CS2105

Introduction to Computer Networks

AY2022/23 Semester 1

Notes by Jonathan Tay

Last updated on October 27, 2022

Contents

Part I

Introduction

1 Network Edge

Hosts (end systems) access the Internet through **access networks**, running network applications, and communicating over **links**.

Wireless access network use access points to connect hosts to routers, either via wireless LANs, e.g. Wi-Fi, or wide-area wireless access, e.g. 4G.

Hosts can connect directly to an access network physically via guided media, e.g. twisted pair cables and fiber optic cables, or over-the-air via unguided meia, e.g. radio.

2 Network Core

A mesh of interconnected routers which forward data in a network.

Transmitting data through a network takes place via **circuit switching** or **packet switching**.

2.1 Circuit Switching

Circuits along the path are reserved before transmission can begin, which mean that no other circuit can use the same path, but performance can be guaranteed.

However, there is a finite number of circuits, so the network is limited in its capacity. This approach is used in telephone networks.

2.2 Packet Switching

Messages are broken into smaller chunks, called **packets**. Packets are transmitted onto a link at a **transmission rate**, also known as **link capacity** or **bandwidth**.

The **packet transmission delay** (d_{trans}) is the time needed to transmit an L-bit packet into the link at a transmission rate R.

$$d_{\text{trans}} = \frac{L \text{ in bits}}{R \text{ in bits/sec}} \text{ seconds}$$

Packets are passed from one **router** to the next across links on the path from the source to the destination.

This incurs a **propagation delay** (d_{prop}), which depends on the length d of the physical link, and the propagation speed s in the medium.

$$d_{\text{prop}} = \frac{d}{s \approx 2 \times 10^8 \ m/s}$$

At each router, packets are **stored and forwarded**, which means an entire packet must arrive before being transmitted onto the next link.

Therefore, with P packets and N links, the **end-to-end delay**:

$$d_{\text{end-to-end}} = (P + N - 1) \cdot \frac{L}{R}$$

At the router, packets are checked for bit errors and the output link is determined using **routing algorithms**. This incurs a **nodal processing delay** (d_{proc}).

Therefore, packets have to **queue** in a **buffer** at each router, also incurring a **queueing delay** (d_{queue}), which is the time spent waiting in the queue before transmission.

In general,

$$d_{\text{end-to-end}} = d_{\text{trans}} + d_{\text{prop}} + d_{\text{queue}} + d_{\text{proc}}$$

2.3 Packet Loss

Router buffers have a finite capacity and packets arriving to a full queue will be **dropped**, resulting in **packet loss**. This is known as **buffer overflow**.

Packets can be corrupted in transit or due to noise.

2.4 Throughput

The number of bits that can be transmitted the per unit time.

Each link has its own **bandwidth** R, so throughput is measured for end-to-end communication.

$$\text{throughput} = \frac{1}{\sum_{i=1}^{n} \frac{1}{R_i}} \text{ where } n \text{ is the number of links}$$

Peak throughput and other throughput calculations are not covered in this module.

3 Network Protocols

The format and order of messages exchanged, and the actions taken after messages are sent and received.

The protocols in the Internet are arranged in a stack of **5 layers**:

1. **application**, e.g. HTTP, SMTP
2. **transport**, e.g. TCP, UDP
3. **network**, e.g. IP
4. **link**, e.g. ethernet, 802.11
5. **physical**, e.g. bits on the wire

Part II

Application Layer

Application layer protocols define the:

1. **types of messages exchanged**, e.g. requests, responses
2. **message syntax**, e.g. message fields and delineation
3. **message semantics**, i.e. meaning of information in fields
4. **rules** for when and how applications send and respond to messages

4 Architectures

In the **client-server** architecture, a **client initiates contact** with a **server**, which waits for the request before providing a service back to the client.

This relies on data centers for scaling, and clients are usually implemented in web browsers.

In the **peer-to-peer** architecture, arbitrary end systems communicate directly with each other, requesting and returning services.

This is **self-scalable** as new peers bring service capacity and demand. However, this architecture is more complex as peers are connected intermittently.

Regardless of which architecture is used, the application layer ride on the **transport layer** protocols — **TCP** or **UDP** — for data integrity, throughput, timing, and security.

5 Hypertext Transfer Protocol (HTTP)

The application layer protocol of the Internet.

HTTP uses the client-server architecture and TCP as the transport service. The client must **initiate a TCP connection** with the server before sending a **request message**.

The server receives the request message and sends the **response message** with the requested object back to the client.

The **round-trip time** (RTT) is the time taken for a packet to travel from a client to the server and back.

The **HTTP response time** in general takes one RTT to establish the TCP connection, one more RTT for the HTTP request to be fulfilled, plus the file transmission delay.

5.1 Request Message

```
1 GET /index.html HTTP/1.1\r\n
2 Host: www.example.org\r\n
3 Connection: keep-alive\r\n
4 ...
5 \r\n
6 <body>
```

Line 1 is the **request line**, specifying the **method**, **URL**, and **HTTP version**.

Lines 2 to 4 are the **header lines**, each specifying the **header field name** and **value**. Only the `Host` header is required.

The extra blank line (line 5) indicates the end of the header lines, after which the body follows.

5.2 Response Message

```
1 HTTP/1.1 200 OK\r\n
2 Date: Wed, 23 Jan 2019 13:11:15 GMT\r\n
3 Content-Length: 606\r\n // in bytes
4 Content-Type: text/html\r\n
5 ...
6 \r\n
7 <data>
```

Line 1 is the **status line** specifying the **HTTP version** and the **response status code**.

Lines 2 to 5 are the **header lines**, and lines 7 and onward contain the data requested, e.g. the HTML file.

5.3 HTTP/1.0 (non-persistent HTTP)

At most one object is sent over a TCP connection, after which the connection is closed.

Downloading multiple objects therefore requires multiple connections, incurring 2 RTTs per object in addition to the overhead for each TCP connection, which some browsers may parallelize.

5.4 HTTP/1.1 (persistent HTTP)

Unlike HTTP/1.0, the server leaves the TCP connection open after sending the response, which is reused for subsequent messages.

