INS: Internet Systems

Important Notes About These Notes

These notes were written by me, Mark Ormesher, during my revision for Computer Science exams at King's College London. I am publishing them here to help my classmates and other students in the years below me. To the best of my knowledge they are correct.

- These notes are **not endorsed** by King's College London, the module lecturers, or any member of College staff.
- These notes are **not checked** by any qualified individual, and I offer no guarantee as to how accurate or complete these notes are. They are offered for free, as-is, and I accept no responsibility for how you choose to use them.
- These notes were relevant for my year (2016/17) but **the content for your course may be different**. Check your lecture slides, syllabi, tutorial guides, etc.
- These notes were produced for my own personal use (it's how I like to study and revise). That means that some annotations may by irrelevant to you, and some topics might be skipped entirely.
- Feel free to **share** these notes, however please only share a link to the repo (see the link below), not individual files.

Contents

Brief Introduction to Everything

Architecture

Networks consist of **interconnected nodes** that fulfil various roles: simple hosts, routers (which communicate with and send messages between hosts), servers (which provide some service), etc.

A **connection** does not imply **communication**: a **protocol** is needed for that. A protocol is an agreed schema for sending messages between nodes.

A **Local Area Network (LAN)** connects machines within some finite localisation, usually a physical location (e.g. an office building).

A **Wide Area Network (WAN)** connects various LANs together over larger areas, usually not directly (i.e. a small number of connections between LANs).

Routing is required because data may have to travel between various intermediate nodes. These nodes need to know that the data is not for them, and where they should send it. **Internet Protocol (IP) addresses** are used for this.

History of the Internet

The very first point-to-point connections were created by the **Advanced Research Projects Agency (ARPA)** in the 1960s. ARPA found some problems with early networking models:

- Centralisation created bottle-necks and high-risk failure points.
- Various protocols in use in different LANs created compatibility issues.
- Commercial protocols were expensive to use.

They created **ARPANET** to solve these problems. ARPANET and the current Internet is fundamentally unreliable:

- Unreliable delivery means that only a best effort is made to deliver packets (i.e. no guarantee is made).
- Packets can be dropped with no notification to sender or receiver.
- Software must be able to deal with lost data.

Addressing

Resource addressing on the Internet is **hierarchical**: addresses are formed of **nested layers**, from port numbers on a machine and its physical wiring, up to access layers, inter-network layers and application layers.

Addresses only need to be unique **within their layer** and can be duplicated in other layers. The whole 'chain' of a given address is what needs to be unique.

See more: Addressing, page 11

Messaging Protocols

Messaging protocols are needed so that machines can be programmed to **understand each other**. Messages in one protocol can be **embedded** into another, such as:

$$Header_{protocol1} \cdot Message_{protocol2} \cdot Tail_{protocol1}$$

Messages will usually be **chunked**, so **flow control** is needed to deal with bandwidth, scheduling, routing, etc. The buffer size of the receiver must be considered. The receiver may drop packets that are sent too fast and ask the sender to slow down – how this happens will depend on the protocol, but **Transmission Control Protocol (TCP)** has this capability.

See more: Transmission Control Protocol (TCP), page 24

Exchanging and Understanding Content

The term **hypertext** was coined in the 1960s and was later extended to **hypermedia** to include sound, video, and other ways of presenting information.

Sir Tim Berners Lee later combined several technologies to create the infrastructure on which the web operates today:

- A language that allowed users to write hypertext documents (**HyperText Markup Language**, or **HTML**).
- A protocol to send those documents over the Internet when a link was followed (**HyperText Transfer Protocol**, or **HTTP**).
- These are both public standards, allowing anyone to publish web content.

XML and HTML

XML (eXtensible Markup Language) and **HTML (HyperText Markup Language)** are both markup languages for the representation of information. A markup languages provides **annotations** for text to denote **structure** and **display**.

Security and Integrity

Public and Private Keys

Public/private key encryption can be used to send encrypted information **without ever sharing private information** or reserved secrets in advance.

Digital Certificates

Digital certificates are used to prove that content has come from the person it claims to be from (commonly used for websites). A popular certificate format is **X.509**, which contains three parts:

- The certificate details
  - Serial number, validity period, issuer and owner details, and the public key of the owner.
- The signature of the certificate.
- The algorithm used to sign the certificate.

Web Services

Web services can consist of a **private implementation** fronted by a **public interface** that uses the **Web Service Definition Language (WSDL)**.

The **Simple Object Access Protocol (SOAP)** can be used to exchange requests and responses. A request consists of an **HTTP** header and a **XML** message conforming to the **WSDL** interface.

Service definitions is usually layered, as below:

- Service Definition 1
  - Port Type (Interface) 1
    - Operation 1
      - Input format
      - Output format
    - Operation 2
      - Input format
      - Output format
    - Operation n
      - ...
  - Port Type (Interface) 2
    - ...
  - Port Type (Interface) n

        - ...
- Service Definition 2
    - ...
- Service Definition n
    - ...

Semantic Web

The goal of semantic web is to make content **machine-understandable**. It requires using **app-specific mark** up in web pages and create **agreements on mark-up concepts and practises** amongst distributed users.

An agreed set of concepts and meanings in a parsable form is an **ontology**.

Resource Description Framework (RDF) is an XML-based specification to describe a particular resource. A set of RDF statements can form an **RDF graph** that contains values and resources at its nodes, and predicates ('knows', 'controls', 'owns', 'observes', etc.) along its edges.

Paradigm Shifts

Software based systems enable very quick modifications.

Cloud-based services are changing the ways people and industries consume technology.

- **SaaS (Software as a Service)**: whole applications can be rented or subscribed to (such as SalesForce CRM).
- **PaaS (Platform as a Service)**: platforms can be rented on which custom software can be deployed (such as Google Cloud Apps and Microsoft Azure).
- **IaaS (Infrastructure as a Service)**: processing and storage capacity can be rented (such as AWS and Rackspace).

Internet Architecture

- The Internet is huge - how can administration be divided into manageable chunks?
- The Internet is distributed - how can changes be implemented without breaking things or requiring changes elsewhere? (i.e. low impact-radius changes)

Architecture Goals

- Connect existing networks together.
- Be robust with regards to small-scale (individual links) and large-scale (entire subnetworks) failures.
  - Routing functionality should adapt to these situations.
- Allow distributed management.
- Support multiple types of content and service.
- Allow host attachment with little effort.
- Be cost-effective in terms of header overhead, re-transmission, router capabilities required, etc.

