5CCS2INS Internet Systems

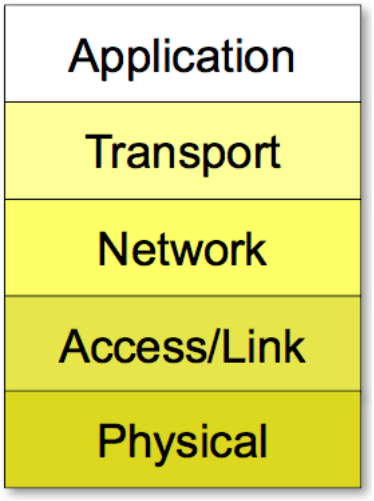
Internet Architecture

- We want to connect, connected networks so they can exchange information from one to the other.
- We want to create a stable network, if a sub-network fails we still want the system to work.

Layers

We can see the Internet Architecture as a stack with different layers.

Protocol Stack:



- **Physical layer** the means by which the data is transferred (copper, fibre, radio).
- **Access layer** knows how to make use of the physical layer to transfer data between devices on the same network (network card).
- **Network layer** defines how to route messages across networks (where IP protocol is).
- **Transport** defines how to provide reliable communication so that data will not be lost or corrupted (TCP, UDP).
- **Application layer** defines how programs request message to be sent across the internet, using encryption, compression.

Protocols

Protocol: **a way in which we communicate.** It defines:

- How to express information of a particular kind.
- How to respond when asked particular questions or given particular commands.
- What form of messages to expect.

Network Layer

Network Layer: defines how to route messages across networks (where IP protocol is).

- Internet Protocol (IP): **for transferring messages between hosts anywhere on the internet.**
- Internet Control Message Protocol (ICMP): **part of and supports IP, used to report errors and other info back to the sender of the IP message (it is sent over IP)**

(Host-to-Host) Transport layer

- Transmission Control Protocol (TCP): **provides reliability measures: acknowledgements, flow control, sessions and multiplexing.**

- User Datagram Protocol (UDP): **allows data exchange across the network with a minimum of overhead. Provides little reliability through checksums (both party encryptions and comparison) unreliable compared to TCP.**

Process / Application layer

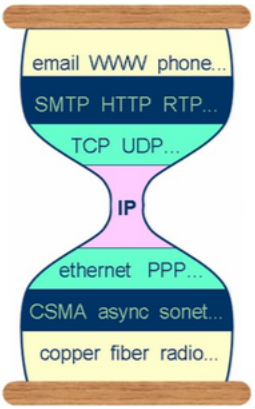
Defines how programs can send messages across the internet, using encryption and compression etc.

- Hypertext Transfer Protocol (HTTP)
- Telnet
- Simple Mail Transfer Protocol (SMTP)
- File Transfer Protocol (FTP)
- Post Office Protocol (POP)
- Domain Name Service (DNS)
- Dynamic Host Configuration Protocol (DHCP)

Edge-Oriented

The success of the internet is due to a edge-oriented approach. The connectionless, packet-forwarding infrastructure ("dumb network") allows a robust structure.

The intelligent edges and the dumb network keep the infrastructure as simple as possible.



- The core network is simple
- New applications can be added without changing the core
- **Glass model:** If the IP breaks everything breaks.

With addresses that are:

- fixed-size numerical quantities, with a simple (net, host) hierarchy
- applied to physical network interfaces, which can therefore be used for both naming a node and for routing to it.

IP Fragmentation

Maximum Transmission Unit (MTU): **the maximum amount of data that a link layer packet can send over the channel that is connecting two hosts.**

If you have larger files than that, then you use fragmentation (break down to sizes that can be sent over the network and are later collected).

MTU	Commonly Used
	Maximum size of link layer
17914	16 Mbit/sec token ring
8166	Token Bus (IEEE 802.4)
4464	4 Mbit/sec token ring (IEEE 802.5)
1500	Ethernet
1500	PPP (typical; can vary widely)
576	X.25 Networks

Networks

Network: **a set of computers connected directly or indirectly.**
Node: **a computer that is part of a network.**
Host: **a node which send or receives messages.**
Routers: **other kinds of nodes.**

Routing and addresses

When one host wants to communicate with another host that is not directly connected data is **routed**, passing through a series of connected nodes from one to another until it reaches its destination. The **Address** is used by the sender to identify the receiver

Internet Protocol (IP)

IP: **Network layer, packet delivery service (host-to-host)**

- Connectionless: each datagram (the format of data that can be recognized by the IP) is independent of all others
- Unreliable: there is no guaranteed delivery

Responsible for:

- Fragmentation / Reassembly of packets (based on MTU)
- Routing

Includes a structure for addressing hosts:

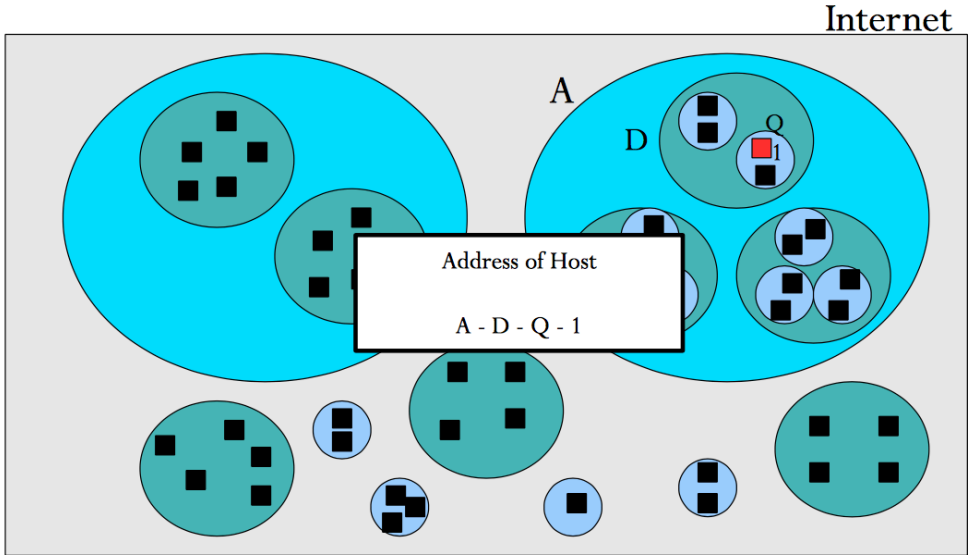
- In the global internet every **host and router** must have a **globally unique IP Address.**
- **IP address technically are associated with an interface** not a host (you can have multiple network cards on your computer with different IPs).

IPv4 vs IPv6

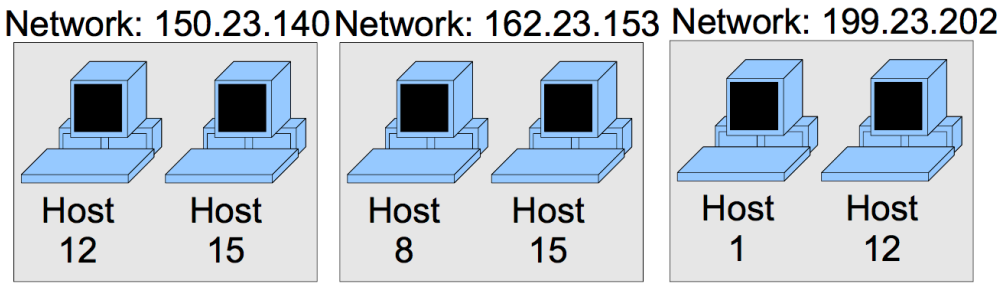
Currently the internet primarily uses version 4 (IPv4), IPv6 compatibility is being deployed slowly. It is needed because:

- Increases Address Space (we're running out of internet addressing)
- Lack of Security at the current Network Layer
- Better quality of Service for end-to-end networking
- Supports new features for applications

Hierarchical Addressing



The internet is divided into networks which have their own **network prefix**, each host has a **host identifier** and together they make the **IP Address**, an IP address typically is 4bytes long or 32-bits (8 * 4).



NOTE: We can have 2 different 15's because they are not in the same network, hence the network address is not the same.

Network Prefix
Global routers pass each message down the local network router for the given network prefix, then the local router passes the message to the host with the given host ID. *Routers need to know how long the prefix of the address is.*

***Global Router --> Local Network Router** (w/ given network prefix) **--> host** (w/ given host ID).*

Host ID

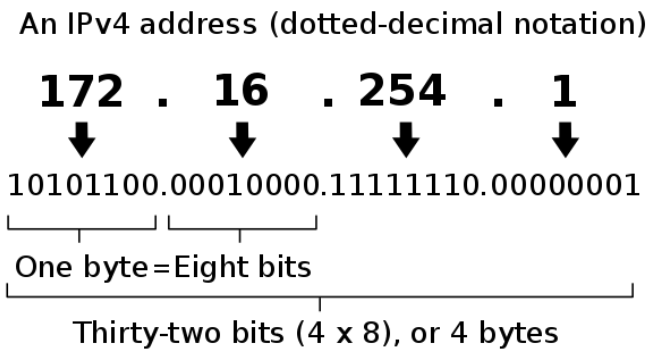
- Host ID is the bits after the network prefix.
- With N bits we can allocate 2^N different addresses.
- In IPv4, **two addresses are reserved** (**network address** and **broadcast address**), so we can only have **$2^N - 2$ hosts.**
- All 0s (network address) and all 1s (broadcast address) which will talk to all hosts.

E.g. A IPv4 *network prefix* is 23 bits long, the *Host ID* is 9 bits. (32 – 23 = 9) TF the network can hold up to 510 hosts ($2^9 - 2 = 510$)

Note TF = Therefore.

$$Hosts\ in\ IPv4 = 2^N - 2$$

n = number of bits for host id (not used by network prefix).



IPv4 has 4 sections, each being 8 bits. Total 32 bits.

Allocating Addresses

There are certain reserved IP locations you can use that will never conflict with other hosts.