Persistent connections with pipelining allow multiple objects to be requested even before the server has responded to previous requests.

This reduces the total response time to as low as one RTT.

5.5 Conditional `GET`

Avoiding unnecessary requests for cached and up-to-date objects.

Clients send an additional `If-Modified-Since` header with the request, containing the date of last modification.

If the requested object has been modified after the date specified, then the server responds with a `200 OK` along with the requested object data.

Otherwise, the server responds with a `304 Not Modified`, which means the client can use its cached version.

5.6 Cookies

Maintaining state despite the stateless nature of HTTP.

Cookies are sent using the `Cookie` and `Set-Cookie` header fields in requests and responses respectively.

They are created by servers, stored client-side, and managed by browsers.

Cookies are sent to the server in subsequent requests, which the server can then use to execute **cookie-specific actions**, e.g. retrieve a shopping cart.

6 Domain Name System (DNS)

Computers use **IP addresses** to identify hosts and communicate, but **hostnames** (e.g. www.example.org) are easier for humans to remember.

The **domain name system** translates between a hostname and its IP addresses — multiple IPs are usually used for load-balancing.

DNS runs on the **UDP** transport protocol (chosen for its speed), which means queries can get lost or corrupted in transmission, but due to the locality of queries, such incidents are rare.

Furthermore, in the event of a query loss, browsers can simply re-issue the query, or even issue multiple queries right from the start.

Use `nslookup` or `dig` at the command line to perform a DNS query.

6.1 DNS Servers

DNS servers store resource records in distributed databases implemented in a heirarchy of many name servers.

1. **Root servers**: answer requests for records in the root zone, returning a list of authoritative name servers for the appropriate **top-level domain** (TLD).
2. **Top-level domain servers**: answer requests for `.com`, `.org`, etc., and the top-level country domains, e.g. `.sg`.
3. **Authoritative servers** belong to organizations and service providers, mapping an authoritative hostname to IP addresses.
4. **Local servers**: cache answers to DNS queries for faster access, and act as proxies to forward DNS queries if the answer is not cached.

6.2 Resource Records (RR)

Resource records format: `(name, value, type, ttl)`.

type	name	value
`A`	hostname	IP address
`CNAME`	alias name	canonical (real) name
`NS` (name server)	domain	hostname of authoritative name server
`MX` (email server)	mail server	name of mail server

`ttl`, a.k.a. **time-to-live**, is the number of seconds that a record is valid for after it is cached in a local server.

When the TTL reaches zero, it is invalidated and removed from the cache.

This also means that changes to an IP address of a host may not be immediately reflected until the TTL expires.

6.3 Domain Name Resolution

In a **recursive query**, the query is forwarded from the client through the local server, root server, TLD server, and then to the authoritative server, and the response is forwarded back to the client.

In an **iterative query**, the local DNS server handles the queries and responses directly, pinging the root, TLD, and lastly the authoritative servers, without any forwarding.

Both query methods are valid, but iterative querying is used in practice.

7 Socket Programming

A **process** is a program running on a host.

Within the same host, the OS can define inter-process communication, but processes on different hosts communicate by exchanging messages.

Processes are identified by an **IP address *and* port number**.

type	number of bits
IPv4	32
IPv6	128
port	16

A **socket** is the software interface — a set of APIs — between processes and transport layer protocols.

Processes send and receive messages via a socket.

7.1 via UDP

The sender attatches the **destination IP address *and* port number** to *each packet*, which the receiver extracts.

1. Client creates `clientSocket`.
2. Server creates `serverSocket`.
3. Client creates packet with `serverIP` and port x, sent via `clientSocket`.
4. Server reads the **datagram** from `serverSocket`.
5. Sever sends the reply specifying the client address and port number via `serverSocket`.
6. Client reads the datagram from `clientSocket`.
7. `clientSocket` is closed.

No connection is established between the client and the server. This method of transmission via datagrams over UDP is *unreliable*.

7.2 via TCP

When contacted by a client, the server TCP connection creates a *new socket* for the server process to communicate with that client.

This allows the server to communicate with *multiple clients simultaneously*.

1. Server creates `serverSocket` on port x.
2. Client creates `clientSocket`, connecting to `serverIP` on port x.
3. Client and server set up a TCP connection.
4. Client and server exchange requests using `clientSocket` and `connectionSocket` respectively.
5. Server closes `connectionSocket`.
6. Client closes `clientSocket`.

Part III

Transport Layer

Transport layer services deliver messages between processes on different hosts, while packet switches check the destination IP address for routing.

The **sender** breaks messages into **segments**, and passes them to the *network layer*.

The **receiver** reassembles the segments into the message, and passes it the *application layer*.

8 User Datagram Protocol (UDP)

UDP transmission is **unreliable** but fast, often used only by loss-tolerant and rate-sensitive applications, e.g. multimedia.

UDP adds the following on top of IP:

– **multiplexing** at sender,
– **de-multiplexing** at receiver, and
– **checksum** at sender and receiver.

At the receiver, every datagram with the same *destination port number* will be directed to the *same UDP socket*.

UDP adds the following headers to each segment:

1. **source port number**,
2. **destination port number**,
3. **segment length**, and
4. **checksum**.

UDP exists because it is:

– **fast**: no connection establishment $\implies$ no delay,
– **simple**: no connection state,
– **small**: header size is small,
– **unlimited**: no congestion control.

8.1 UDP Checksum

Checksums are used to detect transmission errors in each segment, e.g. flipped bits.

Steps to compute the checksum:

1. Partition the segment into 16-bit integers.
2. Perform **binary addition** with every integer.
3. Add any *carry bits* to the next integer.
4. Take the 1s complement of the final integer to get the checksum.

9 Reliable Data Transfer (RDT)

The underlying network layer may *corrupt*, *drop*, *re-order*, or *delay* packets during transmission — it is **inherently unreliable**.

RDT protocols aim to *guarantee packet delivery* (in the order sent) and *correctness*.

RDT protocols are purely hypothetical — they only form the basis for real-life implementations.