Architecture Layers

The Internet is organised as a set of layers. Many issues must be solved for a successful Internet application (routing, reliability, data formatting, flow control, etc.); **each layer solves one or a few** of these issues, and most layers have **multiple implementations**. This allows for different combinations of technologies to be selected to best suit a particular problem.

The main layers are, from the bottom up:

- **Physical** - the actual connectivity (copper, fibre, radio, etc.)
- **Access** - defines how to deliver data between two devices on the same network (most commonly Ethernet)
- **Network** - defines how to route messages across networks
- **Transport** - defines how to provide reliable communication, so that data will not be lost or corrupted
- **Application** - defines how programs instruct messages to be sent by the lower layers (encryption, compression, etc.)

This course focuses on the top three layers (network, transport and application).

Protocols

A protocol is a way of communicating. It specifies how to **express information**, how to **respond** when given certain requests or commands, and the **forms of requests or commands** to expect.

Each layer can be implemented by **multiple alternative protocols** that **guide the communication of hosts** to achieve the layer's purpose.

Network Layer

- **IP (Internet Protocol)** is the main protocol that is used.
  - IP can be used for transferring messages between hosts anywhere on the Internet.
  - *See more: Internet Protocol (IP), page 17*
- **ICMP (Internet Control Message Protocol)** is also sometimes used to augment IP.
  - *See more: Internet Control Message Protocol (ICMP), page 23*

Transport Layer

- **TCP (Transmission Control Protocol)**
  - Provides reliability measures (acknowledgements and flow control), sessions (container for multiple communications), multiplexing (bundling communications for multiple applications into one transmission)
  - *See more: Transmission Control Protocol (TCP), page 24*
- **UDP (User Datagram Protocol)**
  - Minimal overhead.
  - Some reliability measures provided by checksums, but otherwise unreliable.

Process/Application Layer

- HTTP (HyperText Transfer Protocol)
- TELNET
- SMTP (Simple Mail Transfer Protocol)
- FTP (File Transfer Protocol)
- POP (Post Office Protocol)
- DNS (Domain Name Service)
- DHCP (Dynamic Host Configuration Protocol)
- etc.

Edge-Oriented Architecture

The Internet's success is due to its edge-oriented approach to architecture: a **connectionless, packet-forwarding infrastructure** (dumb network) that positioned **higher-level functionality at the edge** of the network for robustness.

Intelligent edges and a dumb network **keep the infrastructure as simple as possible**. Complexity of the core network is reduced, and new applications can be easily added.

Addresses in this system use **fixed sized numerical values** with **simple structures**. They are applied to physical network interfaces, so they can be used for **naming** a node and **routing** to it.

Packet Transmission

```
HTTP > TCP > IP > Link Layer > Copper
```

- HTTP encodes the message of data
- TCP adds its header, packet number, timeout settings, etc.
- IP adds host and destination information, routing information, etc.

IP Fragmentation

Different link layer technologies can carry **packets of different sizes**. The maximum packet size is called the **Max Transfer Unit (MTU)**. The IP is encapsulated in the link layer, so this MTU **limits the size of the IP packet**.

If the outbound link has a smaller MTU than the IP packet the router wants to send, **fragmentation** is the solution: the packet is broken up, each one is sent, and the receiver re-assembles them.

See more: Fragmentation, page 17

Addressing

- How can hosts identify each other when they are not directly connected?
- How can addressing schemes handle various numbers of hosts in an organisation?

Networks

- For two hosts to communicate, there must be a connection between them (cable, wireless, etc.).
- A network is a set of computers **connected directly or indirectly**.
  - A computer that is part of a network is called a **node**.
  - A node from which messages are sent and/or received is called a **host**.
  - Other kinds of nodes are **routers**.

Routing and Addresses

Generally, one host wants to communicate with another host that it is not directly connected to. We need routing to achieve this:

- A path is found along a series of connected nodes.
- Data is sent along the resulting path until it reaches the destination (or fails).

How can a sender identify which receiver it needs to send to, so that a route can be found? **Addressing**.

IP Addresses

In the global Internet, each and every host and router needs **one globally unique address**. Technically, IP addresses are associated with an **interface within a machine**, not a host.

Primarily **IPv4** is used; **IPv6** is being deployed slowly.

Both types of IP addresses use a **hierarchical structure**:

- The Internet is divided into networks.
- Each network has a network prefix.
- Each host in the network has a host identifier.

Together these make the IP address.

Network Prefixes

Global routers pass each message down to the **local network router(s)** for the given network prefix, which then passes the message to the host specified by the host ID.

Network prefixes assigned by **ICANN (Internet Corporation for Assigned Names and Numbers)** and **NICs (Network Information Centres)**. Owners of the prefixes assign host identifiers within them.

Some network prefixes are guaranteed not to be allocated, such as those used for technical purposes, internal networking (`192.168.x.x`), etc.

When routing, the whole address is always passed, so to determine which parts are the network prefix and host ID, routers need to know how long the prefix is. The length of a prefix in an address is indicated with a slash and the length, such as `143.326.3.26/16` for a 16-bit prefix.

IPv4 Addressing

IPv4 addresses are **32 bits long**, giving $2^{32} \approx 4.3bn$ addresses. Within any network, two addresses are **reserved**:

- The prefix followed by **all 0s** (binary) - this is the address of the network itself.
- The prefix followed by **all 1s** (binary) - this is the network's broadcast address.

For example, if the network prefix is 23 bits long then there are 9 left for the host ID. The network can therefore hold $2^9 - 2 = 510$ hosts.

Network Prefix Classes

To provide flexibility to support **different network sizes**, three different classes of addressing were created (**A**, **B** and **C**), plus two non-standard classes for multicasting (**Class D**) and experimentation (**Class E**).

The class of an address and the subnet mask determine how many of the 32 bits belong to the network prefix and how many belong to the host ID.

- **Class A (/8)**
  - Binary IP starts with `0...`
  - Used for very large networks.
  - 8-bit network prefix, giving $2^7 - 2 = 126$ possible /8 networks.
    - `0.0.0.0` is reserved for the default route.
    - `127.0.0.0` is reserved for local loopback functions.
  - 24-bit host ID, $2^{24} - 2 = 16,777,214$ hosts per network.
  - Decimal address range 1 to 126.