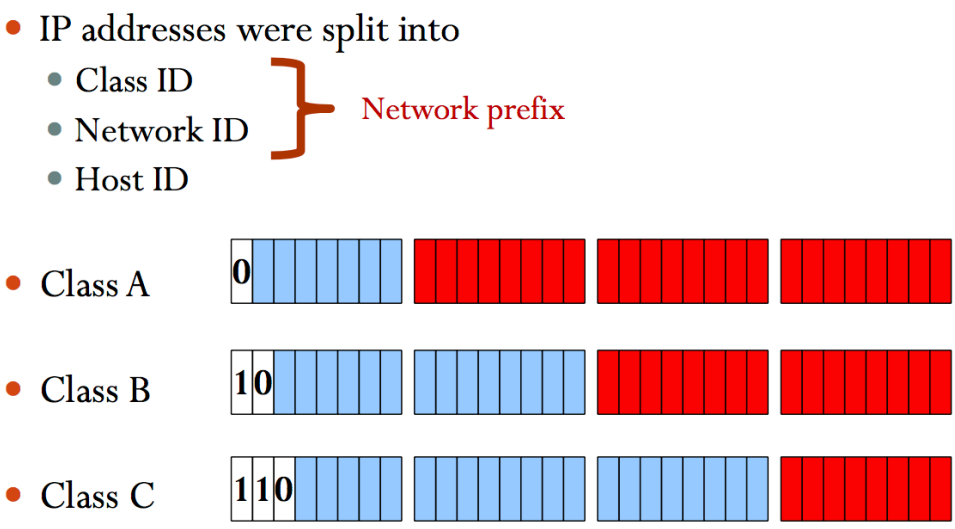
IPv4 Addressing

Its 32 bits long = 4,294,967,296 different addresses

Every IP address consist of two parts, one identifying the network and one identifying the host.

To provide flexibility, classes are introduced into IPv4. They rely on the size of the network you want.

- Classes A to E.



Blue = network prefix. Red = host id.

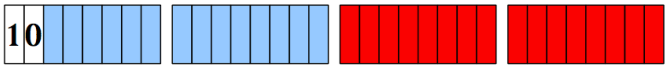
Class A large network

Note that */number* refers to the network defining bits used.

- Class A is for very large networks
  - 8-bit network-prefix with the highest order bit set to 0
  - 24-bit host-number
- A maximum of 126 (2^ 7 -2) /8 networks can be defined.
  - the /8 network 0.0.0.0 is reserved for use as the default route and the /8 network 127.0.0.0 (also written 127/8 or 127.0.0.0/8) has been reserved for the "loopback" function.
- Each /8 supports a maximum of **16,777,214** (2^24 -2) hosts per network.
  - all-0s ("this network") and all-1s ("broadcast") host-numbers may not be assigned to individual hosts.
- Since the /8 address block contains 2^31 individual addresses and theIPv4 address space contains a maximum of 2^32 (4,294,967,296) addresses,
  - the /8 address space is 50% of the total IPv4 unicast address space.
  - range: from 1 to 126

Class B

- Class B is for smaller networks
  - 16-bit network-prefix with the two higher order bits set to "10"
  - 16-bit host-number

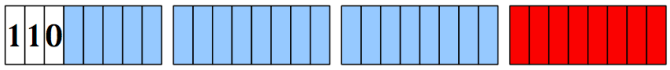


- A maximum of 16,384 (2^ 14) /16 networks can be defined
- Each /16s supports **65,534** (2^16 -2) hosts per network.

- Since the entire /16 address block contains 2^30 (1,073,741,824) addresses,
  - it represents 25% of the total IPv4 unicast address space.
  - Range from 128 to 191

Class C

- Class C is for much smaller networks
  - 24-bit network-prefix with the three higher order bits set to "110"
  - 8-bit host-number



- A maximum of 2,097,152 (2^ 21) /24 networks can be defined.
- Each /24s supports 254 (2 ^8 -2) hosts per network.

- Since the entire /24 address block contains 2^ 29 addresses,
  - it represents 12.5% (or 1/8th) of the total IPv4 unicast address space.
  - Range from192 to 223

Example IP Addresses

- 1.22.11.12 Class A
- 137.22.11.12 Class B
- 201.22.11.12 Class C
- 193.92.96.255 /24 Broadcast address
  - All "1"s at the host part represent the broadcast address.
  - The **broadcast address** is the address used to send a message to all hosts on the network
- 193.92.96.0 /24 Network address
  - All "0"s at the host part represent the network address
  - The **network address** is an IP address for the network as a whole, as used by routers to know where to send the messages

255 is the maximum value, so its all 1s at the host id part so it's a broadcast address.

If the host ID is 0, means that in binary its all zeros, hence it's a network address.

Ranges using classes might be a range of space (Class C 254 but Class B 65,534) so if we need 1000 we will be wasting about 64k addresses since they are blocked by that network.

They introduced the idea that the network prefix size can vary in length. The shape can be /21. This is called (1993 IETF standardised the Classless Inter-domain Routing (CIDR)).

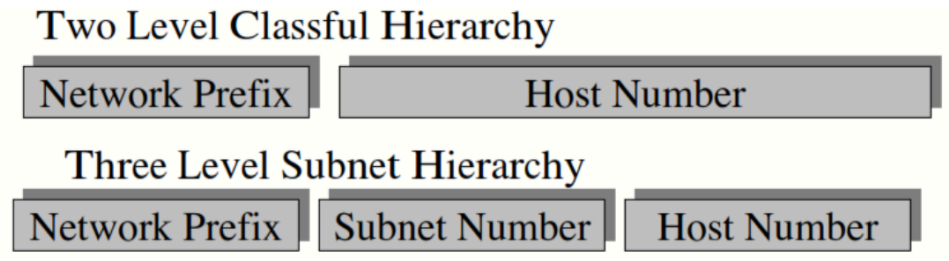
For the example of organisation with 2000 hosts,

- IP addresses of the form a.b.c.d/21 can be allocated (2046 hosts).
- the first 21 bits specify the organisation's network address and are common for all the hosts inside the network,
- The remaining 11 bits can specify a specific host inside this network.
- In a real case scenario the 2000 hosts will be further divided using the last 11 bits in subnets to create different networks inside the organisation

TODAY: address classes are ignored and routers are explicitly told the length of the prefix.

