CS2105 Lecture Notes

[2020-01-21 Tue]

Contents

1 Introduction

1.1 Internet

Components to the internet:

- Hosts/end systems (i.e. end nodes): *clients* and *servers*
- Communication links (i.e. edges)
- Packet switches (i.e. internal nodes): *routers* and *switches*

Network edge: made out of hosts

- Access network: network that physically connects hosts to the first router (edge router)

Network core: mesh of interconnected routers, that forward packets from source to destination

1.2 Concepts

Link bandwidth = Transmission rate of link

Packet transmission delay = $\frac{L\ (bits)}{R\ (bits/sec)}$, packet of L bits and transmission rate R bps

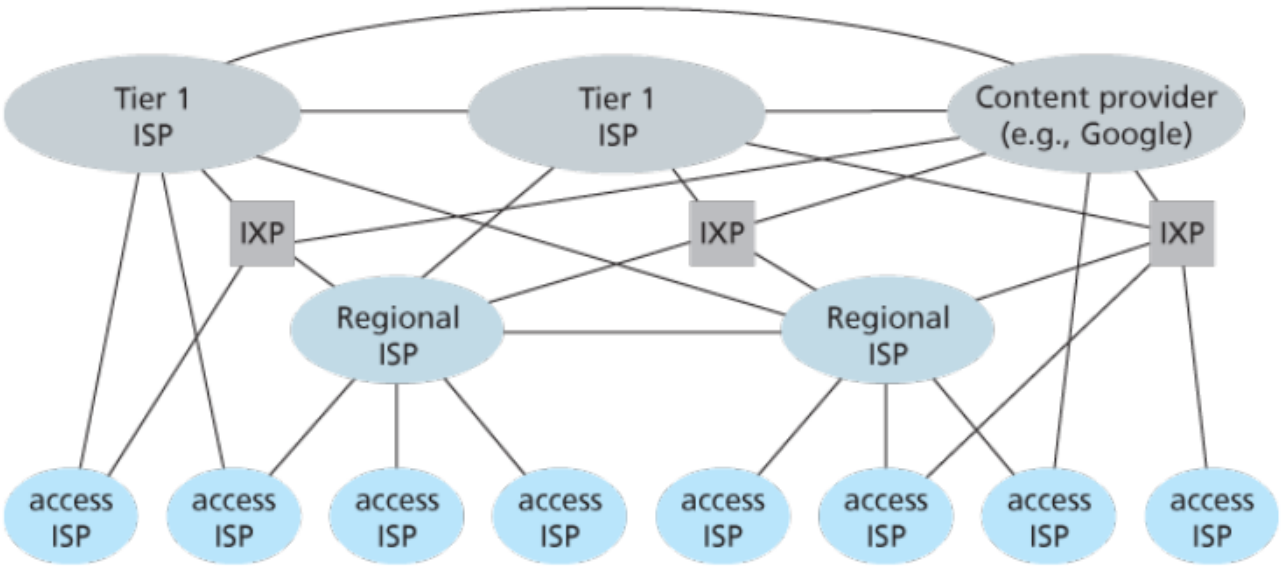
End-end delay = $\frac{NL}{R}$: one-directional movement with N switches in between (assuming no propagation delay)

- Store-and-forward: entire packet must arrive at router before being transmitted

Routing: determining the source-destination route taken by packets

Forwarding: moving packets from a router's input to output

1.3 Internet Structure



- Access ISPs: connected to hosts
- Regional ISPs: connect access nets to one another
- Global ISPs: connected to one another via peering links, IXPs (Internet Exchange Points)

1.4 Delay, Loss, Throughput

Queueing: when arrival rate exceeds transmission rate of link

Delay: when packets have to wait in a queue

Loss: when buffer fills up at switch, so packets are dropped

Throughout: rate at which bits are transferred between sender and receiver

- Note: throughput is how much is being transferred, bandwidth is the theoretical upper bound!

Sources of packet delay: delay $= d_{proc} + d_{queue} + d_{trans} + d_{drop}$

- Nodal processing d_{proc}: check bit errors and determine output link
- Queueing delay d_{queue}: depends on congestion level, buffer size
- Transmission delay $d_{trans} = \frac{L}{R}$ for packet length L bits and transmission rate/link bandwidth R bps: time to push out packet
- Propagation delay $d_{prop} = \frac{d}{s}$ for physical link length d and propagation speed s: time to travel from one end of link to another

1.5 Protocol Layers and Service Models (Overview)

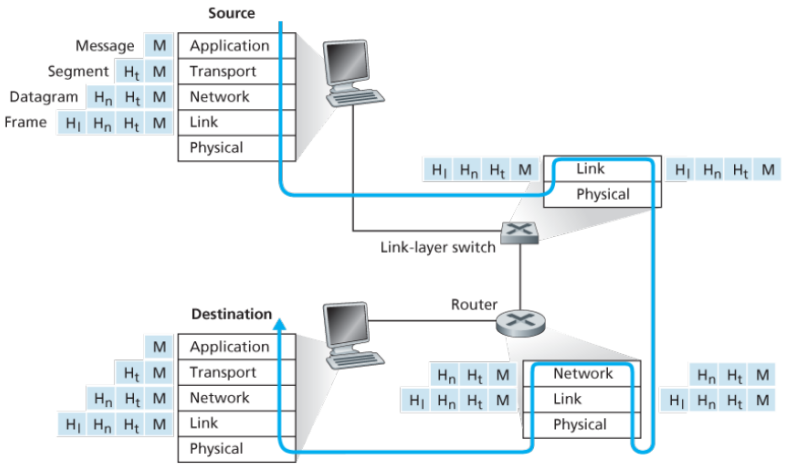
Protocol defines message format, order of messages sent and received, and rules on sending and receiving messages

Protocol layers: each layer implements a service, and relies on lower layers to provide their services

Internet Protocol Stack

- Switch: link + physical
- Router: *network* + link + physical

Network Layer	Network Unit
Application	Message
Transport	Segment
Network	Datagram
Link	Frame
Physical	



2 Application Layer

Client-server architecture: server is always on, with permanent IP address; client communicates with server, can have dynamic IP address

Socket: an abstraction for application processes to communicate with one another over a network

- Identifier for a process: **IP address + port number** (!)

2.1 HTTP (Hypertext Transfer Protocol)

HTTP is the web's application layer protocol

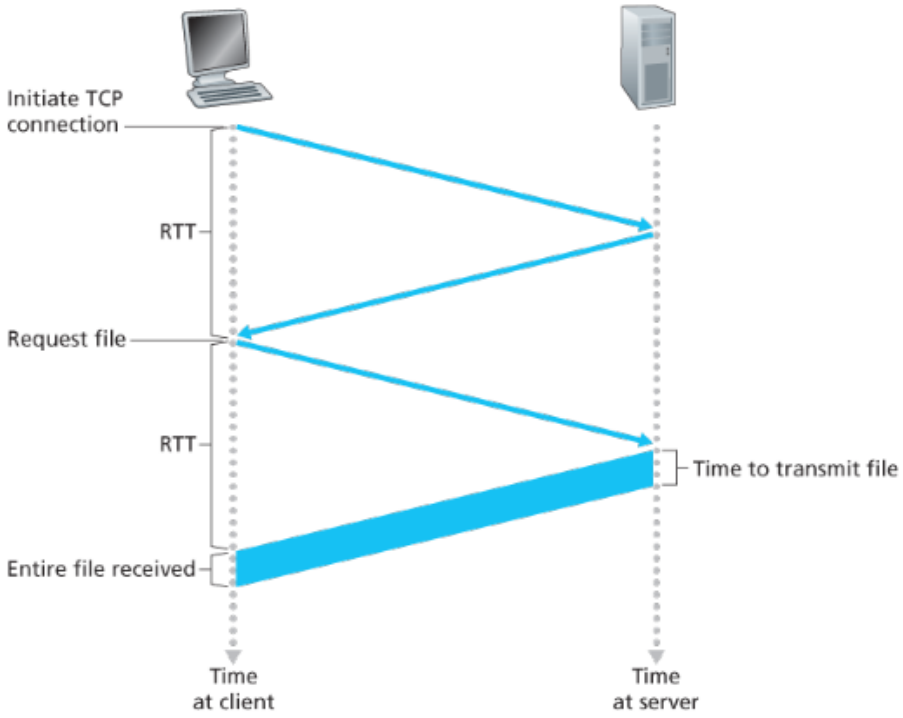
- Uses the client-server model
- Web pages consist of objects (e.g. HTML file, JPEG image, audio file); each object is addressable by URL (host name + path name)
- TCP port 80
- Stateless protocol (unlike TCP): does not keep information about past requests

2.1.1 Persistent vs. Non-Persistent HTTP

`Connection: keep-alive` requests for a persistent connection

Non-persistent: 1 object sent per TCP connection. Default for HTTP 1.0

- HTTP response time: 2×RTT + file transmission time



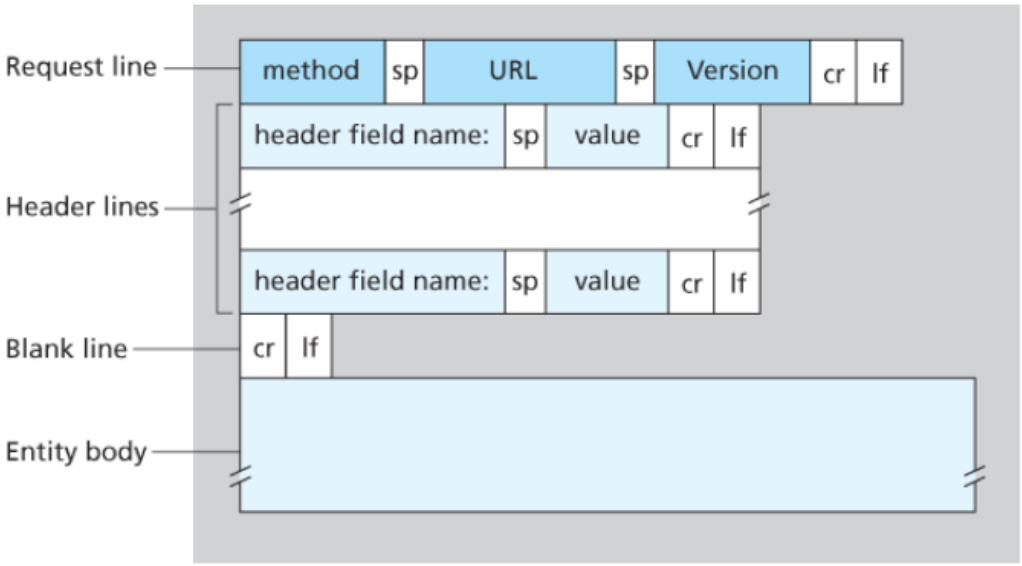
Persistent: multiple objects can be sent per TCP connection. Default for HTTP 1.1

- As little as 1 RTT per referenced object

2.1.2 HTTP request message

In ASCII. Each line ends with `\r\n`

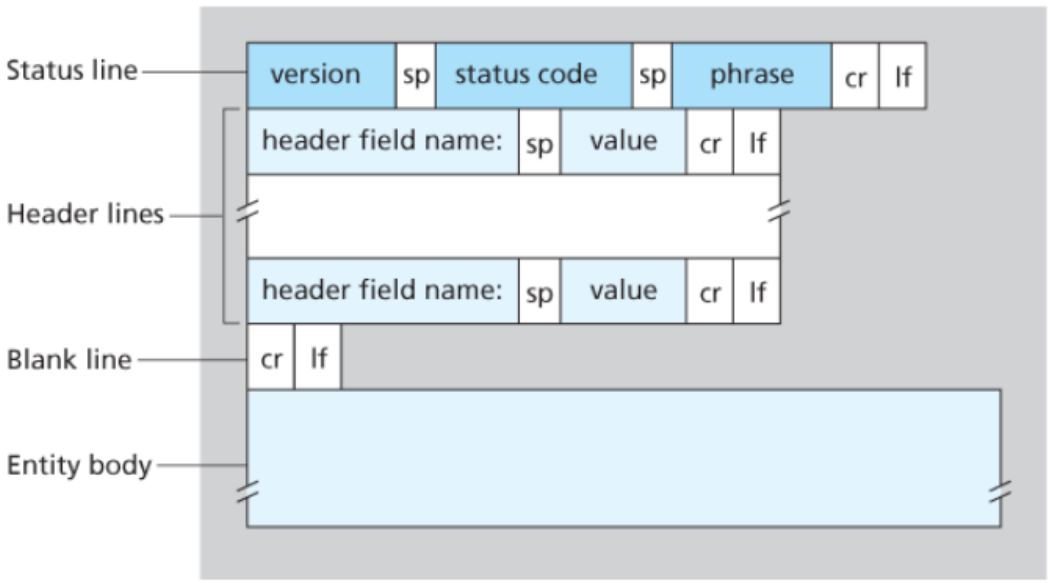
Method types: `GET`, `POST`, `HEAD`, `PUT` (1.1), `DELETE` (1.1)




```
GET /somedir/page.html HTTP/1.1\r\n   # request line
Host: www.someschool.edu\r\n          # header lines
Connection: close\r\n                 # ...
User-agent: Mozilla/5.0\r\n           # ...
Accept-language: fr\r\n               # ...
\r\n
<...body...>                          # body
```

2.1.3 HTTP response message

Status codes: `200 OK`, `301 Moved Permanently`, `400 Bad Request`, `404 Not Found`, `505 HTTP Version Not Supported`, ...