- **Class B (/16)**
  - Binary IP starts with `10...`
  - Used for large networks.
  - 16-bit network prefix, giving $2^{14} = 16,384$ possible /16 networks.
  - 16-bit host ID, $2^{16} - 2 = 65,534$ hosts per network.
  - Decimal address range 128 to 191.
- **Class C (/24)**
  - Binary IP starts with `110...`
  - Used for smaller networks.
  - 24-bit network prefix, giving $2^{21} = 2,097,152$ possible /24 networks.
  - 8-bit host ID, $2^8 - 2 = 254$ hosts per network.
  - Decimal address range 192 to 223.

Classful vs. Classless Addressing

Classful addressing has huge gaps between class sizes, so in 1993 **Classless Inter-Domain Routing (CIDR)** was standardised by the IETF. In CIDR-ised networks, the **network prefix can be any number of bits long**.

For example, to serve 2000 hosts, addresses in the form `a.b.c.d/21` could be assigned to leave 11 host ID bits, giving $2^{11} - 2 = 2046$ hosts.

Today, **address classes are ignored** and routers are explicitly told the prefix length.

Subnets

Subnets split up a network to give finer control and separation. With subnets, addresses take on a three-level structure of **network prefix**, **subnet ID**, **host ID**.

Subnet Masks

In binary format, **1s represent the network number** and **0s represent the host number**.

- Prefix = `IP & Subnet Mask`
- Host = `IP & (~Subnet Mask)`

Subnet Example 1

An organisation has been assigned the network prefix `193.1.1.0/24` and wants 6 subnets for up to 25 hosts each.

- Network prefix: 24 bits.
  - Mask (bin): `11111111.11111111.11111111.00000000`
  - Mask (dec): `255.255.255.0`
- 3 bits are needed to define 6 subnets, so the extended network prefix has 27 bits.
  - Mask (bin): `11111111.11111111.11111111.11100000`
  - Mask (dec): `255.255.255.224`
  - This allows $2^3 = 8$ subnets, so there are 2 available for future growth.
- 5 bits are left for the host ID.
  - $2^5 - 2 = 30$ possible hosts.

Subnet Example 2

An organisation has been assigned 140.25.0.16/16 and needs to create subnets to support up to 60 hosts each.

- To define 60 hosts, the host ID needs 6 bits ($2^6 - 2 = 62$).
  - This is tight, so 7 bits are selected to give $2^7 - 2 = 126$ hosts per subnet.
- The network prefix has 16 bits and the host ID has 7, leaving 9 bits for the subnet ID.
- The extended network prefix is now $16 + 9 = 25$ bits long.
  - Mask (bin): `11111111.11111111.11111111.10000000`
  - Mask (dec): `255.255.255.128`
  - This allows $2^9 = 256$ subnets.

Variable-Length Subnets

Fixed-length subnets create problems. An organisation might need many subnets, but as the subnet ID grows, the number of possible hosts shrinks. They may also need subnets of different sizes.

Using **variable subnet ID lengths**, the host ID space can be iteratively divided into large blocks first, then smaller ones. As the number of bits used for the subnet ID varies, so too must the subnet masks.

Variable-Length Subnet Example

The network `a.b.c.0/24` needs the following five subnets:

- Subnet A requires 90 hosts.
- Subnet B requires 36 hosts.

- Subnets C, D and E require 12 hosts each.

A `/24` network can accommodate $2^8 - 2 = 254$ hosts, so there is enough space!

- First, we can use 1 bit to split A from the rest of the address space.
  - Subnet A has a `/25` address.
  - `a.b.c.0xxxxxxx`
- Then the second largest (B) needs another bit to be separated.
  - Subnet B has a `/26` address.
  - `a.b.c.10xxxxxx`
- Finally the smaller subnets (C, D and E) need two more bits to be separated.
  - Subnets C, D and E have `/28` addresses.
  - `a.b.c.1100xxxx`
  - `a.b.c.1101xxxx`
  - `a.b.c.1110xxxx`
  - `a.b.c.1111xxxx` (spare)

IPv6 Addressing

- Increased address space ($2^{128} \approx 2.3 * 10^{38}$ addresses).
- Network-layer encryption and other security features.
- Better flow control for better end-to-end service quality.
- Supports new features for new applications.

Addresses are 4 times as long as IPv4 (128 vs. 32 bits), but the header is only twice the size. Addresses are expressed in **8-word hex statements** with the same prefix-length notation (e.g. `2001:0db8:0000:0042:0000:8a2e:0370:7334/64`). All local IPv6 networks are /64.

Three types of IPv6 address exist:

- Unicast - single interface.
- Anycast - any host in a network.
- Multicast - every host in a network.

There are no address classes like IPv4, but two prefixes are reserved:

- `11111111...` is used for multicast.
- `1111111010...` is used for link-local unicast.

Two addresses are reserved:

- `0::0` means 'the host has not been assigned an address'.
- `0::1` is used for loopback (for a host to send messages to itself).

Internet Protocol (IP)

IP is the **network layer** for the Internet - a host-to-host packet delivery service.

The key challenges for IP are:

- How can we send a message to the right destination?
- How can we send messages larger than some networks are able to handle?
- How can we know which protocols were used to send the message, so that we can interpret it correctly?

There are several issues that IP does not solve:

- It is **unreliable**.
- Messages may get **corrupted in transit**.
- Message fragments may arrive out of order, arrive duplicated, or not arrive at all.

Higher-level protocols like **TCP** add reliability to IP (*see more: Transmission Control Protocol (TCP), page 24*.

Fragmentation

Every physical network has a limit to the maximum message size it can transmit: it's **Maximum Transfer Unit (MTU)**. To work around this, any message can be **split into fragments** by the sender, each of which can be sent individually. Fragments are reassembled by the receiver (usually by the **TCP/IP** network driver).

IP adds a header to every fragment. Some fragments may take different routes to the destination. In IPv4, **fragments may be further fragmented** when passed to a network with a lower MTU.