Subnets

Instead of having the network prefix and host number we segment the host number to have 3 level subnet hierarchy.



Network Mask

Used by a router to isolate the network prefix part of the network address to know if the message arrived.

- Default Subnet Masks
  - Class A:

    11111111.00000000.00000000.00000000 255.0.0.0
  - Class B:

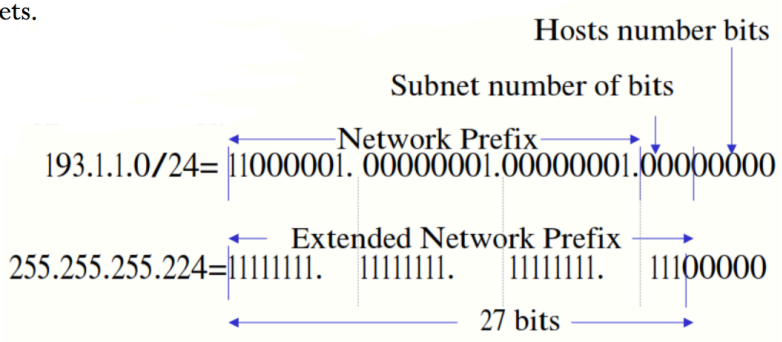
    11111111.11111111. 00000000. 00000000 255.255.0.0
  - Class C:

    11111111.11111111.11111111.00000000 255.255.255.0

Example 1

Organisation has assi Paste the network number 193.1.1.0/24 and wants to have 6 subnets, where the largest one need to support up to 25 hosts.

- The first step is to determine the number of bits required to define the 6 subnets.



We want to look at the number of subnets to create and the number of hosts per subnet is important. Start by the left most and pick the ones that are available.

- **Organisation has assigned the network number 193.1.1.0/24 and wants to have 6 subnets, where the largest one need to support up to 25 hosts.**
- The eight subnets then will be:

Base Net: 11000001.00000001.00000001 .00000000 = 193.1.1.0/24
Subnet #0: 11000001.00000001.00000001. **000**00000 = 193.1.1.0/27
Subnet #1: 11000001.00000001.00000001. **001**00000 = 193.1.1.32/27
Subnet #2: 11000001.00000001.00000001. **010**00000 = 193.1.1.64/27
Subnet #3: 11000001.00000001.00000001. **011**00000 = 193.1.1.96/27
Subnet #4: 11000001.00000001.00000001. **100**00000 = 193.1.1.128/27
Subnet #5: 11000001.00000001.00000001. **101**00000 = 193.1.1.160/27
Subnet #6: 11000001.00000001.00000001. **110**00000 = 193.1.1.192/27
Subnet #7: 11000001.00000001.00000001. **111**00000 = 193.1.1.224/27

These are the sub-networks you have assigned over /27 since we only have 5 bits for the host ID.

We now have 5 bits lefts $2^5 – 2 = 32 - 2 = 30$ which is larger than the 25 we needed.

- **Organisation has assigned the network number 193.1.1.0/24 and wants to have 6 subnets, where the largest one need to support up to 25 hosts.**
- For the host addresses we are using all the 5 bits except all 0 (subnet) and 1's (broadcast)

Subnet #2: 11000001.00000001.00000001.**010**00000 = 193.1.1.64/27
Host #1: 11000001.00000001.00000001.**010**00001 = 193.1.1.65/27
Host #2: 11000001.00000001.00000001.**010**00010 = 193.1.1.66/27
Host #3: 11000001.00000001.00000001.**010**00011 = 193.1.1.67/27
Host #4: 11000001.00000001.00000001.**010**00100 = 193.1.1.68/27
Host #5: 11000001.00000001.00000001.**010**00101 = 193.1.1.69/27
.....
Host #29: 11000001.00000001.00000001.**010**11101 = 193.1.1.93/27
Host #30: 11000001.00000001.00000001.**010** 11110 = 193.1.1.94/27

Defining the Broadcast Address for Each Subnet

- The broadcast address for Subnet #2 is the all 1's host address, or
- 11000001.00000001.00000001.01011111 = 193.1.1.95

Note that the broadcast address for Subnet #2 is exactly one less than the base address for Subnet #3 (193.1.1.96).

- This is always the case – the broadcast address for Subnet#n is one less than the base address for Subnet #(n+1).

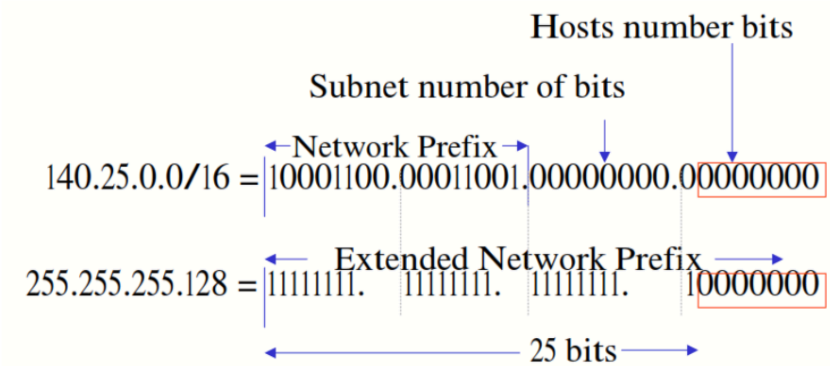
Example 2

- **Organisation has assigned the network number 140.25.0.16/16 and it needs to create a set of subnets that supports up to 60 hosts on each subnet.**

- Step one: number of bits required to define 60 hosts on each subnet is 6 bits
  - 6 bits define 64-2=62 host address
  - in order to take into account future growth we select 7 bits which define 126 addresses per subnet.
- Determine the subnet mask/extended-prefix length:

It didn't tell us how many subnets we need, so we can choose.