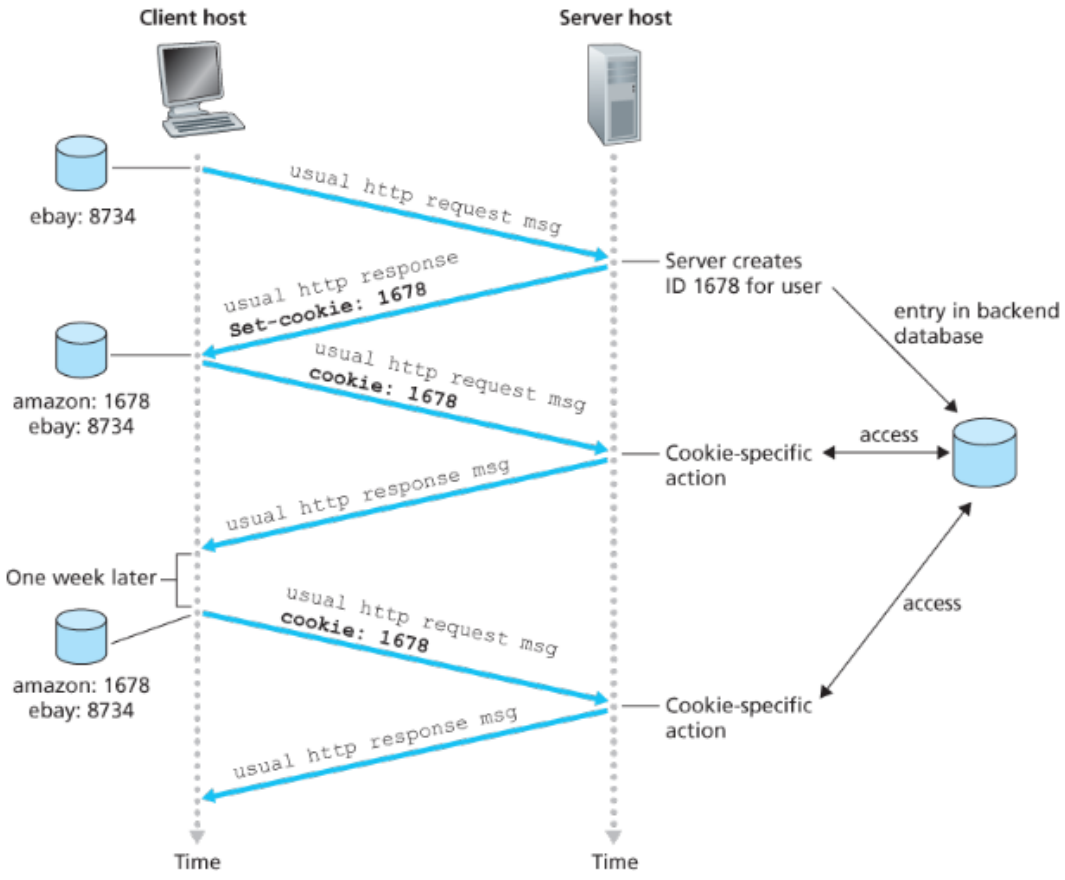
```
HTTP/1.1 200 OK\r\n                                    # status line
Connection: close\r\n                                  # header lines
Date: Tue, 18 Aug 2015 15:44:04 GMT\r\n                # ...
Server: Apache/2.2.3 (CentOS)\r\n                      # ...
Last-Modified: Tue, 18 Aug 2015 15:11:03 GMT\r\n       # ...
Content-Length: 6821\r\n                               # in BYTES (of the body only)
Content-Type: text/html\r\n                            # ...
\r\n
<...data...>                                           # data
```

2.1.4 Cookies

Cookies help to store state

Components to cookies

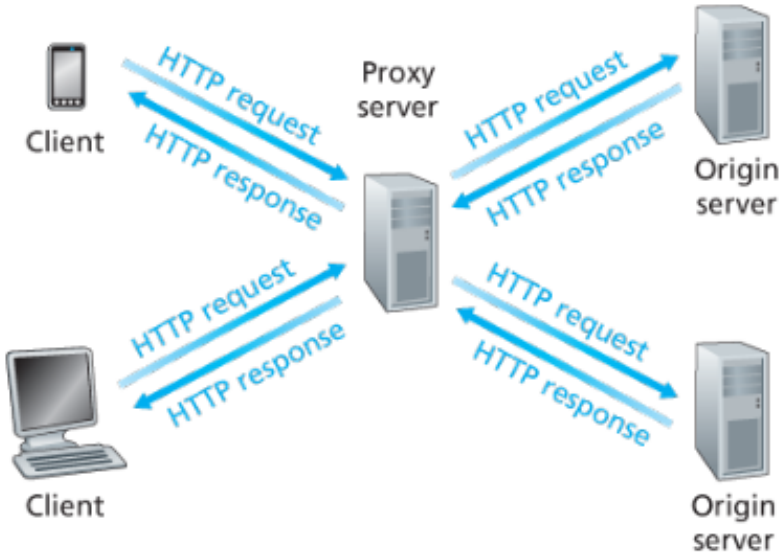
1. Cookie header line in HTTP response
2. Cookie header line in the following HTTP request
3. Cookie file kept on client's host (managed by browser)
4. Server website has backend database to store cookie entries



2.1.5 Web Caches/Proxy Server

Web cache/proxy server: satisfies HTTP requests on behalf of an origin web server, keeps track of recently requested objects

- If object in cache, cache returns object
- Otherwise, cache request object from origin, and returns it to client



Conditional GET

- Problem: what if origin server updates their copy of an object? Then cache's object might be stale
- Solution: cache sends *conditional GET* to verify it's up-to-date: it's a `GET` with `If-Modified-Since:` header line

2.2 DNS (Domain Name System)

DNS features

- Goal: translate by giving mappings between domain name and IP address.
- UDP port 53 (possible to use TCP, esp. with larger packet sizes)

DNS is both:

1. A *distributed* database implemented in a *hierarchy* of DNS servers
2. An application-layer protocol that allows hosts to query the distributed database

2.2.1 Local DNS server

When host makes DNS query, it is first sent to local DNS server (cache) provided by ISP

- If found, simply returns it
- Otherwise, forwards the query into hierarchy, starting with root DNS server

(Often, local name servers cache TLD servers, so don't need visit root name servers)

2.2.2 Distributed, Hierarchical Database

3 classes of DNS servers

- Root DNS servers: (400+ worldwide)

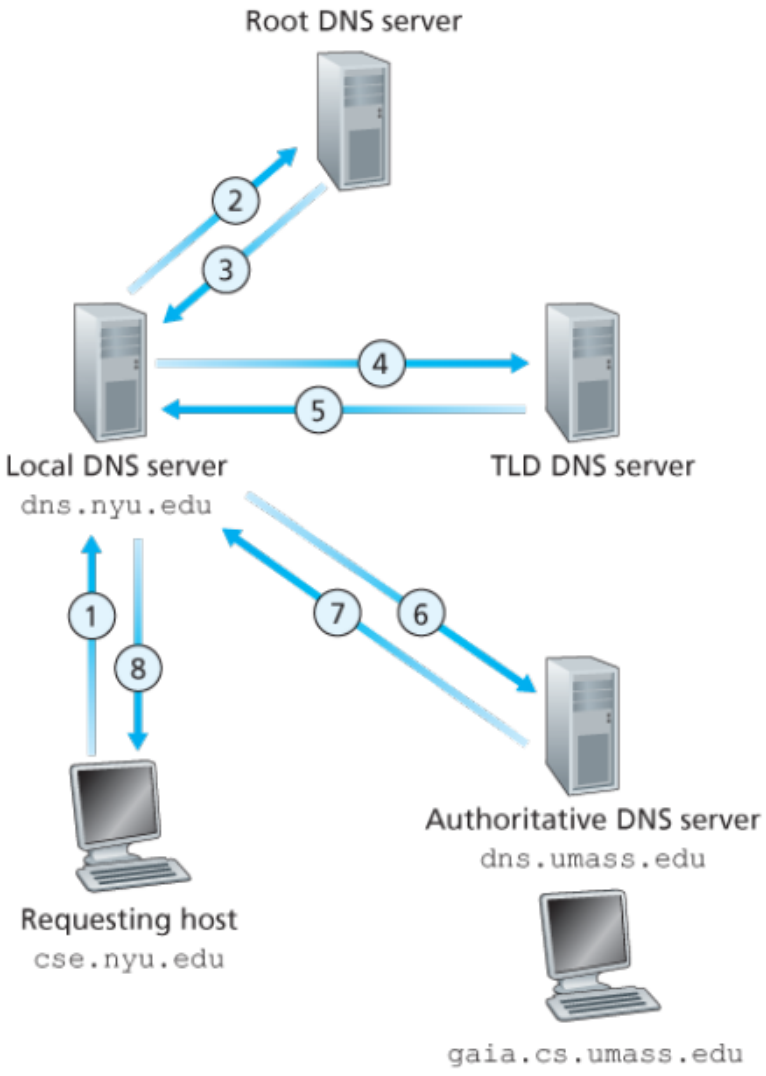
- Top-level domain (TLD) DNS servers: (e.g. com, org, net, edu)
- Authoritative DNS servers (organisation's own DNS server, provides the actual mapping)

2.2.3 Caching and Updating

Mapping is cached by DNS server when it learns it

- Time-to-live (TTL): cache entries timeout after TTL (~2 days?)
- Cache entries might be out-of-date if IP address changes before all TTLs expire worldwide

2.2.4 Iterative vs. Recursive Queries



- Iterative: contacted DNS server replies with the name of a lower-level server to contact next
- Recursive: contacted DNS server handles all the work on behalf of client, and returns actual mapping
- Typically, local DNS server contacts DNS servers that handle the query *iteratively*, before returning mapping to requesting host *recursively*

3 Socket Programming

Socket: interface between application process and end-to-end transport protocol (TCP/UDP)

3.1 Identifiers

($\star$) To identify a process, we need **IP address + port number**!

Port number: 16-bit integer (1-1023 reserved for standard use)

3.2 UDP: unreliable datagram, connectionless

Server has 1 socket to receive ALL clients.