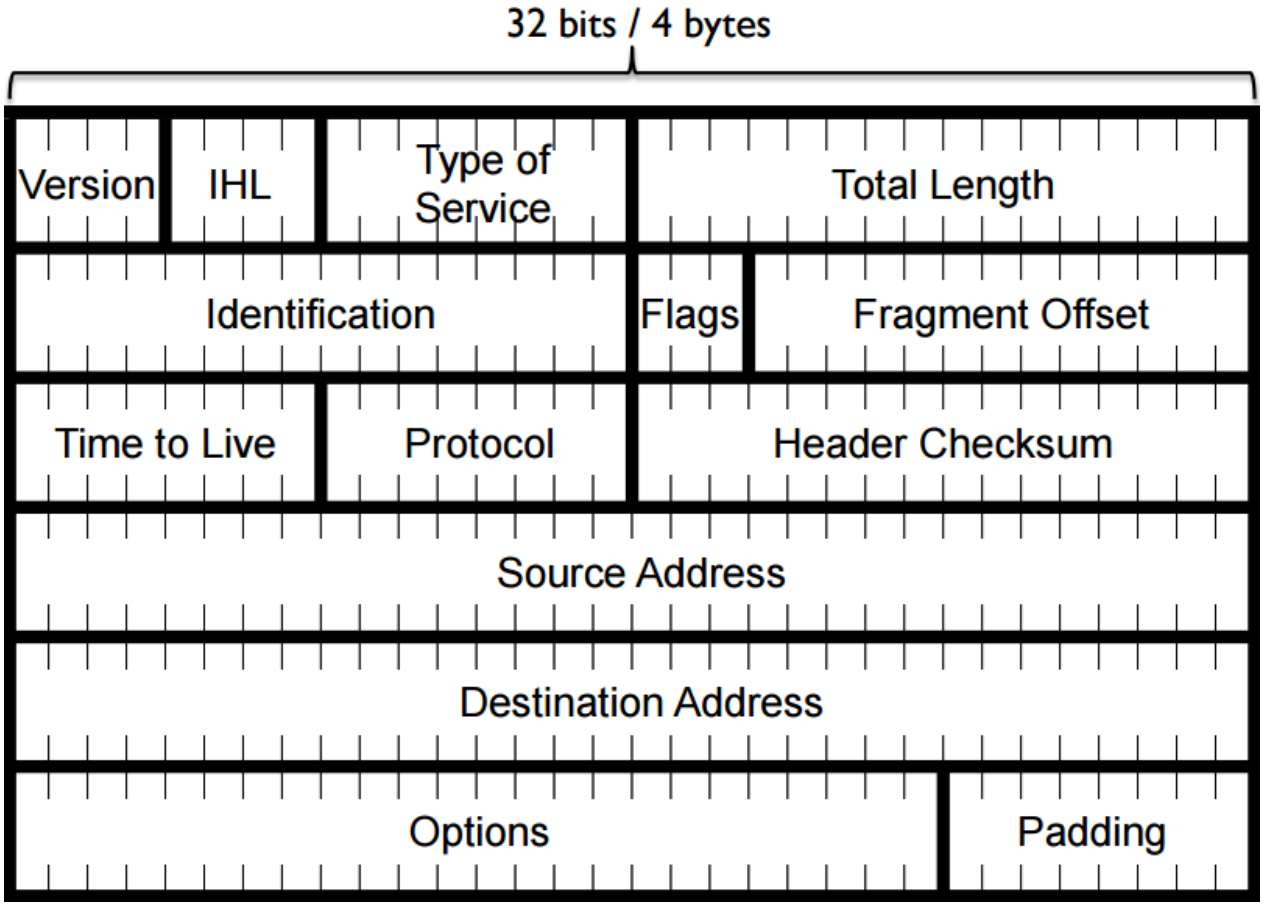
Fragments must be multiples of **64 bits (8 bytes)**, except the last one.

In general, fragmentation is a bad thing: it adds significant overhead (more header data) and delays (fragmentation/reassembly). It is necessary so that multiple networks can connect to an **open Internet**, regardless of their physical restrictions (MTUs).

See more: Path MTU, page 22

IPv4 Header

The IPv4 header consists of 5 32-bit (4-byte) words, with more 32-bit words occasionally added to specify options.



- Version (4 bits)
  - `0100` or `0110` for IPv4 or IPv6.
  - The same is used at the start of the IPv6 header.
- Internet Header Length (IHL) (4 bits)
  - Specifies how many 32-bit words are in the header.
  - Minimum value of 5.
- Type of Service (8 bits)
  - Originally used for specifying the type of service required (to favour throughput vs. reliability).
  - Redefined by modern protocols for congestion handling.
- Total Length (16 bits)
  - Total length **in bytes** of the packet/fragment, **including the header**.
  - Minimum value of 20 (for the minimum 5-word header).
- Identification (16 bits)
  - ID for this message.
  - Every fragment with the same ID is part of the same message.
- Flags (3 bits)
  1. Reserved, always zero

2. **Don't Fragment (DF)**: tells the router not to fragment this packet. If the packet exceeds the MTU and DF is set, the packet is dropped and ICMP (*see more: Internet Control Message Protocol (ICMP), page 23*) is used to send an error message.
3. **More Fragments (MF)**: specifies that there are more fragments from the same message following this one.

- Fragment Offset (FO) (13 bits)
  - Specifies where this fragment fits into the original message.
  - Measured in the number 8-byte chunks that go before this fragment.
    - E.g. FO = 3 means that this fragment starts $3 * 8 = 24$ bytes into the message.
  - Allows values up to 8191.
- Time to Live (TTL) (8 bits)
  - Specifies how long this packet can remain in the system before reaching the receiver.
  - Every host must reduce this counter by 1 when routing the packet.
  - Packets are **dropped when TTL = 0**. This stop packets from getting stuck in loops.
- Protocol (8 bits)
  - Specifies the **transport layer** sitting on top of IP.
  - Main protocols: TCP = 6; UDP = 17; ICMP = 1.
  - Protocol IDs are **shared between IPv4 and IPv6**.
  - IDs are assigned by the Internet Assigned Numbers Authority (IANA), part of the Internet Corporation for Assigned Names and Numbers (ICANN).
- Header Checksum (16 bits)
  - IP does nothing to prevent corruption, so the checksum allows higher-level to verify the **integrity** of a packet header.
  - The checksum must be updated by any node that changes the header (such as updating TTL).
  - *See more: Header Checksum, page 20*.
- Source/Destination Addresses (32 bits each)
  - IP addresses of sender and receiver.
  - *See more: IPv4 Addressing, page 12*.
- Options and Padding (32-bit words)
  - Optional arguments and flags used by IP processing software.
  - **Variable number of bits**, but always padded to 32-bit words with zeros.
  - Covers options for routing, tracing, etc., but **not used very often**.