9.1 RDT 1.0

We assume that the underlying channel is *perfectly reliable*.

The sender sends data through the channel, and the receiver reads the data from the channel.

9.2 RDT 2.0

We introduce **bit errors** into the otherwise perfect underlying channel.

Now the receiver has to validate the **checksum** to *detect* bit errors, and reply with either of the following:

– **acknowledgement** (ACK): if the packet was received correctly, or,
– **negative acknowledgement** (NAK): if the packet was corrupted, requesting re-transmission.

The sender will re-transmit the packet after receiving an NAK, *or* if either the ACK or NAK is corrupted.

However, this is a **stop-and-wait** protocol, as the sender has to wait for each acknowledgement before sending the next packet.

9.3 RDT 2.1

Because ACKs can get corrupted, causing duplicate packets to be transmitted to the receiver in RDT 2.0, we add a **sequence number** to each packet.

If the receiver identifies a duplicate packet, it simply *discards* it — not sending it to the application.

9.4 RDT 2.2

We want a **NAK-free** protocol — such that the receiver *only* sends ACKs.

The NAKs are replaced by ACKs, and *all* ACKs are appended with the sequence number of the last packet that was received correctly.

If the sender receives *duplicate* or *corrupted* ACKs, it re-transmits the current packet.

9.5 RDT 3.0

We introduce **packet loss** and **delays** to the underlying channel.

These are handled via **timeouts** which trigger re-transmission if an ACK is either corrupted, lost, or delayed.

Because re-transmission may duplicate packets, we still use the sequence number to discard duplicate packets.

However, this has **low throughput** and **low utilization**, because a sender spends most of its time waiting for an ACK.

9.6 Pipelined Protocols

We can increase throughput by **pipelining** — sending multiple yet-to-be-acknowledged packets at a time.

The **window size** controls the number of packets sent at a time.

This increases the *throughput* and *utilization*, but at a cost:

- increased range of sequence numbers
- buffering at sender and receiver
- increased design complexity

9.6.1 Go-back-*n* (GBN)

A GBN sender tracks 2 things:

1. a **sliding window** of n contiguous packets, and,
2. a **timer** for the oldest unACK'd packet, which is re-transmitted in the window when it expires.

A GBN receiver demands that all packets are received in order:

- **ACK** every successive packet that is received in order
- **discard** every out-of-order packet (i.e. if packet l is lost, every packet $> l$ is discarded until packet l is received correctly)

GBN is also known as **cumulative ACK** — an ACK means that every packet up to that point has been received correctly.

GBN behaves exactly like RDT 3.0 when $n = 1$.

The timeout has to be chosen correctly — too short and it results in **premature timeouts**, too large and the protocol becomes slow.

9.6.2 Selective Repeat (SR)

SR buffers out-of-order packets instead of dropping them.

An SR receiver ACKs individual packets.

An SR sender maintains a timer for each unACK'd packet, re-sending *only that packet* when its timer expires.

However, the sliding window of the sender cannot advance past any unACK'd packet until it is ACK'd.

10 Transmission Control Protocol (TCP)

TCP has several characteristics:

- **point-to-point**:
  one sender, one receiver
- **connection-oriented**:
  handshakes before transmission
- **full-duplex service**:
  bi-directional data flow
- **reliable, in-order byte stream**:
  bytes are labelled by sequence numbers

Every TCP connection (socket) is identified by a tuple:

1. the **source IP address**,
2. the **source port number**,
3. the **destination IP address**, and,
4. the **destination port number**.

The **maximum segment size** is **1460 bytes**, but this is *not inclusive* of 20 bytes for the **TCP packet headers**:

1. **source port number**: 16 bits
2. **destination port number**: 16 bits
3. **sequence number (`seq`)**:
   32-bit byte number of the first byte in the segment
4. **acknowledgement number (`ack`)**:
   32-bit byte number of the next expected byte
5. **`ACK` bit (`A`)**:
   `1` if the ACK is in use, `0` otherwise
6. **`SYN` bit (`S`)**:
   `1` if this packet is establishing a new connection, `0` otherwise
7. **`FIN` bit (`F`)**:
   `1` if this packet is closing the connection, `0` otherwise
8. **checksum**: 32 bits

10.1 Connection

TCP employs a **3-way handshake** to *establish a connection* before exchanging application data:

1. **client → server**:
   `S = 1, seq = x`
2. **server → client**:
   `S = 1, seq = y, A = 1, ack = x + 1`
3. **client → server**:
   `A = 1, seq = x + 1, ack = y + 1, data = ...`

The client and server choose their **initial sequence numbers** x and y *randomly* and *independently*.

To *close a connection*:

1. **client → server**:
   `F = 1, seq = x`
2. **server → client**:
   `A = 1, ack = u + 1`

3. **server → client**:
   `F = 1, seq = t`
4. **client → server**:
   `A = 1, ack = t + 1`

10.2 Acknowledgements

TCP is also a **cumulative ACK** protocol, but handling of out-of-order segments is implementation-dependent.

Both the sender and receiver have a **buffer** for sent or received **segments**.

A TCP sender tracks 2 things:

1. the **next sequence number**, which is the first byte of the next segment to send, and,
2. a *single* **timer** for the oldest unACK'd packet, which is re-transmitted in the window when it expires.

There are 4 potential actions for a TCP receiver to take when it receives a segment:

1. **in-order, first non-ACK'd segment**:
   delay ACK for up to 500ms for the next segment, otherwise ACK with `ack = seq + len(data)`, which is equivalent to the next expected byte
2. **in-order, previous segment pending ACK**:
   send *single* cumulative ACK, ACKing both segments with `ack = seq` of the next expected byte
3. **out-of-order (gapped up) segment**:
   send *duplicate* ACKs with `ack = seq` of the expected byte
4. **gap-fill segment**:
   send ACK with `ack = seq` of the next expected byte if the packet fills the lowest end of the gap, otherwise it is still out-of-order, so repeat above.

ACK packets can also contain data (**piggy-back**), which is useful for bi-directional communication as it halves the number of packets needed.