- Client: attaches IP destination address + port number *to every packet*
- Server: extracts IP sender address + port number from every packet

UDP is unreliable, so data might be lost or received out of order

3.2.1 UDP Client

```
from socket import *
serverName = 'hostname'
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_DGRAM)
message = ...
clientSocket.sendto(message.encode(), (serverName, serverPort))
receivedMessage, serverAddress = clientSocket.recvfrom(2048) # buffer size
print(receivedMessage.decode())
clientSocket.close()
```

3.2.2 UDP Server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))

while True:
    message, clientAddress = serverSocket.recvfrom(2048)
    outMessage = message.decode().upper()
    serverSocket.sendto(outMessage.encode(), clientAddress)
```

3.3 TCP: reliable byte stream, connection-oriented

Server has 1 welcome socket for ALL clients, but for communications, 1 socket for EACH client.

- Client: creates socket, specifying IP destination address + port number

- Server: creates *welcome* socket (port 80); when contacted by client, create new *connection* socket to communicate

3.3.1 TCP Client

```
from socket import *
serverName = 'hostName'
serverPort = 12000

clientSocket = socket(AF_INET, SOCK_STREAM)
clientSocket.connect((serverName, serverPort))

message = ...
clientSocket.send(message.encode())
receivedMessage = clientSocket.recv(1024)
print(receivedMessage.decode())
clientSocket.close()
```

3.3.2 TCP Server

```
from socket import *
serverPort = 12000

serverSocket = socket(AF_INET, SOCK_STREAM)
serverSocket.bind(('', serverPort))
serverSocket.listen(1) # welcome socket: max number of queued connections