Header Checksum

The header checksum verifies the **integrity** of a header, but **not authenticity** (i.e. it is for corruption detection, not security).

It is computed as follows:

- The header (excluding the checksum) is considered as a series of 16-bit words.
- The one's-compliment sum of the words is computed.
- The one's-compliment of that sum is computed - this is the checksum.

To verify a header, the one's-compliment sum of all 16-bit words (including the header) is computed - if the header is not corrupted, this value will be zero.

Effects of Fragmentation

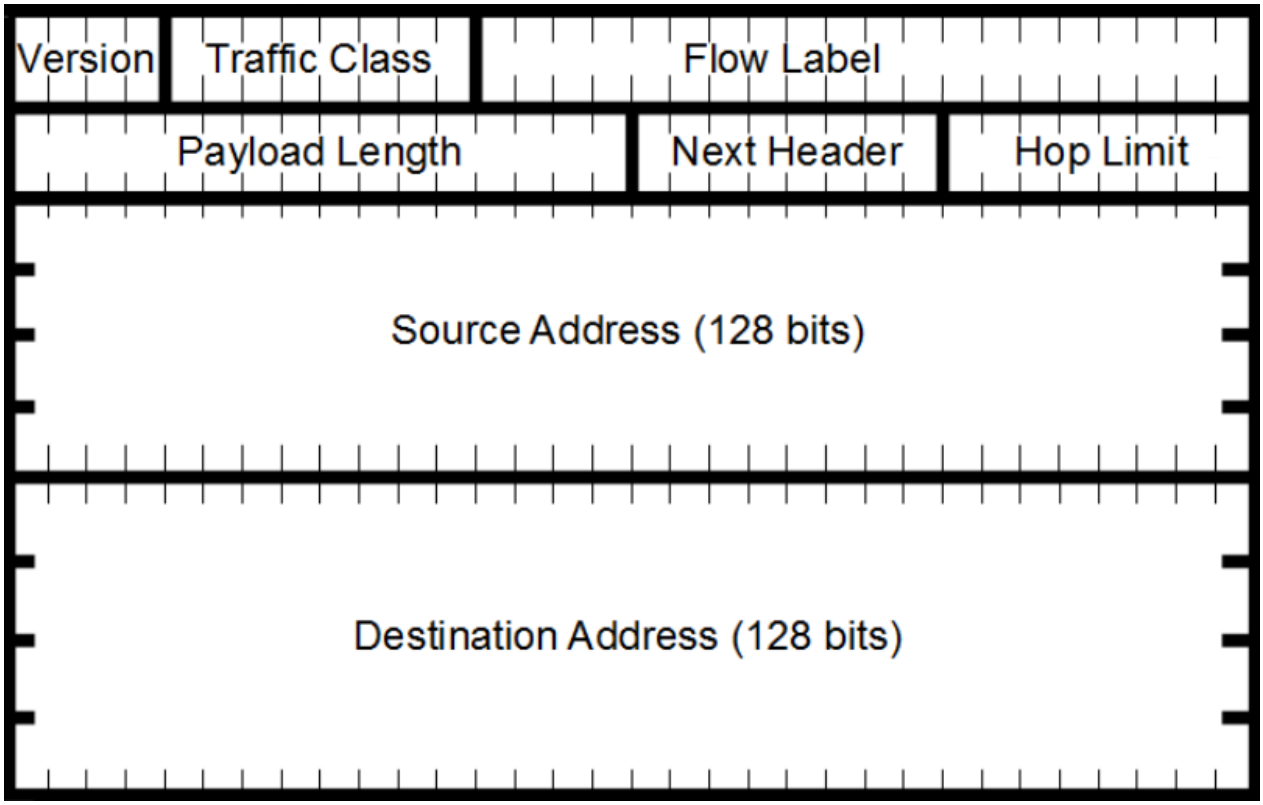
When a packet is fragmented, some **header values must change**.

- 'Total length' will change.
- The **MF** flag will be set to `1` for all fragments except the last one, which will keep its original value.
- The **FO** will change for all fragments except the first one.
  - The new FO for each fragment will be the original packet's FO, plus the fragment's offset (in 8-byte chunks) from the start of the original packet.

Note: fragments must always remain in multiples of 64 bits (8 bytes), except the last one.

IPv6 Header

Almost everything changes for the IPv6 header; **'version' is the only field that does not change**. IHL, flags, FO, header checksum and options/passing are removed entirely; everything else changes it's name and meaning. IPv6 headers are fixed at **80 bytes**.



- Traffic Class (8 bits)
  - Similar to 'Type of Service' in IPv4.
- Flow Label (20 bits)
  - Similar to 'Identification' in IPv4.
- Payload Length (16 bits)
  - Similar to 'Total length' in IPv4, but **excludes the header(s)**.
- Next Header (8 bits)
  - Similar to 'Protocol' in IPv4.
  - Specifies the next header on the packet (works like a singly-linked list).
  - *See more: Extension Headers, page 21*.
- Hop Limit (8 bits)
  - Similar to 'TTL' in IPv4.
- Source/Destination Addresses (128 bits each)
  - As before, just bigger.
  - *See more: IPv6 Addressing, page 15*

Extension Headers

IPv6 allows for options, but not directly inside the header. The main header can be followed by **zero or more extension headers**, all in the same IPv6 header format, **chained together with the 'Next header' field**.

Headers **specify a header protocol number** on the 'Next protocol' field to indicate that another 80-byte header follows it; the last header (which might be the first one) specifies a **higher-level protocol number to end the chain** and start the payload.

Examples:

IPv6 Header, Next header: TCP	TCP Header & Body

IPv6 Header, Next header: Routing	Routing Header, Next header: TCP	TCP Header & Body

IPv6 Header, Next header: Routing	Routing Header, Next header: Fragment	Fragment Header, Next header: TCP	TCP Header & Body

Fragment Headers

As shown above, fragmentation data is stored in extension headers in IPv6. These work the same way as in IPv4 and are only included when the message has been fragmented.

Path MTU

IPv6 **does not use fragmentation in transport**: it requires the sender to ensure messages are **sufficiently fragmented** to cross the networks before sending them.

This is done by fragmenting messages according to the **path MTU**: the minimum MTU along the path from the sender to the receiver.

Note: the path MTU can be used with IPv4, but has to be implemented by a higher protocol like TCP (this is what the DF flag can be used for).