10.3 Timeout

The timeout interval is constantly computed based on the **estimated RTT**:

$\mathrm{RTT}_{\mathtt{est}} = (1-\alpha)\,\mathrm{RTT}_{\mathtt{est}} + \alpha\,\mathrm{RTT}_{\mathtt{sample}}$
$\mathrm{RTT}_{\mathtt{dev}} = (1-\beta)\,\mathrm{RTT}_{\mathtt{dev}} + \beta\,\left\lvert \mathrm{RTT}_{\mathtt{sample}} - \mathrm{RTT}_{\mathtt{est}} \right\rvert$
$\mathtt{timeout} = \mathrm{RTT}_{\mathtt{est}} + 4 \times \mathrm{RTT}_{\mathtt{dev}}$

Typically, $\alpha = 0.125$ and $\beta = 0.25$.

The timeout interval has a **safety margin** factor of 4 $\mathrm{RTT}_{\mathtt{dev}}$.

TCP also employs **fast retransmission** — the sender resends segments immediately after receiving 4 ACKs for the same segment, assuming that they were lost.

Part IV

Network Layer

11 Routing

Routing is done heirarchically through **autonomous systems** of routers and links owned by ISPs.

Routing within an autonomous system is known as **intra-AS** routing.

There are 2 classes of routing algorithms:

1. **link-state algorithms**:
   every router has complete knowledge of the network topology, as they all broadcast link costs periodically
2. **distance-vector algorithms**:
   routers only know their direct neighbors and their link costs, and they have to exchange **local views** with neighbors and update their own

abstracting network topology

A network can be viewed as a graph:

- **vertices**: routers
- **edges**: physical links between routers
- **costs**: either constant, or inversely related to bandwidth

Routing involves finding the **least-cost path** between two vertices:

- **`c(x, y)`**:
  link cost between routers x and y, or ∞ if x and y are not **direct neighbors**
- **$d_x(y) = min_v\{c(x, v) + d_v(y)\}$**:
  least-cost path from `x` to `y`, where `min` is taken over all direct neighbors of `x`

updating local views

1. Every router sends its distance vectors to its direct neighbors.
2. If a router receives a distance vector from a neighbor, and finds a shorter path to another router, it updates its own local view.
3. After several iterations, all routers will have the same local view.

11.1 Routing Information Protocol (RIP)

RIP implements the distance-vector algorithm, using **hop count** as the cost metric, and runs over UDP every 30 seconds, with **self-repair** assuming that a neighbor without an update after 3 seconds has failed.

12 Internet Protocol (IPv4)

12.1 IP Addresses

IP addresses are **32-bit integers** used to identify a host or router.

Each host can have multiple IP addresses, each of which is associated with a single a **network interface** that sends and receives packets.

This means that a device connected over Wi-Fi and Ethernet will have two IP addresses.

converting binary to decimal notation

An IP address can be written in **dotted decimal notation** where its 4 bytes are converted into decimal values between `0` and `255`.

For example, `00000001 00000010 00000011 10000001` is written as `1.2.3.129`.

IP addresses are assigned in 2 ways:

1. manual configuration by a sysadmin, or,
2. automatic assignment by a **DHCP** server.

12.1.1 Dynamic Host Configuration Protocol (DHCP)

A **DHCP server** dynamically allocates IP addresses to hosts on a network.

This allows IP addresses to be **renewable**, **reusable**, and **scalable**.

DHCP runs on **UDP** ports 67 for the client and 68 for the server; chosen for its speed.

DHCP process

1. **DHCP discover**: broadcasted by host
2. **DHCP offer**: response from DHCP server
3. **DHCP request**: from host to server
4. **DHCP ACK**: reply from server with the IP address

In the DHCP discovery message, the **arriving client** sends a broadcast using a **special IP** `0.0.0.0` on port 68 in its `src` field because it has yet to be assigned a valid IP address.

It also uses the **broadcast address** `255.255.255.255` in its `dest` field, which means all hosts on the same **subnet** will receive the message.

However, only the **DHCP servers** running on port 67 will respond with their **DHCP offers**.

DHCP message fields

1. **`src`**: source IP address
2. **`dest`**: destination IP address
3. **`yiaddr`**: proposed IP address for the client
4. **`transaction`**: sequence number for the request
5. **`lifetime`**: duration the offer is valid for

`yiaddr` is proposed by the server during the DHCP offer, after which the arriving client sends a DHCP request with it in its `yiaaddr` field as well.

Because there may be multiple DHCP servers, the arriving client will receive multiple DHCP offers.

The client has to broadcast its decision to `255.255.255.255` in the `dest` field of the DHCP request so that all the other servers can rescind their offers.

DHCP can also provide additional network information:

1. the IP address of the **default gateway** (first-hop router),
2. the IP address of the local DNS server, and,
3. the **network mask**

12.1.2 Special IP Addresses

Special IP addresses are *ranges* in the form `a.b.c.d/x`.

The number represented by x indicates the number of bits in the **network prefix** which are fixed to that range — the remaining 32 — x bits are the **host ID**.

special addresses

`0.0.0.0/8`	non-routable meta-address
`127.0.0.0/8`	loopback address, aka. **`localhost`**
`10.0.0.0/8` **`172.16.0.0/12`** **`192.168.0.0/16`**	**private addresses**
`255.255.255.255/32`	**broadcast address**

Private IP addresses are *not globally unique*, and are used for *internal networks*.

They do not require coordination with IANA or any Internet registry, but **network address translation** is needed to use them on the public Internet.

Network prefixes identify a **subnet**, where a group of hosts are directly connected by cables.

Hosts within the same subnet can communicate without the need for a router.

12.1.3 Classless Inter-domain Routing (CIDR)

IP address assignment is not random; CIDR is the assignment strategy of IP addresses using **heirarchical addressing**:

Routers map *blocks* of IP addresses to a single next-hop router in a **routing/forwarding table**, rather than mapping each individual IP address.

IP addresses are bought from registries or rented from an ISP's address space, who in turn get them from ICANN.