while True:
    connectionSocket, address = serverSocket.accept()
    message = connectionSocket.recv(1024).decode()
    outMessage = message.upper()
    connectionSocket.send(outMessage.encode())
    connectionSocket.close()
```

4 Transport Services and Protocols

Transport layer: logical communication between *processes* (runs only in hosts)

- Sender: breaks application layer messages into segments, passes down to network layer
- Receiver: reassembles segments into messages, passes up to application layer

Network layer: logical communication between *hosts*

- Unreliable "best effort"

4.1 TCP vs. UDP

TCP	UDP
Connection-oriented	Connectionless
Flow control	No flow control
Congestion control	No congestion control

Application	Application layer protocol	Transport protocol
Email	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Streaming multimedia	HTTP, RTP	TCP/UDP
Internet telephony	SIP, RTP, proprietary	TCP/UDP

5 UDP (User Datagram Protocol)

UDP adds very little on top of IP, it's still unreliable

- Connectionless multiplexing/de-multiplexing (deliver data from hosts to processes based on port number)
- Checksum

5.1 Connectionless De-Multiplexing

Use *port number* to differentiate between processes. Segments with same destination IP + port number will always be directed to same socket, even from a different source IP

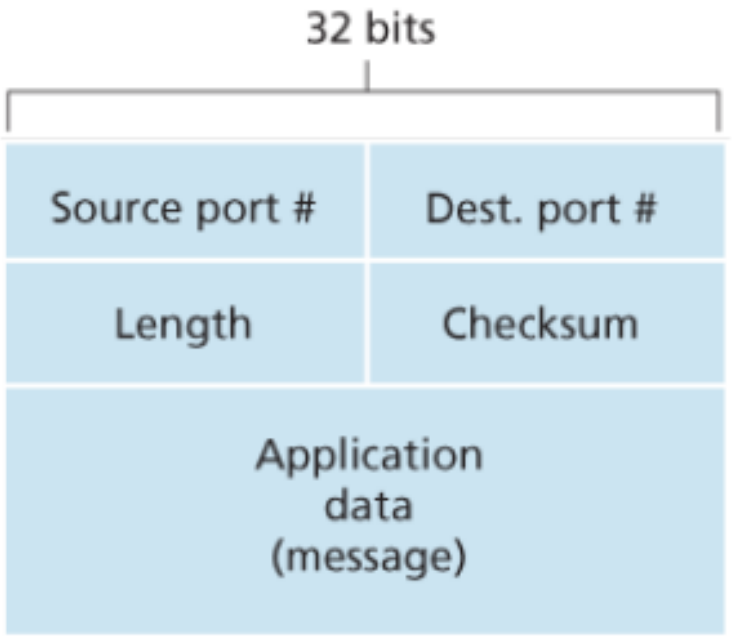
5.2 Checksum

- Treat UDP segment as sequence of 16-bit integers
- Binary addition of every 16-bit integer, *add carry to result*
- Checksum = 1s complement of added number

5.3 UDP Segments

Header size: 64 bits (8 bytes)

- Length field is in *bytes*, length of *ENTIRE* UDP segment



6 Principles of Reliable Data Transfer

Possible errors over an unreliable channel:

- Bit errors in data/feedback
- Packet loss in data/feedback

6.1 RDT 1.0: Totally reliable

Totally reliable channel: simple protocol

- Sender: wait for data from layer above → send across channel
- Receiver: wait for data from channel → extract and push to layer above

6.2 RDT 2.0: Corruption in data

Bit errors in data sent: use checksum to detect errors, along with ACK/NAK

Receiver

- Receiver gets good data → send ACK
- Receiver gets corrupt data → send NAK

Sender

- Sender gets ACK → sends next packet
- Sender gets NAK → sends same packet

6.3 RDT 2.1: Corruption in data + ACK/NAK

Bit errors in acknowledgement too: use checksum on acknowledgement too; introduce alternating seq# 0/1

Receiver

- Receiver gets good data with prev seq# → send ACK and wait for same seq#
- Receiver gets good data with same seq# → send ACK and wait for next seq#
- Receiver gets corrupt data → send NAK and wait for same seq#

Sender

- Sender gets good ACK → move on to send next packet with next seq#
- Sender gets corrupt response or NAK → resend old packet with old seq#

6.4 RDT 2.2: NAK-free protocol

Idea: Instead of sending NAK, send ACK of most recent good packet!

Receiver

- Receiver gets good data with same seq# → sent ACK of same seq#
- Receiver gets corrupt data or prev seq# → send ACK of prev seq#

Sender

- Sender gets good ACK with same seq# → move on to send next packet with next seq#
- Sender gets corrupt ACK or prev seq# → resend old packet with old seq#

6.5 RDT 3.0: Corruption and Loss

Idea: to deal with packet loss, introduce a *timer*: sender retransmits if it times out; resend ONLY when timeout occurs (not when receiving old ACK anymore)

- Corner case: the packet is not lost, just delayed
- Problem: may have low *utilisation* (fraction of time the sender is busy sending)
- $U_{sender} = \frac{L/R}{RTT+L/R}$ where $D_{trans} = L/R$
- Solution: introduce *pipelining*, where sender can send multiple unacknowledged packets at once

6.6 Pipelining: Go-Back-N

Sliding window of size N consecutive un-ACKed packets

- ONLY 1 timer
- Cumulative ACKs
- Sender remembers ONLY send base; upon timeout, retransmits the send base and all higher seq# packets in window
- Receiver remembers ONLY the expected seq# and *discards* any packet that does not have expected seq#; acknowledges ONLY the expected seq#

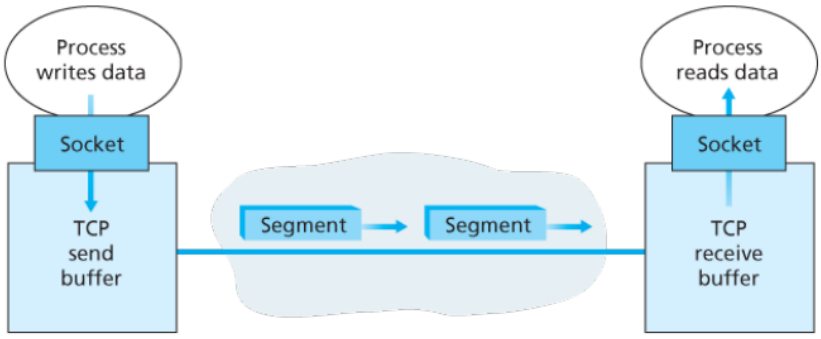
6.7 Pipelining: Selective Repeat

Sliding window of size N consecutive un-ACKed packets

- 1 timer PER un-ACKed packet
- Non-cumulative (individual) ACKs
- Sender remembers send base, and status and timer for EACH un-ACKed packet; upon timeout, retransmits the offending packet ONLY
- Receiver remembers recv base, and buffers out-of-order packets; acknowledges EACH correctly received packet individually

7 TCP (Transmission Control Protocol)

- Point-to-point: one sender and receiver only
- Connection-oriented
- Full *duplex* data: bi-directional data flow in a connection
- Reliable, in-order byte stream
- Pipelined: sliding window size is set dynamically from congestion/flow control



($\star$) Send and receive buffers are created after handshaking on **BOTH** sides! $\rightarrow$ bi-directional data transfer

Maximum segment size (MSS): 1460 bytes = 1500 – 40 (size of TCP+IP headers)

- Limited by maximum transmission unit (MTU), the largest link-layer frame (e.g. 1500 bytes for Ethernet)

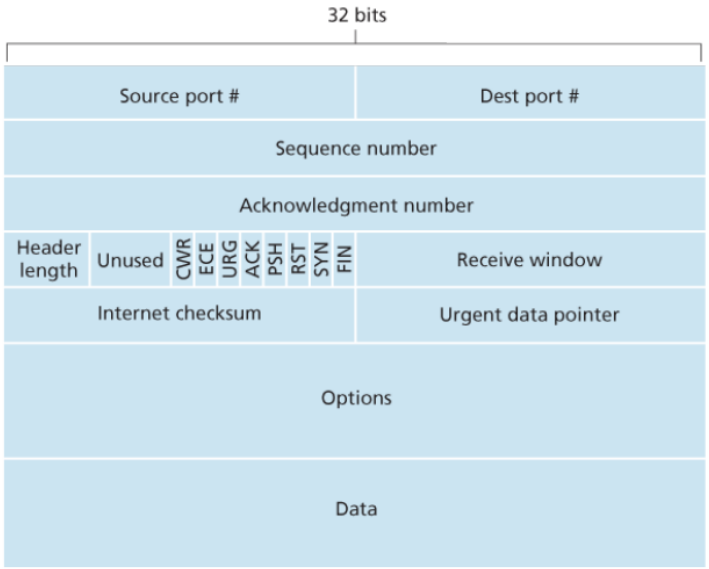
7.1 Connection-Oriented De-Multiplexing

How to figure out which socket to send to?

- TCP socket identified by *4-tuple*: (source IP, source port, dest IP, dest port)

7.2 TCP Segments

Header: typically 20 bytes (can be more)



7.2.1 Sequence/Acknowledgement Number

TCP works on *cumulative ACK*

- Sequence number: *byte stream index* of first byte in segment's data
- Acknowledgement number: sequence number of next byte expected from the other side

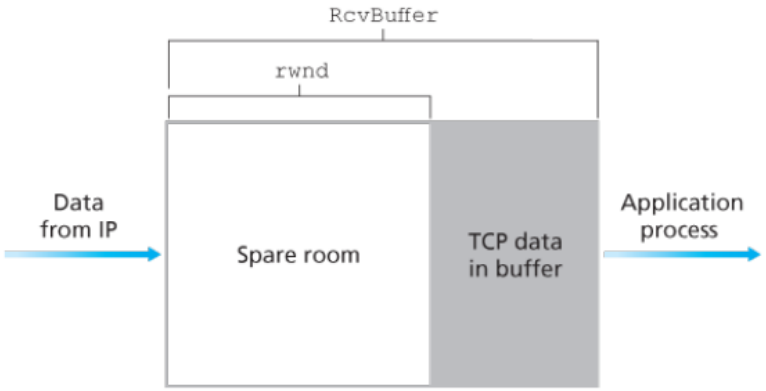
How does receiver deal with out-of-order segments?

- Up to them! Can either buffer or not buffer—just return the next expected sequence number

7.2.2 Receive Window

Receive Window (rwnd): number of bytes receiver is willing to accept

- rwnd = size of free buffer space
- For *flow control*: ensure receiver buffer doesn't overflow



7.2.3 Other components

Header length and options: TCP header can extend beyond 20 bytes to include options, where header length specifies the total length of TCP header. But this isn't really used in practice

- Header length is 4 bits long, represents size of entire header in multiples of 4 bytes
- Typical value is 5 (for header size = 20 bytes), largest value is 15 (for header size = 60 bytes)

Bits

- (×) `URG`: urgent data
- (⋆) `ACK`: acknowledgement
- (×) `PSH`: push data now (gets up to application layer ASAP)
- (⋆) `RST`: reset (server tells client that the socket/process is not for use)
- (⋆) `SYN`: synchronisation
- (⋆) `FIN`: finish

7.3 TCP Connection Management

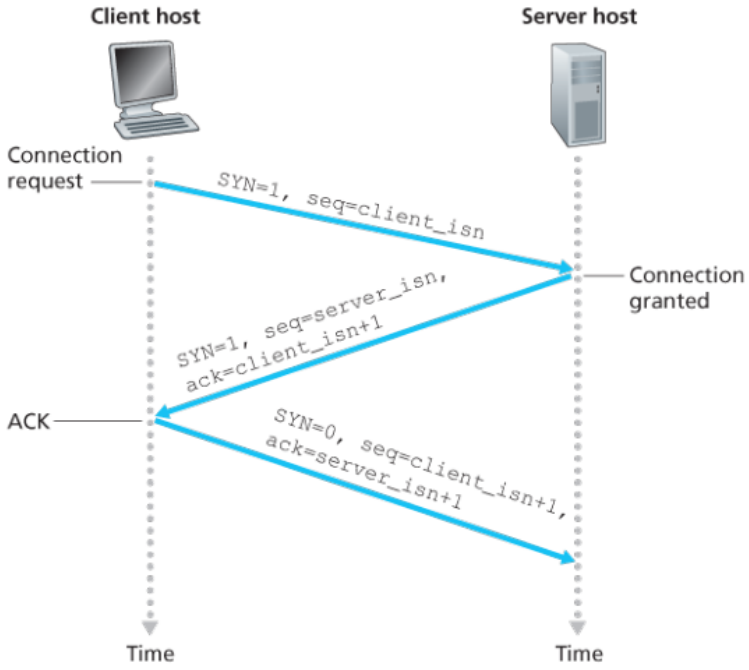
7.3.1 3-Way Handshake

Handshake: agree to establish connection, and agree on connection parameters (e.g. both initial sequence numbers)

- `SYN` →
- ← `SYNACK`
