type of network and device they are on.

- provide mechanism to set up call - something like tcp handshake
- determine current ip address of callee
- add new media stream during call, multiple features to the call

Dynamic Adaptive Streaming over HTTP

Solves the problem of sending HTTP request to get the entire file at one go. Can be wasteful and need large client buffer. **Idea : Use HTTP to stream, break media up into smaller chunks**. Benefits :

- Server is simple - scalable
- No firewall issues - uses TCP
- Standard web caching works

Disadvantages :

- Since segments can be long, doesn't provide low latency for interactivity because of buffering

How DASH works

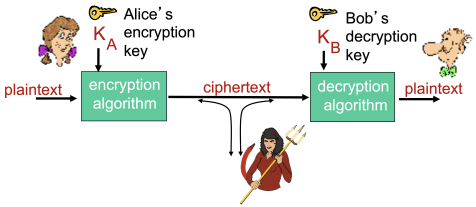
- Create multiple quality of file
- Server provide a playlist that lists all streamlets and their qualities (like a menu).
- Client download menu
- Based on adaptive bitrate algorithm, determine which segment to download based on bandwidht.

Network Security

- Confidentiality : sender and intended receiver only, protect message content
- Authentication : confirmation of identity
- Message Integrity : no tempering of message
- Access and availability : services must be accessible and available to users

Cryptography

- Message and encryption key passes into encryption algorithm
- Encryption algorithm ouputs cypher text, hopefully attackers cannot decrypt this
- Text goes through a channel to reach end-user
- End-user inputs contents and decryption key to output the origin text.



Types of Attacks

Attacker only has the ciphertext

- Brute force - search all keys
- Statistical analysis - reduce search space using some stats

Attacker has plaintext that corresponds to ciphertext. Attacker can get ciphertext from plaintext

Symmetric Key Cryptography

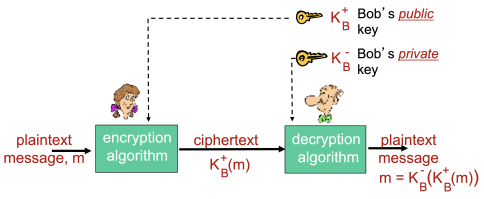
Application of same key for decryption and encryption. Both ends need to agree on key.
Encryption Schemes :

- Substitution cipher : substitute one thing for another. Static mapping (same letter always have same mapping), and key may be very long (is the mapping itself).
- Caesar cipher : shift the letters by fixed amount, wrapping around. Key is just the shift number, still static mapping
- Multiple mappings : decide on order to use mappings. Key will be the mappings involved and the order which they are used. Same letter have different encryption now.
- Data Encryption Standard : 56-bit key, 64 bit input
- Advance Encryption Standard : 128, 192, 256 bit keys. Process data in 128 bit blocks.

Public Key Cryptography (RSA)

Don't have to agree on key. Idea of public and private key. Public key will be known to **everyone**. Private key only to **receiver**. If private key is K^- and public is K^+, Requirements :

- Keys have to work such that if a message m is passed, $K^-(K^+(m)) = m$
- Impossible to derive private key from public key



Message is a bit pattern which can be uniquely represented by integer number. Algorithm for encryption :

- choose 2 large prime numbers that are like 1024 bits each, p, q
- compute $n = pq$, $z = (p-1)(q-1)$
- choose e where $e < n$ that has no common factors with z
- choose d such that $ed - 1$ is exactly divisible by z
- public key would be (n, e), private key would be (n, d)
- Encryption : $c = m^e \bmod n$, decryption : $m = c^d \bmod n$

Uses fact that $(a \bmod n)^d \bmod n = a^d \bmod n$.
Order does not matter - use public then private or private then public - follows modular arithmetic. RSA is secure because it makes use of factorisation, which is hard.

Session Keys

Use RSA to exchange symmetric keys. Once both have the keys, just use symmetric key cryptography. RSA key is valid only for 1 session. Since RSA is secure, can be used to create a secure connection to exchange keys.

Digital Signatures

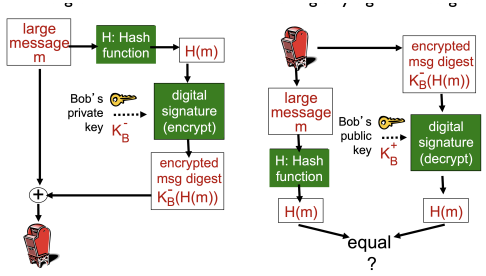
Verifiable, nonforeable, use to prove signee. Show that signee is ok with the document. **Document readable by anyone**. Encrypted message is the signature in this case. Resulting output would be : **signature, original message**. Receipient then apply **public key on signature to decrypt** it, then compare the original message with the decrypted message. If they are the same, then whoever encrypted the message **must have used the respective private key**.

Message Digest

Expensive to encrypt long messages. Instead, use a hash function to produce a fixed size message digest. Hash function properties :

- Many-to-one
- produces fixed size message digest (fingerprint)
- not possible to find original message given the hash
- unique for each message

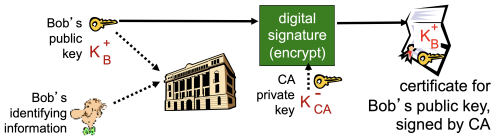
Why not use checksum - same sequence with swapped characters give same checsum. Instead of sending large messsage, send the resulting hash with the digital signature instead. Receiver now compares the hash instead.



Application : password hashing. Instead of storing password, store hash instead. Compare hashes to determine whether password is correct.

Public Key Certification

Public key must be really really public, put everywhere for everyone to see. Verified using **Certification Authority**. Certification Authority binds public key to particular entity. CA signs public key to verify it.



Since now the public key of entity is encrypted, use CA public key to decrypt to get back the original public key of the entity.

VPNs

Cheaper alternatives for creating private networks for security. Creates an encrypted channel between 2 locations. Dedicated like a tunnel. **Can be same subnet now** and use the same services on that specific subnet.

- IP datagram to router
- Router adds VPN header to datagram
- Upon reaching the desired router with same VPN header
- Router restore original IP datagram

Firewalls

Gateway that limits traffic. Isolate organisation internal net from larger internet, allowing some packets to pass and block others.

- Prevent denial of service attacks - attackers hog TCP ports, no resource left for meaningful connections
- prevent illegal modification/access of internal data
- allow only authorised access to inside network

Types of firewalls :

- Stateless packet filtering - firewall look at every packet, base on some rule to make filter
- Stateful packet filtering - tracks status of entire connection instead. Draw conclusion from different packets, more complex rules.
- Application Gateway - specific to applications. Clients need to be reconfigured to appropriate gateway to use the application