Discovery Algorithm

1. Assume the path MTU is the MTU of the first hop in the path (the link to the first router).
2. The sender fragments the message to the current assumed path MTU and sends the first fragment.
3. If the fragment reaches a link where it exceeds the MTU:
   (a) An ICMP 'packet too big' error is send back to the sender with the lower MTU.
   (b) The sender updates the assumed path MTU to this lower value.
   (c) Go back to step 2.
4. When the first fragment reaches the destination, the path MTU is known and the rest of the message is sent.

IPv4 Don't Fragment (DF) Flag

If a fragment with the DF flag set reaches a link with an MTU lower than its size, an ICMP error is sent back to the sender (**ICMP code 4: fragmentation needed**).

Internet Control Message Protocol (ICMP)

This protocol sits on top of IP and is used to report **error messages**, **routing information** and other IP processing messages back to the sender.

ICMP messages include a **message type and payload** where applicable. Some example message types:

- Destination unreachable error (payload specifies which hop failed).
- Echo request/echo response (used for ping).
- Redirection.
- Time exceeded.
- Router advertisment and router solicitation.

Transmission Control Protocol (TCP)

Many protocols are **encapsulated within IP datagrams** (via an inner header in IPv4 or extension header in IPv6) - TCP is one of them. TCP's main features are:

- **Reliability** - TCP guarantees delivery via acknowledgement and re-trying.
- **Multiplexing** - two hosts can have multiple 'conversations' without getting confused over which messages belong to which.
- **Flow/congestion control**.

Note: generally different protocols aim for **clean separation between layers** (*see more: Architecture Layers, page 8*), but with TCP this isn't quite the case, because its checksum (*see more: Header Checksums, page 31*) uses components of the IP header.

TCP Connections

Ports

TCP conceptually **divides a host's network interface into ports**, each of which can hold a separate channel of conversation. This is how **multiplexing** is achieved.

Some ports are reserved:

- Port 20/21 - FTP
- Port 25 - SMTP
- Port 80 - HTTP
- Port 443 - HTTPS

Sockets

A socket is a combination of a host's **IP address and port number**. Every TCP connection is between two sockets (i.e. two hosts using specific ports).

Initially, **a server will listen** on a given socket (e.g. a web server listening on port 80). **Clients can initiate** a connection to the socket offered by the server. The connection is usually initiated by something at application level, like a web browser.

Multiple clients can connect to the **same server socket** from different client sockets.

Once a pair of sockets is conneced, data can be **sent in both directions** between the client and server (i.e. the connections are **full-duplex**).

Note: Clients and Servers

A **server** in this context is a host that is ready to receive requests and send data (later referred to as a **sender**). It signals to its TCP software that is ready by sending a **passive OPEN request**.

A **client** in this context is a host that is sending requests and receiving data (later referred to as a **receiver**). It initiates a connection by sending an **active OPEN request** to a server.

Note: both roles could be filled by the the same host (such as a web browsers sending requests to a server running on `localhost`).

Connections

In TCP, hosts must establish a connection, requiring **set-up** and **tear-down**. Data can only be sent between hosts within a connection. There are a few reasons for this:

- It allows 'extra' information to be shared between hosts.
- It enables reliability.
- It allows resource reservation to ensure quality of service (more applicable on the server-end of the connection).
- It allows for flow control and congestion management.

A TCP connection is a kind of **session**; many other protocols use sessions as well.

Connection Set-up

A **3-part handshake sequence** of messages between a client and server is required to set up a connection before sending data.

1. The client sends a `SYN` (**synchronisation**) message to the server.
2. The server replies with a `SYN ACK` message to acknowledge the first message.
3. The client replies with an `ACK` message to acknowledge the acknowledgement.

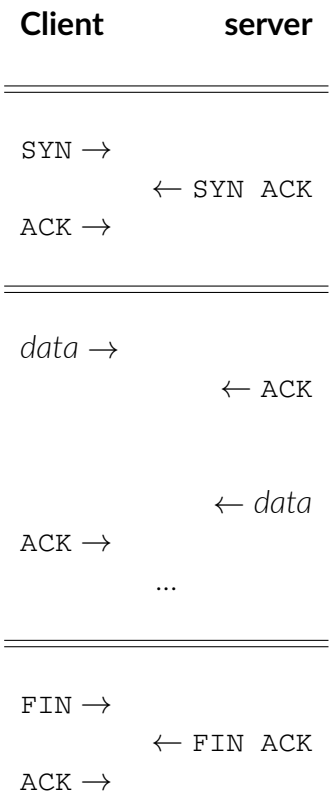
Connection Teardown

A **3-part handshake sequence** is also used to close a TCP connection.

1. The sender sends a `FIN` message to **finalise** the connection.
2. The receiver replies with a `FIN ACK` message to acknowledge the first message.
3. The sender replies with an `ACK` message to acknowledge the acknowledgement.

Connections can be **closed from either side**. Connections in which only one end point has closed are in a **half-closed state**.

Sequence Summary



Flow Control

A host can only receive and process data at a given rate, which **may vary** depending on processing load. If the rate of arriving data is too fast then eventually buffers will fill up and either **existing data will be overwritten** or newly **arriving data will be dropped**.

This is resolved within TCP by **allowing a receiver to tell the sender how much data it can handle**. The sender can then control the rate at which it sends the data to suit the server.

Segmentation and Acknowledgement

One part of the solution to flow control is **segmentation: splitting the message** to be sent into multiple segments, each of which can be transmitted separately.

This is similar to IP fragmentation (*see more: Fragmentation, page 17*), but at a higher level and for different reasons. IP splits messages to cope with the physical limits of network access layer protocols; TCP splits messages to cope with the limits of the receiving host, for flow control, and to help with reliability.

IP fragmentation is expensive because dropped segments will be re-tried, so **TCP tried to choose a segment size to match the path MTU** (*see more: Path MTU, page 22*).

Sequence Numbers

Every byte within a message from a client to a server has a **unique sequence number**, which is **used to re-assemble** the message from its segments.

For any given connection, there will be an **Initial Sequence Number (ISN)**, used as a point of reference for all bytes within the connection. The ISN is determined during the set-up handshake (*see more: Connection Set-up, page 25*).

The first byte send with have a sequence number of $ISN + 1$. A segment will contain all of the data in the range of sequence numbers $ISN + a$ to $ISN + b$.

See more: TCP Header, page 29.