- `ACK` →

States

- Client: `LISTEN` → `SYN_SENT` → `ESTAB`
- Server: `LISTEN` → `SYN_RCVD` → `ESTAB`



*Note: can send data in the final `ACK`!

7.3.2 Closing a Connection

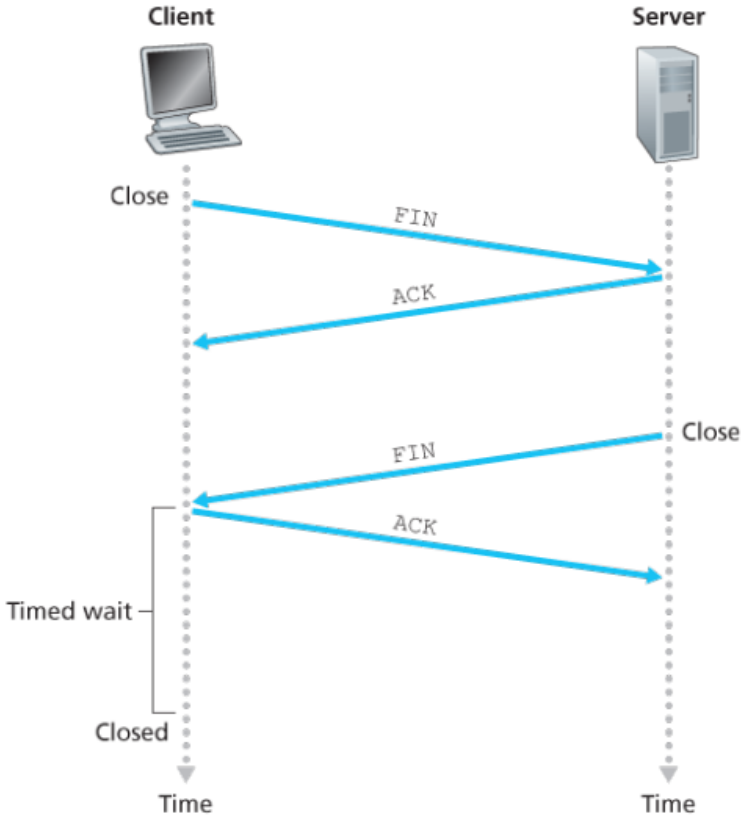
Closing a connection: timed wait by client = 2 × max segment lifetime (`ACK` and `FIN` can be combined into `FINACK`)

- `FIN` →
- ← `ACK`
- <data...>
- ← `FIN`
- `ACK` →

States

- Client: `ESTAB` → `FIN_WAIT_1` → `FIN_WAIT_2` → `TIMED_WAIT` → `CLOSED`

- Server: `ESTAB` → `CLOSE_WAIT` → `LAST_ACK` → `CLOSED`



7.4 TCP Reliable Data Transfer

7.4.1 TCP Sender Events

1. Data received from application layer
   - If not already running, start timer
   - Create segment and pass to IP
   - NextSeqNum += length(data)
2. Timeout
   - Retransmit ONLY oldest unACKed packet
   - Restart timer
3. ACK received, ACK value = y
   - If ACK acknowledges previously unACKED segments, update sliding window + start timer if there are still unACKed segments

7.4.2 TCP Receiver Events

1. In-order segment arrives with expected seq#, all up to seq# ACKed
   - Delayed ACK—wait up to 500ms for next segment before sending ACK
2. In-order segment arrives with expected seq#, segment has ACK pending

- Immediately send ONE cumulative ACK for both in-order segments

3. Out-of-order segment arrives with higher-than-expected seq# (there's a gap!)
    - Immediately send duplicate ACK with seq# of next expected byte
4. Segment arrives that fills gap
    - Immediately send ACK if the segment fills up lower end of gap

7.4.3 TCP Timeout Interval

$TimeoutInterval = EstimatedRTT + 4 \cdot DevRTT$

- $EstimatedRTT = (1 - \alpha) \cdot EstimatedRTT + \alpha \cdot SampleRTT$, $\alpha \approx \frac{1}{8}$
- $DevRTT = (1 - \beta) \cdot DevRTT + \beta \cdot |SampleRTT - EstimatedRTT|$, $\beta \approx \frac{1}{4}$
- ($SampleRTT$ measures time from transmitting segment to receiving ACK, ignoring retransmissions)

7.4.4 TCP Fast Retransmit

Idea: sender can detect packet loss before timeout, using duplicate ACKs, so can resend the last packet earlier.

- 4 ACKs for the same sequence number $\rightarrow$ resend the oldest unACKed segment

8 Network Layer

(Note: *routers* go through network layer, switches do not)

8.1 Network Layer Functions

1. Routing: determine route packets should take along path from source to destination
   - Routing algorithms: determine the forwarding table used by routers
2. Forwarding: move packets from router's input link to the appropriate output link

8.2 Planes

1. Control plane: network-wide logic; determines how datagram is *routed*
   - Traditional routing algorithms: implemented in routers
   - Software-defined networking (SDN): implemented in remote servers
2. Data plane: local, per-router function; determines how datagram is *forwarded*

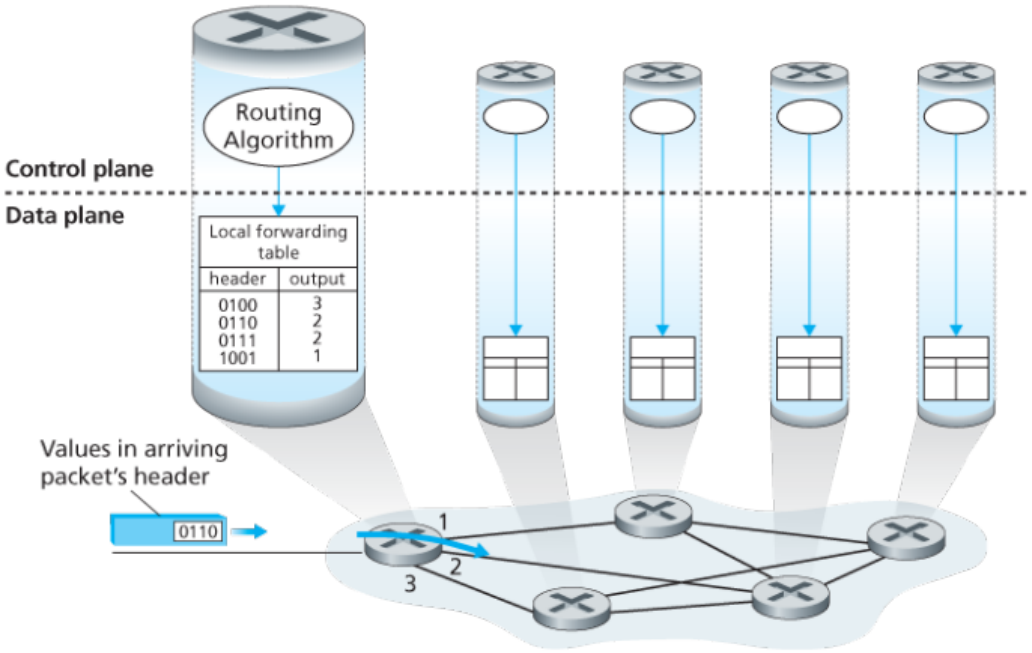


Figure 4.2 Routing algorithms determine values in forward tables

9 IP (Internet Protocol)

9.1 IP Address

IP address: 32-bit identifier for a host/router *interface*

- Each IP address is associated with an interface

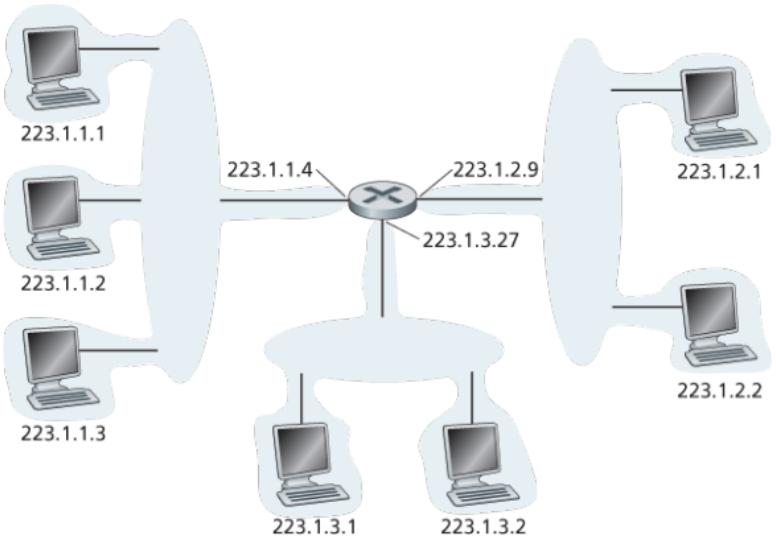
Interface: the connection between host/router and physical link

- Routers typically have multiple interfaces
- Host can have more than one interface! (e.g. wired Ethernet, wireless 802.11)

(How are the interfaces connected?)

- Wired Ethernet interfaces connected by Ethernet switch (used for LANs)
- Wireless WiFi interfaces connected by WiFi base station

9.2 Subnets



Subnet: a network formed by a group of "directly interconnected" hosts

- Directly interconnected: hosts in the same subnet can physically connect with each other without any intervening router; connect to the outside world with a router
- How many subnets are there? Remove the routers → check the number of isolated networks

9.3 CIDR (Classless InterDomain Routing)

CIDR is the internet's address assignment strategy

Two parts to an IP address of form $a.b.c.d/x$

- Network (subnet) prefix: x bits
- Host ID: $32 - x$ bits

When an outside router forwards a datagram to the subnet, only x bits need to be considered → reduces size of forwarding table

9.4 Subnet Mask

Subnet mask: made by setting all network prefix bits to 1, all host ID bits to 0

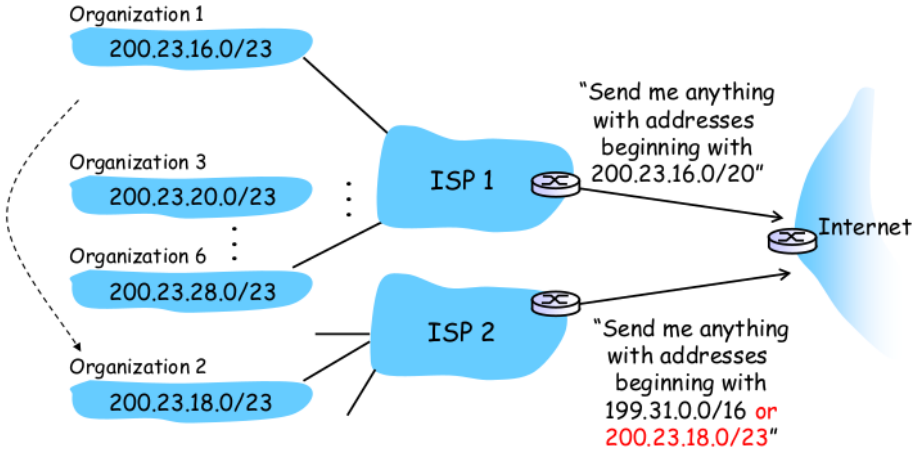
- Bitwise AND operation with IP address → can determine which network it belongs to

9.5 Special IP addresses

IP address	Use
0.0.0.0/8	Non-routable meta-address, for special use
127.0.0.0/8	Loopback address, used for local testing
10.0.0.0/8 (10.255.255.255) 172.16.0.0/12 (172.31.255.255) 192.168.0.0/16 (192.168.255.255)	Private addresses, can be used without any coordination with an Internet registry
255.255.255.255/32	Broadcast address, all hosts on same subnet will receive datagram

9.6 Hierarchical Addressing: Route Aggregation

Hierarchical: each tier is responsible for receiving packets for all its smaller networks, by aggregating their IP addresses



More specific routes: when an organisation changes ISP, the ISP can simply add and advertise another entry to neighbouring routers

- Longest prefix matching (match as much as possible) to deconflict

9.7 Managing IP addresses

ICANN allocates IP addresses, manages global root DNS servers, assign domain names and resolves disputes

Getting IP addresses

- Organisations' block of IP addresses: buy from registry or rent from ISP's address space
- Routers' IP addresses: usually hard-coded manually
- Hosts' IP addresses: dynamically get address from server using *DHCP*

	Binary Address	Decimal Address
ISP's block	11001000 00010111 0001 0000 00000000	200.23.16.0/20
Organization 0	11001000 00010111 0001 0000 00000000	200.23.16.0/23
Organization 1	11001000 00010111 0001 0010 00000000	200.23.18.0/23
Organization 2	11001000 00010111 0001 0100 00000000	200.23.20.0/23
...	...	...
Organization 7	11001000 00010111 0001 1110 00000000	200.23.30.0/23

use 3 bits to differentiate 8 organizations

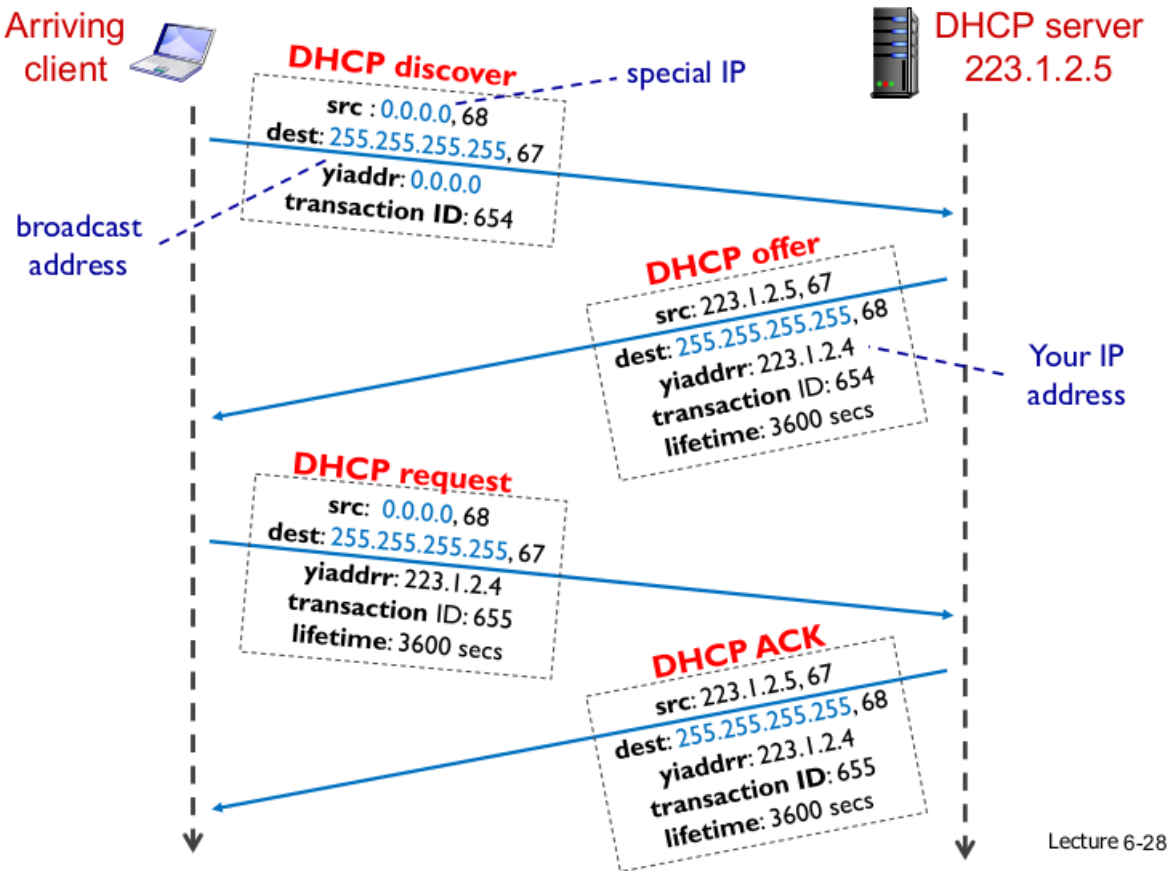
9.8 DHCP (Dynamic Host Configuration Protocol)

DHCP: application layer protocol, where host doesn't even have an IP address (plug-and-play)

- Returns IP address for host
- Returns address of first-hop router for client
- Returns name and IP address of DNS server
- Returns network mask

DHCP runs over *UDP*

- Server: port 67
- Client: port 68

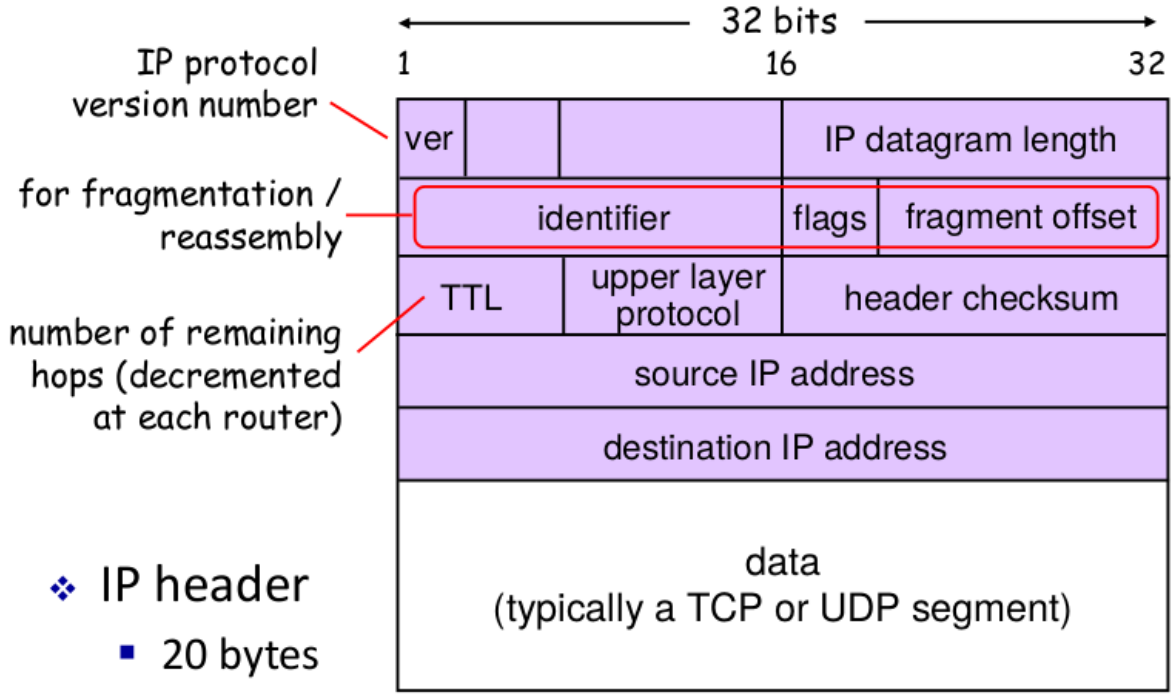


DHCP process: [client uses src 0.0.0.0, dest is always 255.255.255.255]

- DHCP discover →
- ← DHCP offer
- DHCP request →
- ← DHCP ACK

10 IPv4

10.1 IPv4 Datagram Format

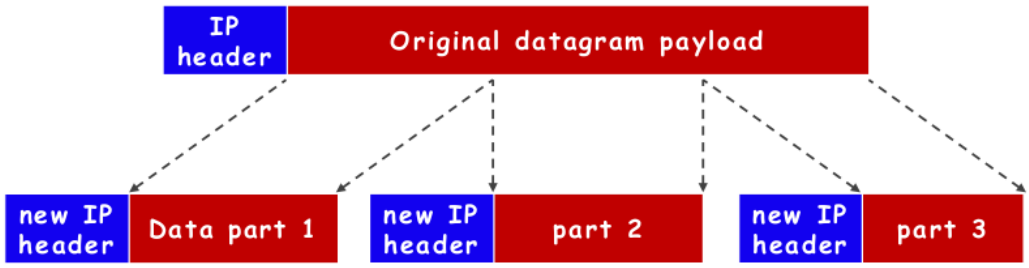


Length refers to total length (e.g. 500 bytes), not data length (e.g. 480 bytes)

10.2 IP Fragmentation and Reassembly

Why need to fragment and reassemble?

- Each link can have different MTU (maximum transfer unit), i.e. maximum amount of data a link-level frame can carry
- If IP datagrams are too large, needs to be fragmented, then reassembled afterwards