The **subnet (network) prefix** of IP addresses are of arbitrary length, and are assigned by the ISP.

subnet mask

Subnet masks are 32-bit integers used to determine the subnet of an IP address.

The subnet prefix bits are set to `1` and the host ID bits are set to `0`.

For example, for the IP address `200.23.16.42/23`, the subnet mask is `11111111 11111111 11111110 00000000`.

Its decimal equivalent is `255.255.254.0`.

12.1.4 Network Address Translation (NAT)

Private IP addresses, unlike public IP addresses, are not globally unique and cannot be used as destination IP addresses for routing.

A **NAT router** bridges the gap between local networks and the Internet by replacing private IP addresses and port numbers with its own.

It does this for all outgoing datagrams, and remembers and undoes the replacement for incoming packets using a **NAT translation table**.

12.2 IPv4 Headers

Maximum transfer unit (MTU) is the maximum size of a datagram that can be sent over a link.

IP datagrams which are too large may be **fragmented** by routers and **reassembled** at the receiver.

IP fragmentation

All fragments of an IP segment have the same ID.

The flag is set to `0` if it is the last fragment in the segment, and `1` otherwise.

The offset is the number of 8-byte blocks of data (excluding the headers) from the start.

Therefore, the IPv4 header is 20 bytes long:

field	bits	description
version	1	always 4 for IPv4
total length	16	length of the entire datagram (headers + data)
identifier	16	for fragmentation/reassembly
flags	3	
fragment offset	13	
time-to-live	8	number of remaining hops, decremented by 1 at each router, and discarded if 0
protocol	8	identifies the protocol used by the transport layer, e.g. TCP/UDP
checksum	16	computed for the header only
source	32	sender IP address
destination	32	receiver IP address

13 Internet Control Message Protocol (ICMP)

ICMP is used by hosts and routers to communicate network-level information, such as error reports and echos.

ICMP is carried in IP datagrams and has its own set of headers which are appended after the IP headers.

Part V

Link Layer

The **link layer** sends datagrams between *adjacent* nodes (hosts or routers) over a single link.

IP datagrams are encapsulated in **frames**.

The link layer is implemented in hardware adapters (network interface cards) or on a chip.

14 Error Detection and Correction

Links may be **error-prone**.

Error detection and correction bits (`EDC`) are appended to the **data bits** (`D`) before transmission across a link.

However, **error detection schemes** are not 100% reliable — they may not detect all errors, but a larger `EDC` increases the probability of detecting errors.

There are 3 popular error detection schemes:

1. **parity checking**:
   works well mathematically, but not in practice as errors are clustered together
2. **checksums**: used in TCP/UDP/IP
3. **cyclic redundancy checking**: used in the link layer

14.1 Parity Checking

1D parity checking

Include an additional **parity bit**.

For **even parity**, the parity bit is set to `0` if the number of 1s in the data bits is even, and `1` otherwise.

For **odd parity**, the parity bit is set to `1` if the number of 1s in the data bits is even, and `0` otherwise.

2D parity checking

If the data bits are arranged in a **2D matrix** with i rows and j columns, then we can compute each row and column parity.

In addition, we can compute the parity bit for the column and row parity bits, for a total of $i + j + 1$ parity bits.

1D parity checking can *detect* **single bit errors** (or any odd number of them), but not correct them.

2D parity checking can *detect and correct* **single bit errors**, by intersecting the error row and column.

In addition, it can *detect* **two-bit errors**, but not correct them.

14.2 Cyclic Redundancy Checking (CRC)

non-binary cyclic redundancy checking

First, some terminology:

- **D**: a non-binary number to transfer
- **R**: an r-digit error detection code
- **G**: an r-digit **generator** known to both sender and receiver

To transmit the new message `M`:

1. Create a new number `D'` by appending `9`, r times, to `D`.
2. Find the remainder y of `D'` divided by `G`.
3. Transmit `M = D' - y`.

Notice how `M` is divisible by `G` — the receiver can calculate the new remainder y', and discard the message if $y' \neq 0$.

---

binary cyclic redundancy checking

First, some terminology:

- **D**: binary data bits
- **R**: an r-bit error detection code
- **G**: an $r + 1$-bit **generator** known to both sender and receiver

All calculations are done **modulo 2**, to avoid carries for addition and borrows for subtraction — now identical to `XOR`.

To transmit the new message `M`:

1. Create a new number `D'` by appending `0`, r times, to `D`.
2. Find the remainder of `D'` divided by `G`, which is used as `R`.
3. Transmit `M = (D, R)`, which is `R` appended to `D`.

Now, the receiver divides `M` by `G` and checks if the remainder is `0`.

A non-zero remainder indicates an error.

CRC's error detection capabilities:

- **single bit errors**: all odd numbers of errors
- **burst errors < r+1-bits**: all such errors
- **burst errors > r-bits**: probability $1 - 0.5^r$

CRC is also easy to implement on hardware due to the modulo-2 arithmetic.

15 Link Access Control

Point-to-point links connect a sender and receiver directly — there is no need for multiple access control.

Broadcast links connect multiple nodes to a single shared broadcast channel.

Every node receives a copy of every broadcasted link, causing **collision** if two signals are received simultaneously.

ideal multiple access control

Given a broadcast channel of rate R bps, the muliple access control protocol should be:

1. **collision-free**
2. **efficient**: a single transmitting node should send at rate R
3. **fair**: each transmitting node among M nodes should send at an average rate R/M
4. **decentralized**: no coordination between nodes

In addition, **channel sharing coordination** *must use the channel itself* — no other out-of-band channel.

15.1 Channel Partitioning Protocols

time division multiple access (TDMA)

Let there be n nodes.

Each node is equally allocated a **time slot** of length T, spanning a **time frame** of length $n \cdot T$, during which they get access to the channel; outside of which they are **idle**.

frequency division multiple access (FDMA)

FDMA is akin to TDMA but with **frequency bands** instead of time slots.

TDMA and FDMA are *collision-free*, *perfectly fair*, *decentralized*, but **inefficient** as unusued slots are wasted.

15.2 Controlled Access Protocols

polling protocol

One node is designated as the **master node**.