Step two: determine the subnet mask/extended-prefix length



- Defining Each of the Subnet Numbers: the 9 bits allocated in the subnet portion of the IP address allows 512 different subnetworks.

Base Net: 10001100.00011001.00000000.00000000 = 140.25.0.0/16
Subnet 0: 10001100.00011001.00000000.00000000 = 140.25.0.0/25
Subnet 1: 10001100.00011001.00000000.10000000 = 140.25.0.128/25
Subnet 2: 10001100.00011001.00000001.00000000 = 140.25.1.0/25
Subnet 3: 10001100.00011001.00000001.10000000 = 140.25.1.128/25
Subnet 4: 10001100.00011001.00000010.00000000 = 140.25.2.0/25
.....
Subnet 511: 10001100.00011001.11111111.10000000 = 140.25.255.128/25

- Defining Hosts Addresses for Each Subnet: The 7 allocated bits will give 126 different host ID.
- For subnet 3 for example we have:

Subnet #3: 10001100.00011001.00000001.10000000 = 140.25.1.128/25
Host 1: 10001100.00011001.00000001.10000001 = 140.25.1.129/25
Host 2: 10001100.00011001.00000001.10000010 = 140.25.1.130/25
Host 3: 10001100.00011001.00000001.10000011 = 140.25.1.131/25
.....
Host 126: 10001100.00011001.00000001.11111110 = 140.25.1.254/25

- Defining the Broadcast Address for Each Subnet:
- For subnet 3 again this address will be the following:
  - 10001100.00011001.00000001.11111111 = 140.25.1.255

Note: if we have to define subnets we start from the left, if we have to define hosts, we start from the right.

Problems with fixed length sub netting
If we have different requirements with the different labs. One lab requires 200 others require 50, this means we have to create 200 for all but we will be wasting 150 addresses for those which use 50.

- A /24 network needs the following five subnets
  - Subnets P, Q, R require 12 hosts each
  - Subnet S requires 36 hosts
  - Subnet T requires 90 hosts

- 254 available addresses in the network ($2^8 - 2 = 254$)
- If we use a 3 bit subnet ID
  - We can accommodate all subnets, as $2^3 = 8$ and we have 5 subnets
  - But each has only 30 hosts, as $8 - 3 = 5$ host ID bits, so $2^5 - 2 = 30$ hosts
- If we use a 1 bit subnet ID
  - Enough hosts per subnet, as $2^7 - 2 = 126$ hosts
  - But only 2 subnets, as $2^1 = 2$

To solve this we can **use variable length sub-netting**.

- You will always pick the largest network (90 hosts). How many bits do I need? 7 bits.

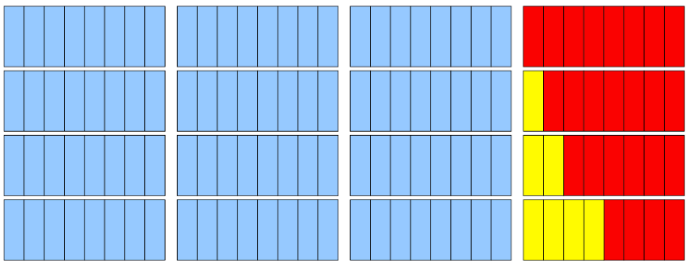
- Using variable subnet ID lengths, we iteratively divide up the host ID space, first into large blocks, then into smaller ones
  - First, to accommodate the largest subnet, T, we use 1 bit to split the T subnet from the rest
  - We then accommodate the second largest subnet, S, by splitting the remainder: S needs 6 bits for host IDs, so use 1 more bit to split S from the rest
  - Finally, we need 2 bits to split subnets P, Q and R

if the **Host Id** starts by 1, then we are referring to subnet T, if not then it's some other subnet. We assign 1 to the network prefix, this means that the subnet mask (*/number*) also does, we have to pay attention to this.

- As the number of bits used for the subnet prefix vary, so the subnet masks vary for each variable length subnet
- Subnet T has addresses **X.X.X.X /25**
  - 24 bits for network prefix + 1 for subnet ID
  - 25 bits on extended network prefix!
- Subnet S has addresses **X.X.X.X /26**
  - 26 bits on extended network prefix!
- Subnets P, Q, R have addresses **X.X.X.X /28**
  - 28 bits on extended network prefix!