Flag (frag flag):

- 1 if there is next fragment from same segment
- 0 if this is the last fragment

Offset: expressed in units of *8 bytes*

	Length	ID	Flag	Offset
Before fragmentation	1200	x	0	0
	500	x	1	0
	500	x	1	60
After fragmentation	240	x	0	120

10.3 Network Address Translation (NAT)

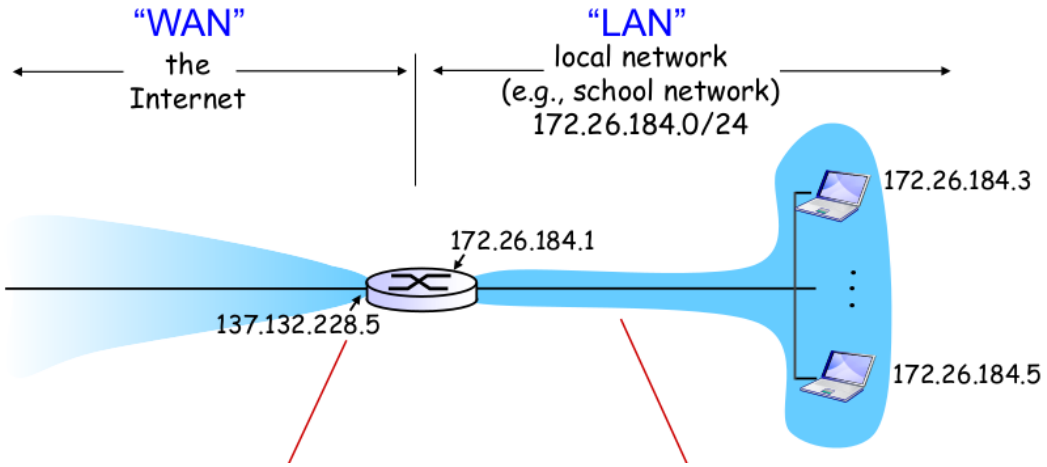
NAT maps one address space into another, commonly used to hide private IP address space behind a single public IP address

WAN: the Internet — use public IP addresses

- All datagrams leaving local network have the *same* source NAT IP address

LAN: local network — use private IP addresses

- Within local network, hosts use private IP addresses for communication



NAT translation table: implemented by NAT routers

- Store mapping from WAN to LAN: (NAT IP address, new port #) ↔ (source IP address, port #)
- Translate outgoing datagrams: source → NAT
- Translate incoming datagrams: NAT → source

WAN side	LAN side
137.132.228.5, 5001	172.26.184.3, 3213
...	...

Benefits of NAT

- Only one public IP for NAT router
- Can change host addresses in local network without affecting outside world
- Can change ISP without changing addresses of hosts in local network
- Security concerns: hosts inside local network are not explicitly addressable and visible to outside world

Challenges of NAT

- Host can reach out to server with public IP address, but host cannot reach out to another private host (because other NAT doesn't know which to send to)
- Peer-to-peer applications don't work directly: might need third-party helper node

11 Routing Algorithms

AS (Autonomous System): Internet is a hierarchy of ASs (eg. ISPs), each controls its own routers and links

Intra-AS routing: finds a path between two routers within an AS

- Commonly used protocols: RIP, OSPF

Inter-AS routing (not covered): handles interfaces between ASs

11.1 Intra-AS routing

Problem: How to find the fastest path between one router and another?

Representation: Graph where vertices are routers, edges are physical links between routers, weights are costs (e.g. congestion level, bandwidth, latency, etc.) $\rightarrow$ shortest path problem

11.2 "Link State" Algorithms (not examinable)

Routers know the entire network topology and all link costs

- Routers periodically broadcast link costs to each other
- Solution: each router runs Dijkstra's algorithm locally

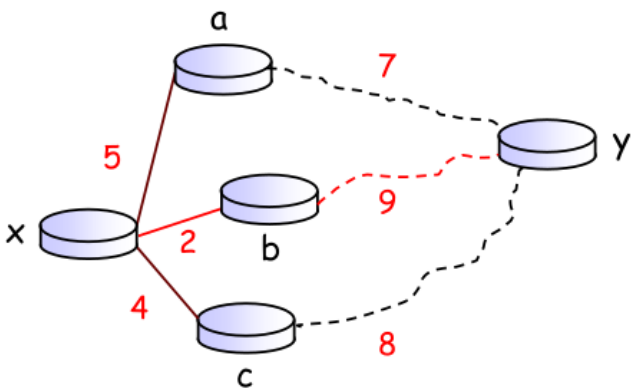
11.3 "Distance Vector" Algorithms

Routers know only their direct neighbours and link costs to neighbours

- Routers exchange "local views" with neighbours, update their own "local views" based on neighbours' views
- Solution: iterative process of computation
  1. Swap local view with direct neighbours
  2. Update own local view
  3. Repeat until no more change to local view

Let $c(x, y)$ be the cost between routers x and y, $d_x(y)$ be the least-cost distance from x to y from x's view.

$$\text{Bellman-Ford equation: } d_x(y) = \min_v\{\ c(x,v) + d_v(y)\ \}$$



Algorithm

- Each router sends its distance vectors to its direct neighbours
- If x finds that y is advertising a cheaper path to z,
  - x will update its distance to z
  - x will note that packets for z should be sent to $y \rightarrow$ used to create x's forwarding table
- After several exchanges, all routers will know least-cost paths to all other hosts

11.4 RIP (Routing Information Protocol)

RIP: implements the Distance Vector algorithm

- Cost = number of hops (insensitive to network congestion)
- Entries in routing table are aggregated subnet masks (so we are routing to destination subnet)
- UDP port 520: exchange routing table every 30s
- "Self-repair": if no update from neighbouring router for 3 minutes, assume neighbour has failed
- Distributed, iterative, asynchronous

11.5 ICMP (Internet Control Message Protocol)

ICMP: used by hosts and routers to communicate network-level information

- Error reporting: unreachable host/network/port/protocol
- Echo request/reply (used by ping)
- ICMP messages carried in IP datagrams: ICMP header starts after IP header

ICMP message format

- *Type*
- *Code*
- *Checksum*
- (Others)

Type	Code	Description
8	0	Echo request (ping)
0	0	Echo reply (ping)
3	1	Destination host unreachable
3	3	Destination port unreachable
11	0	TTL expired
12	0	Bad IP header

- *ping*: checks if a remote host will respond to us. Not always available nowadays
- *traceroute*: sends messages of 1 TTL, 2 TTL, etc. $\rightarrow$ see path of routers

12 Link Layer

Network layer: communication between any two hosts, however many intermediate nodes

Link layer: sends datagram between adjacent nodes over a single link

- IP datagrams are encapsulated in link-layer *frames* for transmission

12.1 Link Layer Services

- Framing: encapsulates datagram in a frame, adding *header* and *trailer*

- Link access control: when multiple nodes *share* a single link, need to coordinate which nodes can send frames at a certain point of time (sort of like scheduling)
- Reliable delivery: some protocols do this: often used on error-prone links (e.g. wireless), but not low bit-error links (e.g. fiber)
- Error detection: errors usually caused by signal attenuation or noise. Receiver detects errors, and depending on protocol, may signal for retransmission or simply drop frame
- Error correction: receiver can identify and correct bit errors without needing retransmission

12.2 Link Layer Implementation

Implemented in hardware: "adapter" (aka NIC) or on a chip

- E.g. ethernet card/chipset, 802.11 card

Routers are semi-autonomous, implementing both link and physical layers

12.3 Error Detection and Correction

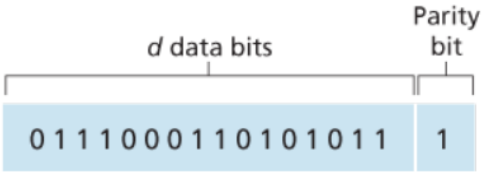
12.3.1 Checksum

Not implemented in link layer usually

12.3.2 Parity Checking

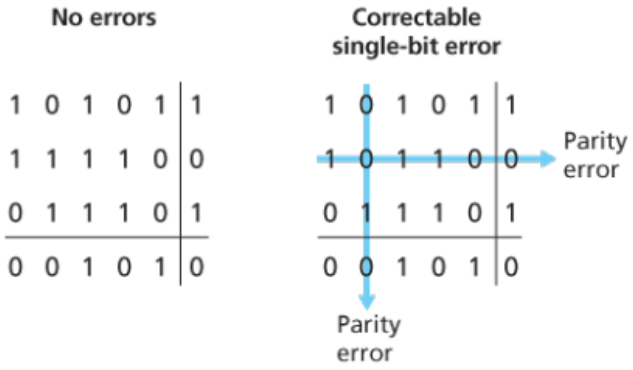
Single bit parity: can detect single bit errors in data

- Even parity: total #1s is even → 0, otherwise 1 (this is the default, just sum total #1s)
- Odd parity: total #1s is odd → 0, otherwise 1



Two-dimensional bit parity: can detect and correct single bit errors in data

- Use the 2D matrix to find row and column of flipped bit



12.3.3 Cyclic Redundancy Check (CRC)

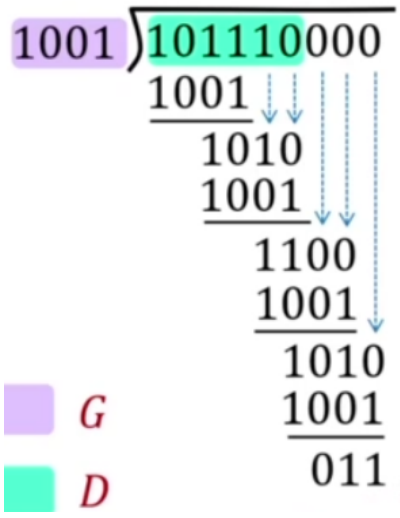
Commonly used in link layer

Idea: divide the data by the generator to get a remainder

- D: data bits, as a binary number
- G: generator of $r+1$ bits, agreed by sender and receiver beforehand
- R: will generate CRC of r bits
- Division is equivalent to XOR in binary
- Sender sends (D, R), receiver divides (D, R) by G (known beforehand) $\rightarrow$ error if remainder is not 0

Example:

- $D = 101110$, $G = 1001$, $r = 3 \rightarrow R = 011$
- Sender sends 101110|011



12.4 Multiple Access Links and Protocols

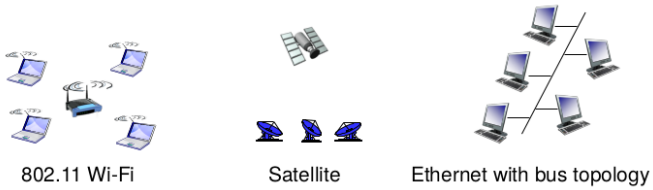
12.4.1 Two Types of Network Links

Point-to-point link: sender and receiver connected by a dedicated link