The master node polls each **slave node** in turn, allowing it to transmit a pre-determined maximum number of frames.

Polling is *collision-free*, *efficient*, *perfectly fair*, but **not decentralized** as the master node results in a single point of failure.

token passing protocol

In a ring network topology, a special **token** frame, is passed between nodes sequentially.

The node possessing the token can transmit a pre-determined maximum number of frames before forwarding the token.

Token-passing is *collision-free*, *efficient*, *perfectly fair*, and *decentralized*.

However, a **token loss** and a single **node failure** can be disruptive.

15.3 Random Access Protocols

A node with data to send should be able to transmit at full channel data rate R, with **no *a priori* coordination**.

Random access protocols must *detect* and *recover* from collisions.

slotted ALOHA

Like TDMA, time is divided into slots of length L/R, and synchronized at each node.

Nodes transmit only at the beginning of a slot, re-transmitting in the event of a collision in each subsequent slot with probability p.

Slotted ALOHA is *fair* and *decentralized*, but **not collision free**.

It is also **efficient-by-definition** when there is only 1 node, but only 37% efficient otherwise, as slots are wasted by collision and by being empty.

pure (unslotted) ALOHA

No synchonization and time slots are needed — nodes transmit immediately at any time.

In the event of a collision, they wait for a 1-frame transmission time before re-transmitting with probability p.

Unslotted ALOHA is worse than slotted ALOHA with 18% maximum efficiency as collision probability is higher.

In general, ALOHA is flawed, as transmission decision is made independently of other nodes.

carrier sense multiple access (CSMA)

Nodes defer transmission if the channel is busy, and transmit immediately if it is idle.

Collisions still occur due to propagation delay.

Both ALOHA and CDMA are flawed, in that they do not stop transmitting when collision is detected.

CSMA/CD (collision detection)

Like CSMA, but abort transmission if collision is detected, and re-transmit after a random delay, determined by **binary exponential backoff**:

1. after a collision, choose $k \in \{0, 1\}$
2. wait k time units before re-transmitting
3. after another collision, choose $k' \in \{0, 1, 2, 2^2 - 1\}$
4. wait k' time units before re-transmitting
5. after m collisions, choose $k'' \in \{0, 1, 2, 2^2 - 1, ..., 2^m - 1\}$
6. wait k'' time units before re-transmitting

The time unit is 512-bit transmission time for Ethernet.

A **minimum frame size** (64 bytes for Ethernet) is imposed to ensure increase the chance that collision is detected.

Both CSMA and CSMA/CD *efficient*, *fair* and *decentralized*, but **not collision-free**.

16 Media Access Control (MAC)

All nodes receive all frames in the same broadcast channel.

Thus, adapters filter received frames by their destination MAC address, passing only those addressed to itself to the network layer.

MAC addresses

Every adapter has a **MAC address** — a **48 bit hexadecimal sequence** burned into ROM.

The **first 24 bits** are allocated by IEEE and reserved for vendor identification.

The **broadcast address** is `FF-FF-FF-FF-FF-FF`.

Use `ifconfig` to find the MAC address of an adapter.

17 Ethernet

Local area network (LAN) is a computer network within a geographical area.

Wi-Fi (IEEE 802.11) and **Ethernet** (IEEE 802.3) are the most common LAN technologies.

Ethernet frame structure

preamble	dst	src	type	data...	CRC
8B	6B	6B	2B	46-1500B	4B

The data size is upper bounded by the link **maximum transmission unit (MTU)**, and lower bounded to ensure collision detection.

Ethernet uses **CRC-32** using a 32-bit generator.

The type field is necessary to specify the network layer protocol, which allows Ethernet to multiplex different network layer protocols.

preamble sequence

Each frame has a **preamble** starting with 7 bytes of `10101010`, or AA_{hex}, which is used as a wake-up signal.

The last byte in the preamble is `10101011`, or AB_{hex}, which serves as a delimiter indicating the start of the frame.

The square wave pattern of the first 7 bytes synchronize the sender's and receiver's clock rates.

Ethernet is **unreliable** as no acknowledgements are sent — dropped frames are recovered only by a higher layer protocol.

Ethernet uses **CSMA/CD** with exponential backoff.

17.1 Topology

bus topology

A single backbone coaxial cable to interconnect all nodes in a **broadcast LAN** with several drawbacks:

1. backbone cable is a single point of failure,
2. difficult to troubleshoot,
3. slow and not ideal for larger networks due to collision

star topology

Nodes are connected to either **hubs** or **switches**.

A hub is a **physical layer device** which re-creates the *bits* received, and boosts their signal to all nodes, but very slow and not ideal for larger networks.

A switch **link layer device** which stores and forwards *frames* with no collisions, and are much faster than hubs.

17.2 Ethernet Switches

Ethernet switches employ **selective forwarding** to forward frames only to single or multiple outgoing links via **CSMA/CD** depending on the destination MAC address.

Switches have other benefits:

- **transparency**:
  no frames are addressed to the switch, hosts remain unaware of their presence
- **plug-and-play**:
  no configuration needed
- **simultaneous transmission**:
  every node has a dedicated, direct, and *buffered* link **interface** using Ethernet to the switch — no collisions!
- **interconnectivity**:
  switches can be connected in a heirarchy to form a LAN, with a router connected to the root switch

selective forwarding

A switch knows that a host is reachable via an interface using a **switch table**.

Every entry consists of the **host MAC address**, **interface number**, and **TTL**.

To *populate* the switch table, it has to employ **self-learning** by recording entries for every incoming frame.

To *forward* a frame to a *known* host, the switch looks up the destination MAC address in the switch table, and forwards the frame to the corresponding interface: (**forwarding**).

To forward a frame to an *unknown* host, the switch broadcasts the frame to all interfaces except the incoming one: (**flooding**).

There is one exception: a switch receiving a frame on an interface that it would forward it back onto is simply dropped: (**filtering**).

18 Address Resolution Protocol (ARP)

ARP is used to determine the MAC address of a host given its IP address.