Example

- /24 network
  - Subnets P, Q, R require 12 hosts each
  - Subnet S requires 36 hosts
  - Subnet T requires 90 hosts
- Network
  - T : /25
  - S : /26
  - PQR: /28



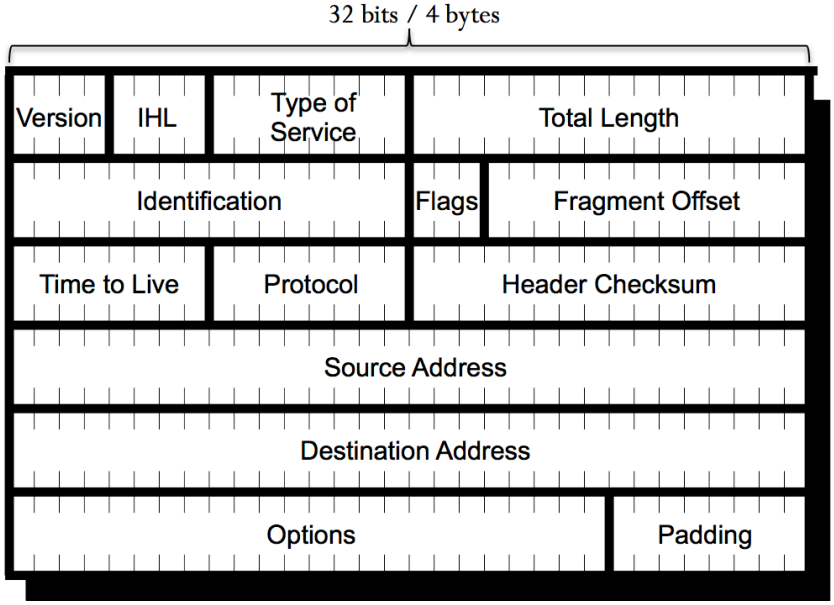
T: 10001001 01001001 00001001 0 . . . . . .
137.73.9.0 /25
S: 10001001 01001001 00001001 10 . . . . . .
137.73.9.128 /26
P: 10001001 01001001 00001001 1100 . . . .
Q: 10001001 01001001 00001001 1101 . . . .
R: 10001001 01001001 00001001 1110 . . . .
137.73.9.192 /28
137.73.9.208 /28
137.73.9.224 /28

A message may:

- Arrive corrupted
- Arrive with fragments
- Arrive multiple times
- Be lost

Each network has an MTU (maximum transmission unit), the sender fragments messages to fit the local MTU.
These packages may be fragmented further when moving from one network to another with smaller MTU.
Packets are then reassembled when reached destination (each fragmentation must be a multiple of 8 bytes [64 bits] except the last one).

IPv4 header



An IP header with no options is 20 bytes long

Version Field

Specifies the IP version, 4 bits, 0100 for IPv4, 0110 for IPv6 (4 or 6 in binary)

Internet Header Length (IHL) field

Indicated the length of the header. 4 bits, minimum value is 5 (to include all header files), header must be multiple of 32 bits (if there is options, padding will correct length.

IHL	Header in bits
5	160
6	192
10	320

Type of service field

Used by message sender to specify how a message should be handled in transmission.

- 8 bits

- Originally used as a set of flags specifying choices of the kind of service required (e.g. higher throughput vs higher reliability)
- Redefined by more recent protocols handling network congestion

Total Length Field

- Total number of bytes in the whole datagram
- 16 bits: 0 -65,535
- Header plus fragment data
- Measured in bytes / octets (multiples of 8)
- Minimum: 20

Identification Field

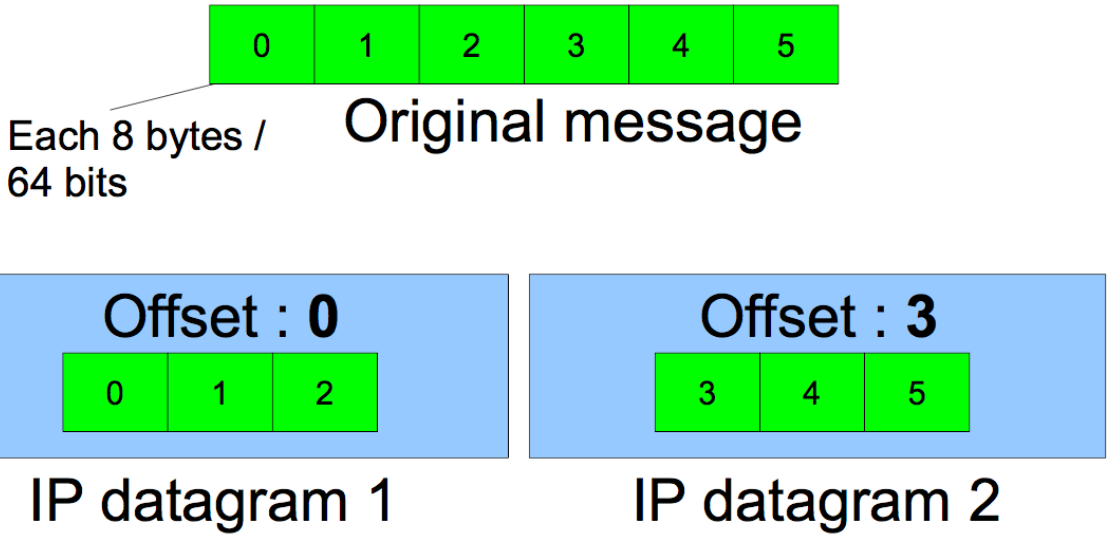
Identifies the whole of fragmented message, 16 bits (0 – 65 535). For a receiving host every fragment with the same identification from the same host is part of the same message. We can reconstruct the message from the fragments (once a message has been delivered, the same id can be used).

Flag field

- An informational bit (0 = no, 1 = yes)
- 3 fields in a IPv4 header (1 bit each, total 3 bits)
- **Reserved:** first bit must always be 0
- **Don't fragment (DF):** this data should not be fragmented during transmission.
- **More fragments (MF):** this fragment is followed by others in the message.