Reliability and Acknowledgements

The receiver of a message will **acknowledge every segment** that it receives, using the sequence number to identify bytes that have been received. The sender will use these acknowledgements to **re-try** any segments that it believes have been dropped.

An acknowledgement from the receiver to the sender states that the receiver has **received all of the data before a given sequence number**.

For example, if the receiver receives the segments with sequence numbers $[1..20]$ and $[21..30]$, it will send **31**. If the receiver receives the segments with sequence numbers $[1..20]$ and $[41..50]$, it will send **21**; the server will know that the segments with bytes 21 and onwards may have been dropped and should be re-tried.

Re-send Strategies

When will a sender know when to re-send a segment? Two approaches:

- **Time-out**: a sender may re-try a segment if it has not received an appropriate acknowledgement within a given time frame.
- **Repetition**: a sender may re-try a segment X if it receives acknowledgements suggesting that other segments are being received but X was dropped.
  - For example, if the receives segments $[1..20]$, $[31..40]$, $[41..60]$ and $[61..80]$ it may send the acknowledgement for 21 four times, which suggests that the segment starting at 21 is missing.

Some implementation may combine these methods.

Maximum Segment Size (MSS)

This is the **maximum amount of data that can be accepted by a receiver** at once - segments bigger than this will always be dropped.

The MSS is specified by each connection during the set-up handshake (*see more: Connection Set-up, page 25*). It may be different for each end of the connection (i.e. larger segments can travel in one direction but not the other).

Window Size

This is the **maximum amount of data that can be processed by a receiver** at once (often informed by a combination of buffer sizes and processing speed).

This size can be **adjusted throughout the connection lifespan** to accept more/less data (achieved through messages sent to the sender).

Segment Sizing Problem

The amount of data that a receiver can accept depends on the rate at which it can process already-received data (i.e. its **window may be partially filled already**).

The sender can only ever be sure that data for which it has received an acknowledgement was actually accepted by the receiver.

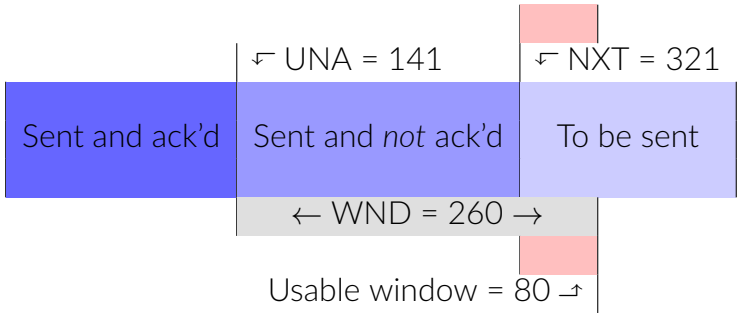
Solution: Usable Window Size

The usable window size is an estimate of the bytes that have **not yet been sent** but the sender believes the **receiver is ready to accept** (it is **computed by the sender**).

To determine it, the sender keeps track of **three variables**:

- UNA: the sequence number of the first byte that has been sent but not yet acknowledged.
- NXT: the sequence number of the next byte to be sent.
- WND: the window size reported by the receiver.

Usable window size: $UNA + WND - NXT$.



Silly Window Syndrome & Nagle's Algorithm

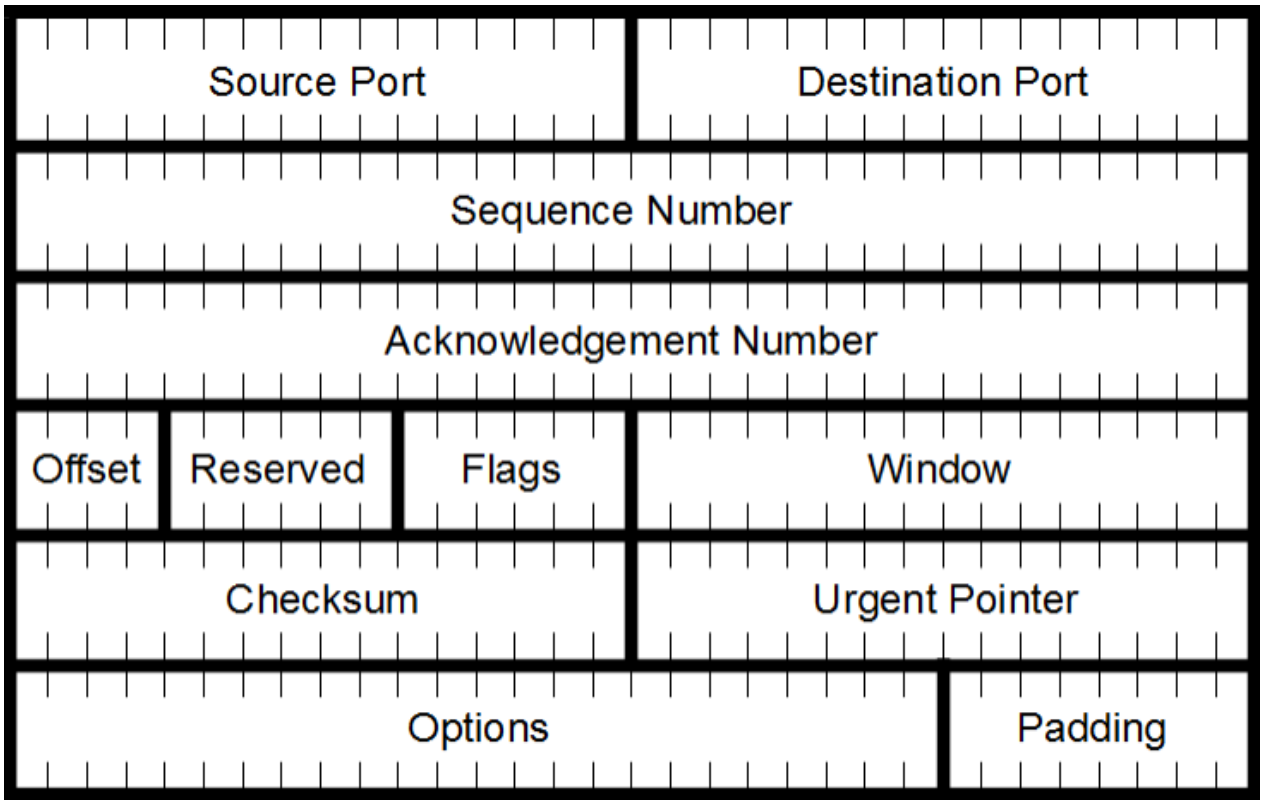
If the receiver adjusts the window size to be **too small**, bandwidth usage becomes **very inefficient**, because lots of very small segments will be sent, and there will be an acknowledgement for each. This is Silly Window Syndrome.