- Example protocols: Point-to-Point Protocol (PPP), Serial Line Internet Protocol (SLIP)

Broadcast link (shared medium): multiple nodes connected to a shared broadcast channel

- When node transmits a frame, channel broadcasts the frame and every other node receives a copy
- Problem: *collisions* if a node receives multiple frames at the same time
- Solution: *multiple access protocols* (channel partitioning/taking turns/random access)

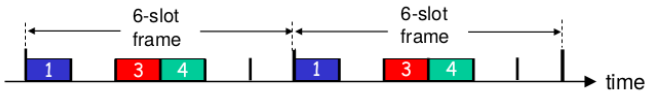


12.4.2 Channel Partitioning Protocols

Divide channel into fixed smaller pieces (e.g. time slots/frequency), each piece allocated to a node exclusively

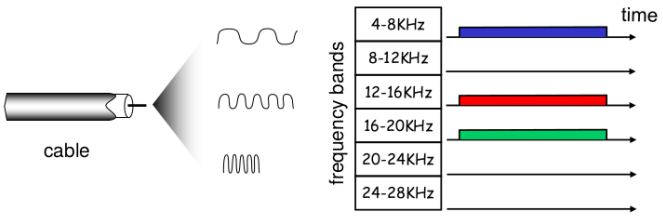
TDMA (Time Division Multiple Access)

- Channel accessed in rounds, where each node gets a fixed time slot
- Unused slots go idle



FDMA (Frequency Division Multiple Access)

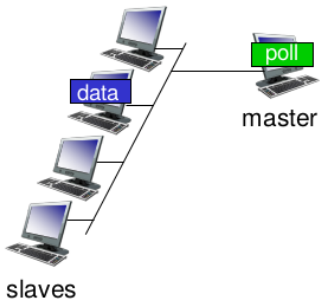
- Channel spectrum divided into frequency bands, where each node gets a fixed frequency band
- Unused transmission time in frequency bands go idle



12.4.3 Taking Turns Protocols

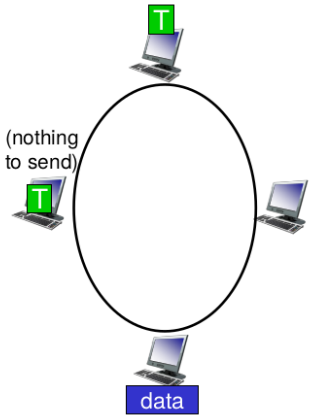
Polling

- Master node invites slave node to transmit in turn
- Concerns: polling overhead (minor), single point of failure on master node



Token Passing

- Control token passed from one node to the next sequentially
- Concerns: token overhead (minor), single point of failure on token

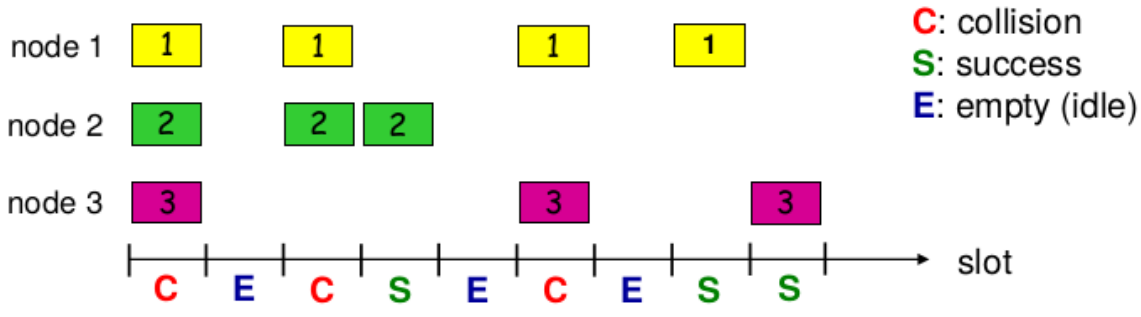


12.4.4 Random Access Protocols

Allow collisions to happen, but *detect* and *recover* from them

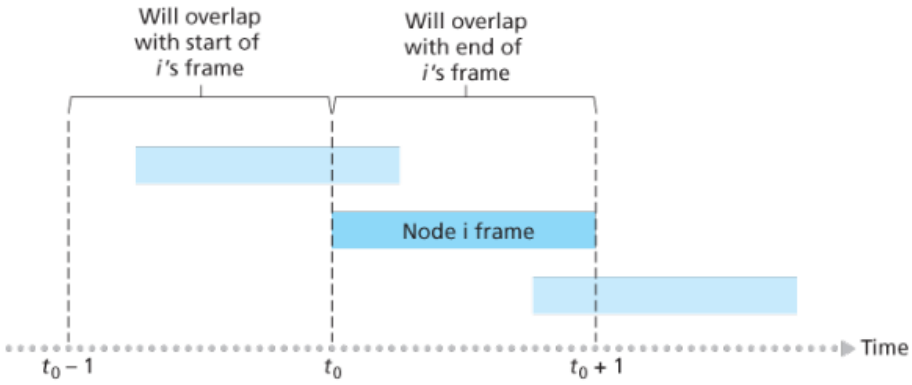
Slotted ALOHA

- Assume all frames of equal size, time divided into equal slots, nodes transmit only at beginning of slot
- If collision happens, node retransmits frame in every subsequent slot with probability p until success



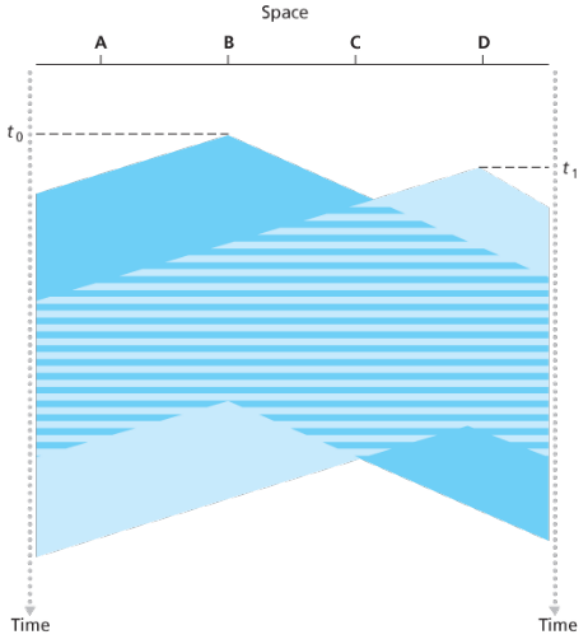
Pure (unslotted) ALOHA

- Time not divided into slots, no synchronisation
- When there is a new frame, transmit immediately
- Chance of collision increases: frame sent at t_0 collides with other frames sent in $(t_0 - 1, t_0 + 1)$ → utilisation is *worse* than slotted ALOHA (about half)



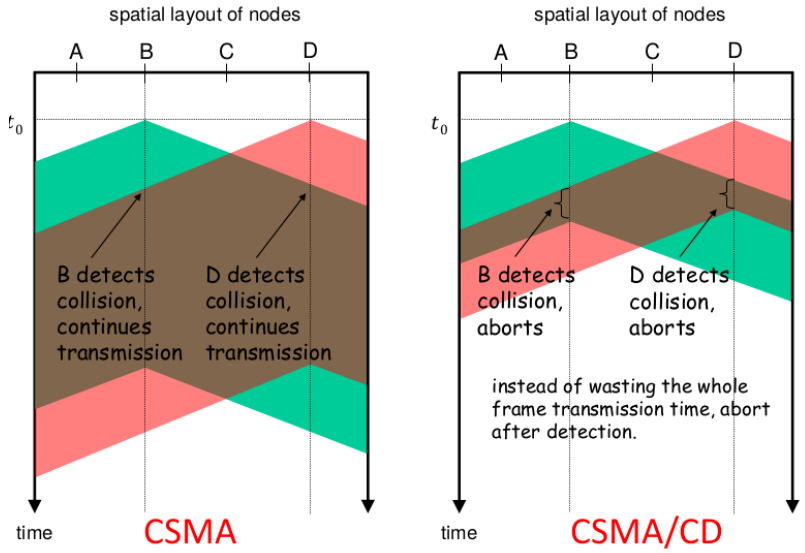
CSMA (Carrier Sense Multiple Access)

- Sense the channel before transmission
  - If channel is sensed *idle*, transmit frame
  - If channel is sensed *busy*, defer transmission
- Collisions may still happen: both nodes sense idle at same time, and start transmission after (because of *propagation delay*, nodes don't hear each others' transmission immediately)

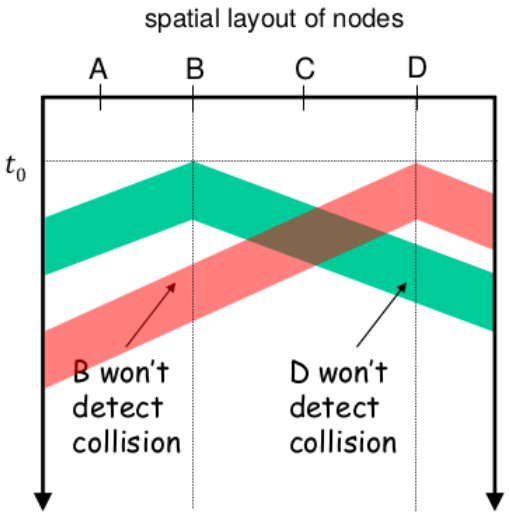


CSMA/CD (Collision Detection)

- CSMA, where you *abort transmission* when collision is detected → reduce channel wastage
- Retransmit after a random amount of time
- (Used in early Ethernet)

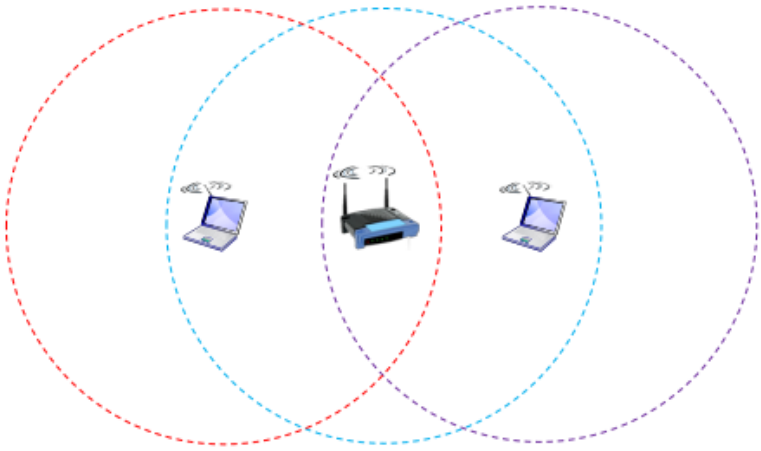


($\star$) *Minimum frame size* in CSMA: if frame size too small, collision may not be detected $\rightarrow$ no retransmission



CSMA/CA (Collision Avoidance)

- Difficult to detect collisions in some systems (hidden node problem)
- Receiver needs to return ACK if frame received is OK
- (Used in wireless LANs)



12.5 Switched Local Area Networks

12.5.1 MAC Address

MAC/Physical Address: every adapter has one, used to send and receive link layer frames

- 48 bits long (e.g. `5C-F9-DD-E8-E3-D2`)
- Usually burned in NIC ROM, sometimes software settable
- Every MAC address is supposed to be *unique*—first 3 bytes identifies vendor of an adapter
- When adapter receives frame, it checks if destination MAC address matches its own:
  - If yes, take the frame and extract the datagram
  - If no, simply discard the frame

IP Address	MAC Address
32 bits	48 bits
Network-layer address	Link-layer address
Moves *datagrams* from source to dest	Moves *frames* over a single link
Dynamically assigned, hierarchical	Permanent, identifies the hardware (adapter)

12.5.2 Address Resolution Protocol (ARP)

Question: how to translate IP addresses $\leftrightarrow$ MAC addresses?

ARP Table: each IP node (host/router) has one

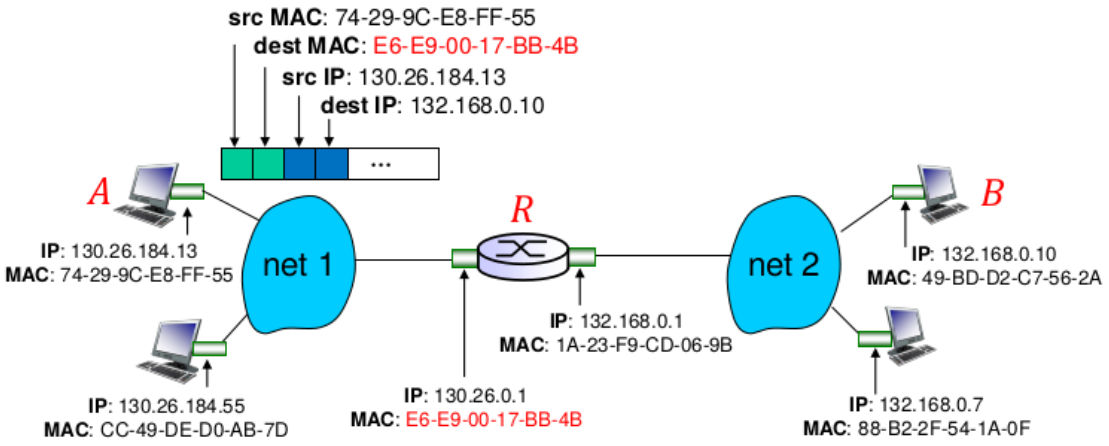
- `<IP address, MAC address, TTL>`: stores mapping from IP to MAC addresses

Sending frame in *same* subnet

- If A knows B's MAC address from ARP table: create and send frame with B's MAC address
- If A doesn't know B's MAC address: broadcast *ARP query packet* containing B's IP address, setting dest MAC to `FF-FF-FF-FF-FF-FF` $\rightarrow$ only B replies to A with its MAC address $\rightarrow$ A stores B's IP-MAC mapping in its ARP table

Sending frame in *different* subnet

- A sets dest MAC to R's MAC address, and dest IP to B's IP
- R will create a new frame with B's MAC address



12.6 Ethernet

Local Area Network (LAN): network that interconnects computers in a geographical area, e.g. university campus. Can consist of multiple subnets

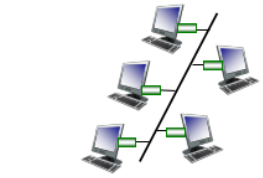
- LAN technologies: token ring, ethernet, Wi-Fi, others; ethernet is the dominant wired technology

12.6.1 Physical Layer Media

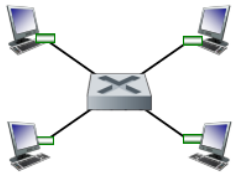
- Twisted Pair Copper Connectors: usually shorter (<100m)
- Optical Fibre Connectors: can be much longer (hundreds of km)

12.6.2 Physical Topology

- Bus topology: all nodes can collide with one another
- Star topology: *switch* in center, nodes do not collide (point-to-point connection)

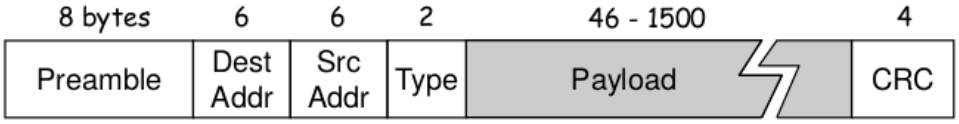


Ethernet with bus topology

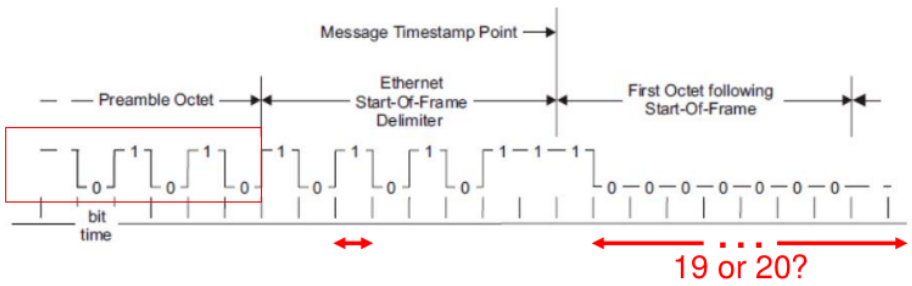


Ethernet with star topology

12.6.3 Ethernet Frame Structure



- Payload: typically IP datagram. Min 46 bytes (for collision detection), max 1500 bytes (MTU)
- Preamble: 7 bytes with pattern 10101010 followed by 1 byte with pattern 10101011, used to synchronize sender and receiver clock rates (so receiver can figure out width of a bit)
- Dest address: if NIC receives frame with its dest address or broadcast address, extract data in frame; otherwise, discard
- Type: indicates higher level protocol (typically IP)
- CRC: detect corruption



12.6.4 Ethernet Data Delivery

- Connectionless and unreliable: no handshaking, no ACKs or NAKs
- Multiple Access Protocol: CSMA/CD with binary (exponential) backoff

12.6.5 Ethernet CSMA/CD

Sender

- Create frame
- If sense channel idle, start frame transmission; if busy, wait till channel idle
- If transmit entire frame without collision, done; else abort and send *jam signal*
- After aborting, enter *binary back-off*

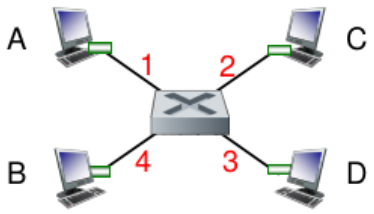
Binary/exponential back-off

- After m^{th} collision, choose K at random from $\{0, 1, 2 \ldots, 2^m - 1\}$
- Wait for $K \times 512$ bit times

12.7 Link Layer Switches

12.7.1 Ethernet Switch

- Stores and forwards Ethernet frames
- Buffers frames and is full duplex
- Examines incoming MAC address, and selectively forwards it to one or more outgoing links
- *Transparent* to hosts: no IP address, hosts unaware of presence of switches