ARP tables

Every IP node maintains an **ARP table** which maps IP addresses to MAC addresses within a subnet, and can be listed with `arp`.

Every entry has a TTL, usually in the order of minutes.

ARP is **plug-and-play** — nodes create their ARP tables without intervention from a network administrator.

ARP queries

If A does not have B's MAC address in its ARP table, it broadcasts an **ARP query packet** containing B's IP address to the subnet.

This query packet has the destination MAC address set to the broadcast address `FF-FF-FF-FF-FF-FF`.

All nodes in the subnet receive the query, but *only* B responds with its MAC address by sending it as a **reply frame**.

ARP queries are used to determine destination MAC addresses if they aren't already cached in the ARP table, and used when sending frames in the *same subnet*.

sending frames outside of a subnet

Suppose host A wants to send a frame to host B in a different subnet, via a router R.

A will send the frame to R with the destination MAC address of the *receiving interface* at R.

The IP datagram will have the destination IP address of B.

At R, it will create the link-layer frame with the destination MAC address of B and source MAC address of the *transmitting interface* at R.

The Ethernet frames change between links, but the IP datagrams do not, as IP is end-to-end.

Part VI

Network Security

The tenets of network security are:

- **confidentiality**: only the sender and intended receiver should understand message contents
- **authentication**: sender and receiver want to confirm the other's identity
- **message integrity**: messages should not be altered without detection
- **access and availability**: services must be accessible and available to users

19 Cryptography

Cryptographic techniques allow senders to encrypt data (**ciphertext**) and receivers to recover the original data (**plaintext**).

A **key** is a string of input numbers or characters to the encryption or decryption algorithm:

- m: plaintext message
- $K_A()$: encryption algorithm with key K_A, such that $K_A(m)$ is the ciphertext
- $K_B()$: decryption algorithm with key K_B, such that $K_B(K_A(m)) = m$

19.1 Symmetric Key Cryptography

Symmetric key cryptography uses the same key K_S for encryption and decryption ($K_A = K_B$), but the algorithms $K_A()$ and K_B need not be the same.

However, the need for an agreed, shared key is a major drawback of symmetric key cryptography.

simple substitution ciphers

In a **Caesar's cipher**, each alphabet is shifted by a fixed number — only 25 possible shift values, which is easy to brute-force.

In a **monoalphabetic cipher**, each alphabet is mapped to a different alphabet, e.g. by a random permutation — vulnerable to statistical analysis and common n-grams.

In a **polyalphabetic cipher**, n ciphers are rotated in a predefined sequence known as a **cycling pattern**.

Encryption schemes can be broken in 3 ways:

1. **ciphertext-only attack**: only ciphertext is known, and can be analyzed
2. **known-plaintext attack**: ciphertext is *known* for a corresponding plaintext
3. **chosen-plaintext attack**: ciphertext can be *requested* for a corresponding ciphertext

block ciphers

Messages are encrypted in blocks of K bits, and each block is encrypted independently.

Blocks are mapped by a one-to-one substitution to a different block, such that there are $2^K!$ keys.

Data Encryption Standard (DES) is a block cipher with 56-bit keys and 64-bit blocks, but is vulnerable to brute-force attacks.

Advanced Encryption Standard (AES) replaced DES, and uses 128 bit blocks and 128, 192, or 256 bit keys.

19.2 Asymmetric Key Cryptography

Also known as **public key cryptography**, using different keys for encryption and decryption:

- **public key** K_B^+: known to everyone, used for encryption
- **private key** K_B^-: known only to the receiver, used for decryption

Similar to symmetric key cryptography, the plaintext $m = K_B^-(K_B^+(m))$.

However, it must be impossible to compute the private key K_B^- from the public key K_B^+.

key property of modular arithmetic

$(a \mod n)^d \mod n = a^d \mod n$

19.2.1 RSA

A **message** is a bit pattern that can be uniquely represented by an integer, such that encrypting a message is equivalent to encrypting a number.

creating RSA public/private key pairs

1. choose two large prime numbers p and q
2. compute $n = p \cdot q$, $z = (p-1)(q-1)$
3. choose e such that e and z are coprime (only common factor is 1), and $e < n$.
4. choose d such that $e \cdot d \mod z = 1$

The public key $K_B^+ = (n, e)$ and the private key $K_B^- = (n, d)$.

RSA encryption and decryption

The ciphertext $c = m^e \mod n$ is decrypted into $m = c^d \mod n$, where m is the message.

In RSA, the order of encryption and decryption is irrelevant: $K_B^+(K_B^-(m)) = m = K_B^-(K_B^+(m))$.

In practice, RSA is used only to transfer a symmetric **session key** K_S which is then used as the symmetric key in DES to encrypt data in a session.

This is because exponentiation in RSA is computationally expensive, and DES is significantly faster.

19.3 Hashing

Checksums and CRC are useful for detecting errors, but are vulnerable to attacks as they are not collision-resistant.

A **hash function** $H()$ takes a message m and returns a *fixed-length* message digest (**fingerprint**) h.

A **cryptographic hash function** is a hash function such that it is computationally infeasible to find a collision, i.e. $H(m_1) = H(m_2)$ for $m_1 \neq m_2$.

MD5 and **SHA-1** have been **cryptographically broken**, and are replaced by **SHA-2** and **SHA-3**.

19.3.1 Hash Functions

MD5 produces a 128 bit fingerprint using the `md5sum` command at the command line.

A small change in the message m results in a large change in the message digest $H(m)$.

However, this still does not preserve message integrity as an attacker can simply replace a message and its hash $(m, H(s))$ with their own $(m', H(s'))$.

19.3.2 Message Authentication Code

Messages are sent as $(m, H(m + s))$, where s is an **authentication key** known only to the sender and receiver.

19.4 Digital Signatures

Digital signatures must be **verifiable** and **unforgeable**.

digital signing

1. B signs m by encrypting with private key K_B^-, producing the signature $K_B^-(m)$
2. the message is paired with its signature and sent as $(m, K_B^-(m))$
3. the signature $K_B^-(m)$ is verified by decrypting with public key K_B^+, producing $K_B^+(K_B^-(m))$

$K_B^+(K_B^-(m)) = m \implies m$ is verified to have been signed by B, as whoever signed m must have used B's private key, K_B^-.