Nagle's algorithm is designed to help the sender and receiver work together to tackle this problem.

- A **sender** does not send more data further data until either...
  - ...all the data that has been sent as been acknowledged, or
  - ...the data to be send reaches the **MSS**.
- As it becomes able to accept more data, a **receiver** doesn't tell the sender about the larger window until either...
  - ...the window reaches the **MSS**, or
  - ...the window reaches half of the recevier's maximum buffer size.

TCP Header

Like the IPv4 header, the TCP header consists of 5 32-bit words, with additional 32-bit words sometimes used to specify options.



- Source/destination ports (16 bits each)
  - Unsigned integers (negative ports don't exist).

  - Source ports are allocated by TCP software.
  - Destination ports are chosen based on the service required.

- Sequence number (32 bits)
  - During the set-up handshake this is the initial sequence number (ISN).
  - In normal messages, this indicated the sequence number of the first byte of the segment.
  - *See more: Segmentation and Acknowledgement, page 26*.
  - *See more: Sequence Numbers, page 27*.

- Acknowledgement number (32 bits)
  - Used in `ACK` messages.
  - *See more: Segmentation and Acknowledgement, page 26*.

- (Data) offset (4 bits)
  - Length of the TCP header in 32-bits (minimum value of 5).
  - Determines how far into the datagram the actual data starts.

- Reserved (6 bits)
  - Reserved for future use.

- Flags (6 bits)
  - Extra information about the message.
  - **`SYN`**: this is a set-up (synchronise) message.
  - **`FIN`**: this is a teardown (finalise) message.
  - **`ACK`**: this is an acknowledgement message.
  - **`PSH`**: historically this meant that all data that's ready to be sent should be sent right away; in practise this is now always set.
  - **`URG`**: indicates that this segment requires immediate action (useful on a slow connection) (example: `Ctrl + C` on a remote shell).
  - **`RST`**: *see more: Reset Flag, page 31*.
  - **Note**: flags can be mixed, for example a `SYN ACK` message.

- Window (16 bits)
  - The receiver's current window size.
  - Send in `ACK` messages.
  - *See more: Window Size, page 28*.

- Checksum (16 bits)
  - Used to validate the integrity of the header **and message**.
  - *See more: Header Checksums, page 31*.

- Urgent pointer (32 bits)
  - Used with the `URG` flag.

  - Points to the first byte **after** the urgent data in this segment.
  - Note: segments may mix urgent and non-urgent data.

- Options and Padding (32-bit words)
  - Optional arguments and flags used by TCP processing software.
  - **Variable number of bits**, but always padded to 32-bit words with zeros.
  - Example use: declaring the maximum segment size (MSS) during the set-up handshake.

Reset Flag

If one host **crashes** or there are **severe transmission problems**, one host that's connected may lose its knowledge of that connection, meaning it **will not be expecting the data** sent from the other host.

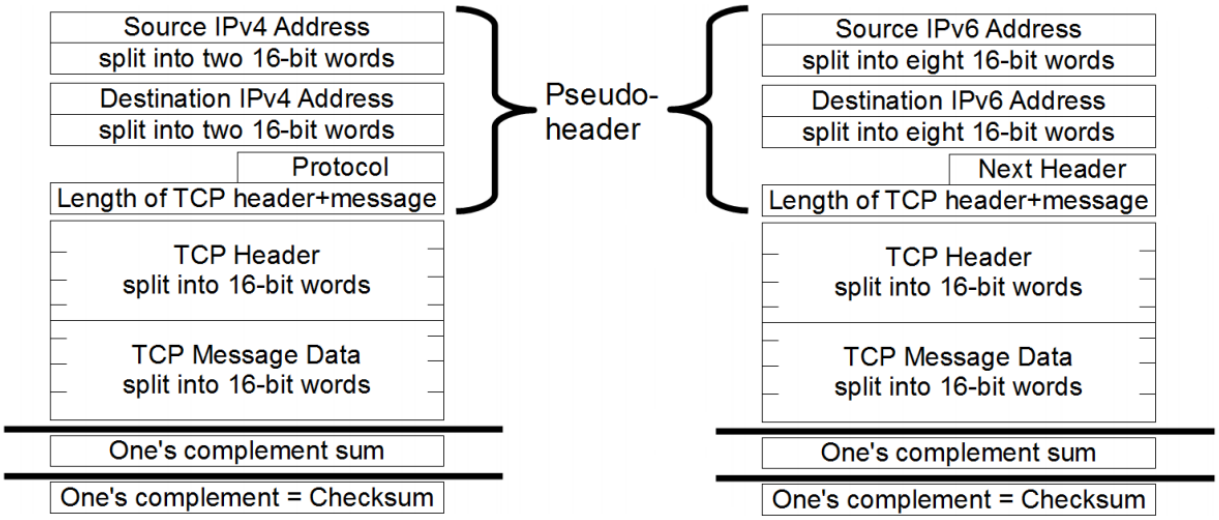
If a host receives unexpected TCP data it responds with a **reset** message to **stop to connection** and resolve the problem by starting the connection again.

The `RST` flag is also sent if a client tries to communicate with a **port that is not open**.

Header Checksums

As with IPv4 (*see more: Header Checksum, page 20*), the TCP header checksum is the one's-compliment of a one's-compliment sum of 16-bit words. The words included are:

- The TCP header (with the checksum set to zero).
- The message data.
- The IP addresses (from the IPv4/6 header).
- The protocol/next header field (from the IPv4/6 header).
- The length of the TCP message (header **and** data).
- The components of the IP header make up a section called the **pseudo-header**, shown on the diagram below for TCP/IPv4 and TCP/IPv6 checksums.



TCP and IP

TCP **runs over** IP: TCP datagrams become the **payload/content** of IP datagrams.

IP deals with addressing hosts and fragmentation to ensure the network can transmit the data.

TCP can more or less ignore what IP does and just pass data to send to its given address.

TCP and IP are **not fully separated**, because the TCP header checksum depends on parts of the IP header (*see more: Header Checksums, page 31*).