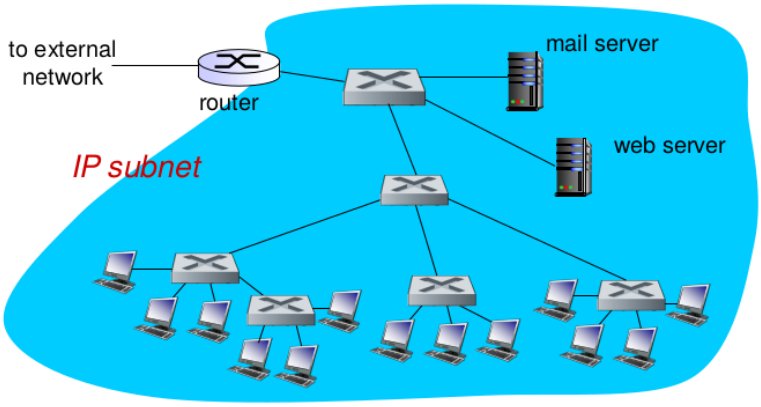
12.7.2 Switch Forwarding Table

MAC address	Interface	TTL
A	1	60
B	2	120
...		

- Format: `<MAC address of host, interface of host, TTL>`
- Self-learning: switch learns which hosts can be reached through which interfaces
  - When receiving frame from A, store location of A in switch table
  - If B is found in table, forward frame to that link
  - If B is not found in table, broadcast frame to all outgoing links

12.7.3 Interconnected Switches

Switches can be interconnected in hierarchy, and form a *spanning tree*



12.7.4 Router vs. Switch

Router	Switch
Network layer	*Link* layer
Check *IP address*	Check *MAC address*
Store-and-forwards *datagrams*	Store-and-forward *frames*
Computes routes to destination	Simply forwards to outgoing link or broadcast
More complex, usually needs to be configured	Simpler, plug-and-play, self-learning

13 Multimedia Networking

How do we deliver multimedia over the internet (OTT), when networking applications don't by themselves make latency/real-time guarantees?

3 types of multimedia networking applications

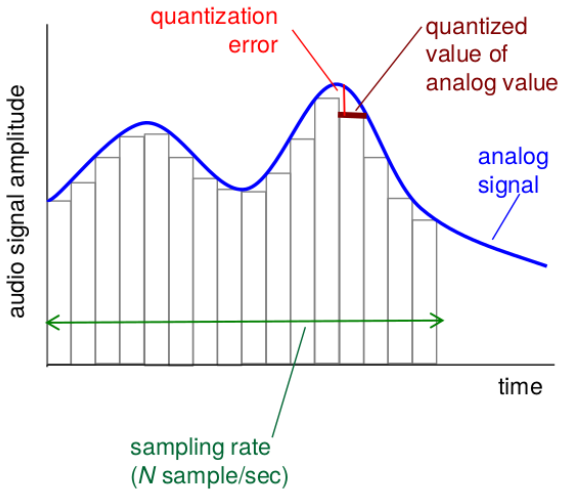
- Streaming stored content (video/audio): e.g. YouTube, Netflix
- Conversational/two-way live voice/video over IP, e.g. Skype, Zoom
- Streaming/one-way-live audio/video

13.1 Multimedia: Audio

Audio: sequence of signal amplitudes across time

Sampling

- Sample an analog audio signal at a certain rate (e.g. 8,000 samples/s for telephone, 44,100 samples/s for CDs)
- Then *quantize* each sample to an integer value (e.g. 2^8, or 2^{16} values for CDs)



13.2 Multimedia: Video

Video: sequence of images displayed at a constant rate (e.g. 30/s)

Encoding: use *redundancy*—within and between images to decrease #bits required to encode image

- Spatial: within an image
- Temporal: from one image to the next—can send only the difference from one frame to another

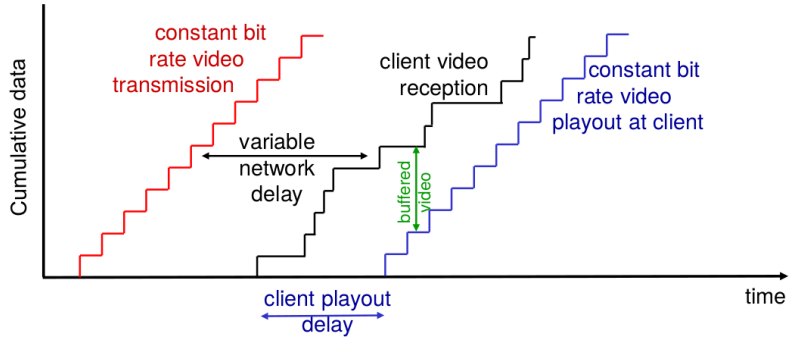
Compressions

- Constant bit rate: encoding rate fixed
- Variable bit rate: encoding rate changes as amount of spatial, temporal encoding changes (allows for better quality when there's more to encode)

13.3 Streaming Stored Content

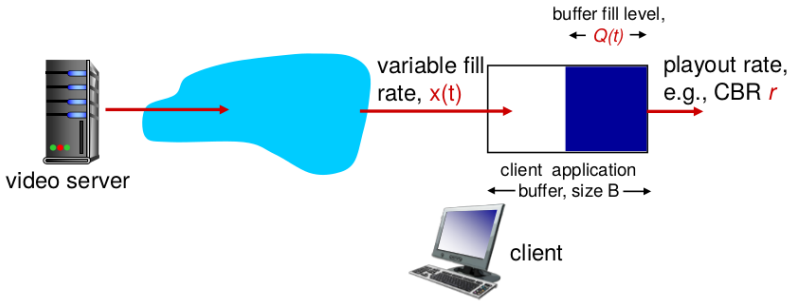
Continuous playout constraint: once client playout begins, playback must match original timing

- Client-side buffer: helps to deal with variable network delays
- Client playout delay: only start playing back some time after we receive packets, buffer is semi-full



Playout buffering: average fill rate $\bar{x}$, playout rate r

- If $\bar{x} < r$, buffer eventually empties, will eventually freeze
- If $\bar{x} > r$, buffer will not empty if initial playout delay is large enough
  - Initial playout delay: if too large, buffer starvation less likely, but larger delay until content starts playing
  - Avoid full buffer: just don't take from TCP buffer (so rely on TCP congestion mechanism), OR application layer sends message down



Using UDP (push-based streaming)

- Server just sends at appropriate rate, without congestion control
- Error control at application level

Using HTTP (pull-based streaming)

- Client retrieves data by HTTP GET, sent at maximum possible rate under TCP
- Fill rate fluctuates due to TCP congestion control, and in-order delivery

13.4 Voice-over-IP (VoIP)

Latency (end-end-delay) requirement: delays need to be small, <400ms!

Speaker's audio: alternating talk spurts and silent periods

- Only generate packets and transmit during talk spurts
- 64kbps encoding, partitioned into 20ms chunks and sent at 8Kbytes/s (same), 160bytes of data/chunk

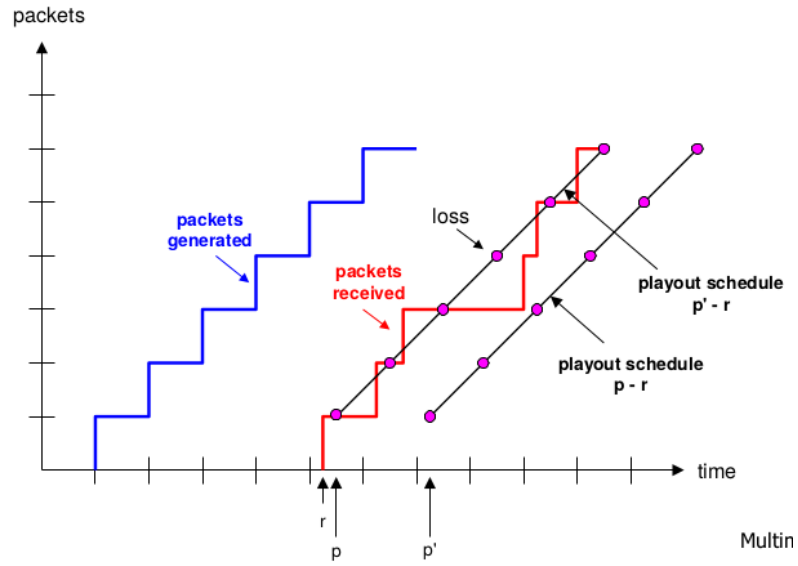
Loss

- Network loss: packet lost in network (e.g. buffer overflow)
- Delay loss: packet arrives too late for playout
- Loss tolerance: between 1% and 10%

13.4.1 Playout Delay

Fixed playout delay q

- Large q: less packet loss
- Small q: better interactive experience



Adaptive playout delay

- Estimate packet delay d_i using exponentially weighted moving average
- Estimate average deviation v_i of delay—use about $K = 3$ or $K = 4$ standard deviations
- Playout time $= t_i + d_i + Kv_i$

$$d_i = (1-\alpha)d_{i-1} + \alpha\,(r_i - t_i)$$

delay estimate after ith packet

small constant, e.g. 0.1

time received - time sent (timestamp)

measured delay of ith packet

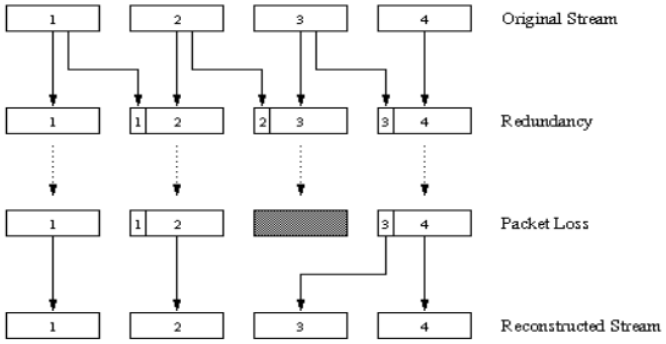
13.4.2 Recovery from Loss

Redundant chunk every N chunks

- Redundant chunk is simply XOR of N original chunks
- Reconstruct at most 1 lost chunk every N+1 chunks (with playout delay)

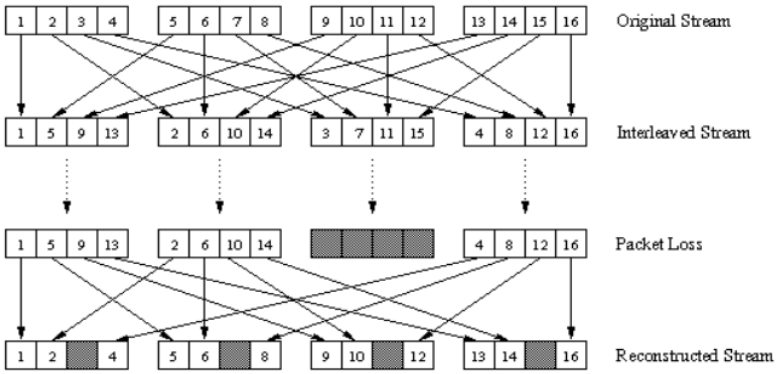
"Piggyback lower quality stream"

- Send high resolution stream + low resolution stream for previous one
- It's a form of Forward Error Correction (FEC): send enough bits to allow for recovery



Conceal loss by interleaving

- Packet contains units from different chunks → if loss, still have most of each chunk
- No redundancy, but increases playout delay

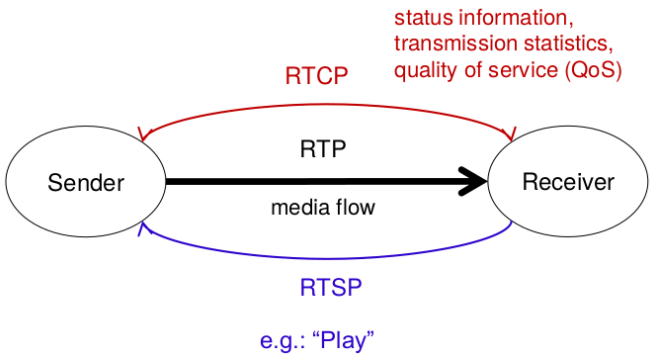


13.5 Protocols: for Real-Time Conversational Applications

13.5.1 Real-Time Protocol (RTP)

Suite of protocols: RTP, RTCP (control), RTSP (streaming: done over TCP)

- Each uses a different port number
- Runs in end systems over UDP



RTP header

- Payload type (7 bits): type of encoding
- Sequence number (16 bits): increment by 1 for each packet sent
- Timestamp (32 bits)
- Synchronisation Source ID (SSRC) (32 bits): identifies source of RTP stream

13.6 Dynamic Adaptive Streaming over HTTP (DASH)

Stream media over HTTP protocol, by dividing it into streamlets of different quality

- MPD (Media Presentation Description) file: gives client information on available videos and qualities
- Client runs adaptive bitrate algorithm (ABR) to determine which segment of what quality to download next

Advantages and disadvantages

- (+) Simple server, just a regular web server
- (+) No firewall problems, port 80 HTTP
- (+) Standard caching works
- (–) Longer media segments
- (–) High latency: several seconds

13.7 Summary of Multimedia Applications

VoD (e.g. YouTube, Netflix)

- One-way, stored, long latency okay → DASH

Live-streaming (e.g. Twitch)

- One-way, live source, not too long latency → DASH

VoIP (e.g. Skype, Zoom)

- Two-way, must have low latency → RTP

14 Protocol Summary

Layer	Protocol	Underlying Protocol	Information
Application	HTTP	TCP 80	
	DNS	UDP 53	Find IP address for domain name
	DHCP	UDP 67/68	For host to obtain IP address/other info
	RIP	UDP 520	Distance vector routing algorithm
	RTP	UDP	
Transport	TCP	IP	
	UDP	IP	
Network	IP	(Link layer)	
	ICMP	IP	Communicate network-level information
Link	Ethernet	(uses CSMA/CD)	
	Wi-Fi	(uses CSMA/CA)	
	ARP		Discovers and translates IP and MAC address

Access control protocol type	Protocol
Channel partitioning	TDMA
	FDMA
Taking turns	Polling
	Token passing
Random access	(Slotted) ALOHA
	CSMA/CD
	CSMA/CA