Optimization of digital signing involves signing the *hash* of the message instead of the entire message, and the hashes $H(m)$ and $K_B^+(K_B^-(H(m)))$ are compared at the receiver.

19.4.1 Public Key Certification

An attacker can sign a message with their own private key, and send it with their own public key and claiming to be someone else.

Therefore, to verify that a public key belongs to a particular person, a **certificate authority** (CA) maintains a public database of public keys and their corresponding identities.

However, an attacker can still intercept communication with the CA and alter it, so the CA signs its messages.

However, we still do not know the CA's public key, so such knowledge is made universal at the OS level as a list of **trusted root certification authorities**.

certification

Any entity E, e.g. person or router can prove its identity to the CA.

The CA creates a **certificate** binding E to its public key K_E^+, and signs it with its private key K_{CA}^-.

This certificate can be verified by decrypting with the CA's public key K_{CA}^+, producing $K_{CA}^+(K_{CA}^-(K_E^+)) = K_E^+$.

20 Firewalls

Firewalls isolate an internal network from the Internet, and selectively allow packets to pass through.

This can prevent **denial-of-service** (DoS) attacks and illegal modification or access of internal data.

This can also allow *only* authorized users to access the internal network.

However, they are vulnerable to **IP spoofing** and can become a bottleneck.

stateless packet filtering

A **router firewall** filters packet-by-packet and forwards or drops packets based on any of the following:

- source and destination IP addresses
- TCP/UDP source and destination port numbers
- ICMP message type
- TCP SYN and ACK bits (prevents **SYN flooding**)

Access control lists are a table of rules applied sequentially from top to bottom on incoming packets to determine whether to allow or deny a packet.

Part VII

Multimedia Networking

21 Multimedia Applications

Multimedia networking has 3 main applications:

1. streaming stored audio/video
2. conversational (2-way live)
3. streaming live (1-way live)

21.1 Streaming

Streaming stored video allows playback before the entire video is downloaded.

Videos are stored in **chunks** and transmitted faster than they are played.

The **continuous playout constraint** demands that playback must match the original timing of the video once playback begins.

Client-side buffering compensates for variable network delays (**jitter**).

buffering, formally

- $x(t)$: the variable fill rate
- $Q(t)$: the buffer of size B's fill level
- r: the playout rate

With an average fill rate of $\bar{x}$:

- $\bar{x} < r$: buffer will empty, freezing playback
- $\bar{x} > r$: buffer will not empty, as long as the **initial playout delay** absorbs the variability of $x(t)$

A high initial delay decreases the likelihood of **buffer starvation**, but in itself delays the start of playback.

Push-based streaming via **UDP** with its lack of congestion (rate) control allows the server to send at any rate appropriate for the client with a short playout delay to remove jitter.

Drawbacks include cost and complexity of media control servers, and firewalls often block UDP traffic.

Pull-based streaming via **HTTP GET requests** sends at the maximum rate via **TCP**, which is firewall-friendly, and the infrastructure is already in place and optimized for it.

However, this requires a longer playout delay, and the fill rate fluctuates due to TCP congestion control.

21.2 VoIP

End-to-end delay must be minimized (ideally < 150ms, at worst < 400ms), but without guarantees on delay and packet loss from the IP layer.

Packets are generated in **chunks** of 20ms at 64kbps during **talk spurts**.

The application layer header is encapsulated with each chunk into a UDP or TCP packet.

Any delayed IP datagram past a 400ms tolerable delay is dropped (**delay loss**), in addition to inherent **network loss** from network congestion.

fixed playout delay, q

Each chunk has a **sequence number** and **timestamp**.

Playout of each chunk is delayed by a constant q ms after it is generated — a larger q means less packet loss, while a smaller q is a better interactive experience.

No value of q can guarantee optimal performance — packet loss or wasted playout time is inevitable.

adaptive playout delay, q_i

The playout delay is adjusted dynamically to compensate for network conditions, but **only silent periods** are compressed or elongated.

Packet delay at time i is estimated via an exponentially-weighted moving average:

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

average delay deviation estimate,
$v_i = (1-\beta)v_{i-1} + \beta|r_i - t_i - d_i|$

playout delay,
$q_i = t_i + d_i + 4v_i$

where α and β are small constants, r_i is the time the chunk is received, and t_i is the time the chunk is transmitted.

d_i and v_i are computed per packet, but q_i is only applied per talk spurt (hence only silent periods are adjusted).

Now, with delays addressed, we address **packet loss** via **forward error correction**:

simple FEC

Create and send a redundant chunk for each n chunks by `XOR`-ing the n chunks.

At most one lost chunk can be reconstructed from the n chunks out of the $n+1$ sent.

This comes at the expense of an increased bandwidth requirement by $\frac{1}{n}$ and increased playout delay.

piggyback FEC

For every chunk that is sent, prepend a lower resolution version to the next chunk which is played.

This conceals *non-consecutively lost chunks*.

interleaving FEC

Subdivide each chunk into *n* subchunks and distribute among *n* chunks evenly, such that each packet now has one subchunk from each of the *n* chunk.

Packet loss is concealed by **packet repetition** or **packet interpolation**.

This does not incur any redundancy overhead, but always increases playout delay.

21.3 DASH

Simple HTTP streaming GETs an entire video file which may be wasteful and requires a large buffer.

Also, all clients receive the same video encode regardless of their device and network bandwith, which is not ideal.

dynamic, adaptive streaming over HTTP (DASH)

The server divides a video file into multiple chunks, and each chunk is encoded at multiple bitrates.

The **manifest file** provides URLs for each different encoding, which the client downloads.

The client uses an **adaptive bitrate algorithm** to measure bandwidth and requests the best encoding for its current sustainable bandwidth.

DASH is simple and scalable, but cannot provide the low latency needed for interactive two-way applications.

Content distribution networks (CDNs) store and serve multiple copies of videos at multiple geographically distributed sites.