Fragment Offset Field

- The fragment offset (FO) field specifies where in the original message the fragment in this datagram starts
- Used to re-assemble message from fragments in the correct order
- It is a position from the start of the message counted in 8 bytes (64 bits) units, so FO= 3 means this fragment starts after 24 bytes in the original message (8x3 = 24)
- The FO field is 13 bits long, so allows values 0 – 8191



Time to live

- Specifies how long the datagram is allowed to remain in the sustem before being deleted.
- 8 bits: 0 – 255
- When it reaches 0, the datagram should be destroyed.
- Prevents data from cycling around internet in an endless loop when they cannot reach the intended destination.

Protocol Field

- The host-to-host transport layer protocol used by the message sent over IP.
- 8 bits
- Protocols include TCP: 6, UDP: 17, ICMP: 1
- Each has a numeric identifier
- Protocol identifiers are assigned and amnintaned by the Inetrnet Addinged numbers authority.

Header checksum Field

- You put all the header data into 16 bits and then you add one's complement.
- Sender, receiver perform the same check.
- Used to ensure integrity in transmission.

One's complement sum

```
  11111111 11111100
  00000000 00000100
1 00000000 00000000
                  1
  ----------------
  00000000 00000001
```

One's complement

```
00110011 10101010
11001100 01010101
```

Source and destination fields

- IP address of sending/source host
- IP address of destination host (intendent recipient)
- 32 bits each.

10001001 01001001 00001001 11101000

137.73.9.232

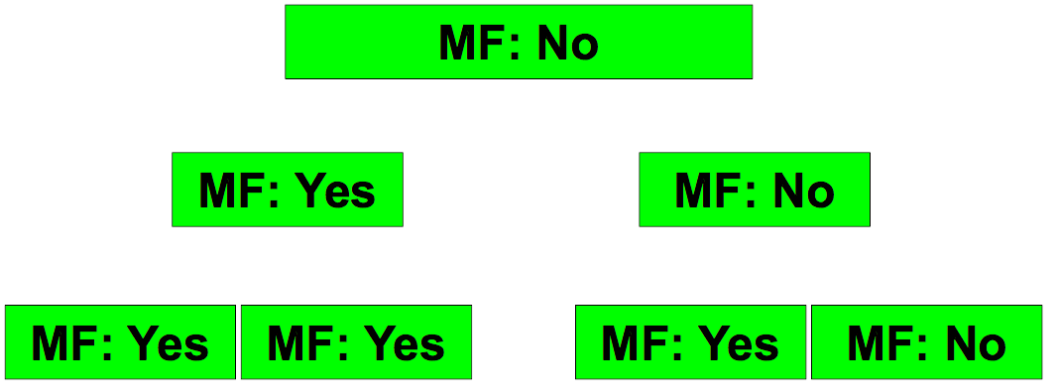
Options and Padding

- Optional arguments usable by IP processing software
- Variable number of bits
- Cover routing options, tracing of route etc.
- Not used much in practice
- See IPv4 specification (RFC 791) for more
- Followed by padding of 0s to make header up to multiple of 32bits

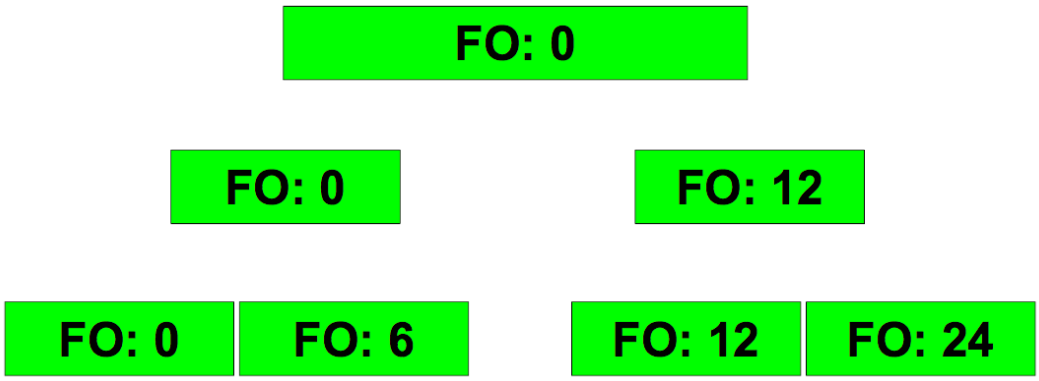
Changes in Fragmentation

- In IPv4, when moving from one network to another with a smaller MTU, datagrams may be re-fragmented
- Fragments must be in octets except the last one (because the FO is counted in 8 byte units)
- Some of the header fields will remain the same in the fragments others will change
- For example, the Total Length field gives the length for the datagram plus the header, so changes In fragmentation.

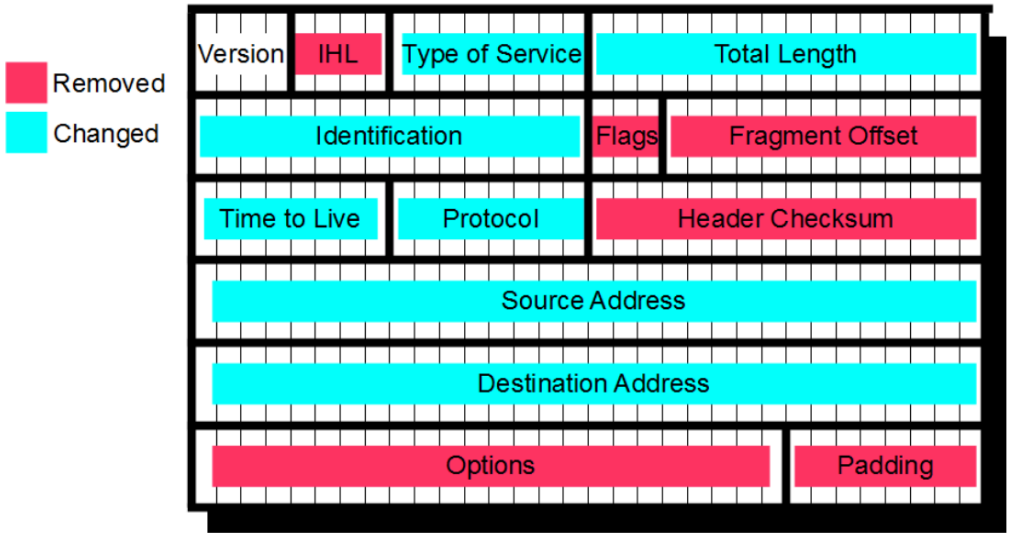
More Fragments flag changes



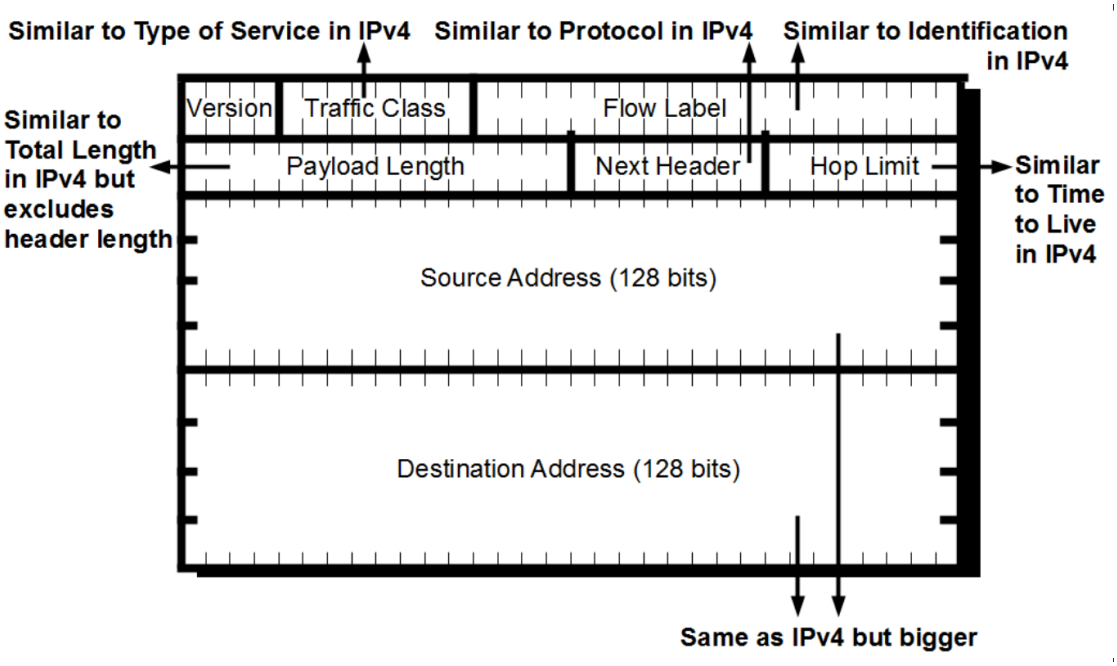
Fragment Offset field Changes



Header Changes from IPv4 to IPv5



IPv6 Header



IPv6 Options

- IPv6 also allows options but not in one block inside header like iPv4.
- An IPv6 can have an extension header with optional data, this can be followed by other extension header to create a chain of options.
- The next header field is sued to specify what the next block of data is an extension header and which kind.

Extension Headers

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 Option

- One example of an IPv6 extension header is the fragment extension header

- If a message is fragmented (by the sender) the receiver has to be able to re-assemble it
- To do this a Fragment Offset field is needed, as in IPv4
- Where a IPv6 packet contains a fragment of a message it will have a fragment extension header containing the fragment offset
- Re-assembly occurs just as in IPv4

MTU Values

- When we configure the router we can set this (by default its set to the maximum).

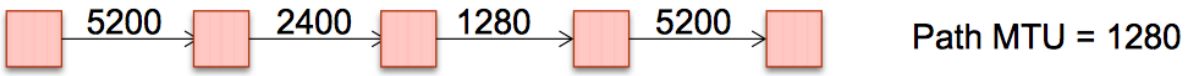
To see the MTUs obtained by IP Software:

- netstat –nr to find the interface (network access protocon instantiation) of rach route.
- netstat –i to get the MTU of an interface.

Path MTU

IPv6 does not use fragmentation in transport, it requires the sender to ensure messages are small enough to cross the networks before sending them

- the Path MTU is the minimum MTU on the path from the sender to receiver



- Path MTU can also be used with IPv4

Path MTU Algorithm

1. Initially assume that the Path MTU is the MTU of the first hop in the path.
2. Fragments message to current assumed Path MTU, sends first fragment.
3. If the fragment reaches the link where it is too large to pass through:
    a. An ICMP Package Too Big messages return to the sender, stating that smaller MTU.
    b. Sender sets its Path MTY to MTU in ICMP message
    c. Try again from step 2.
4. If the first fragment reaches, rest of message I sent, fragmented to known Path MTU.

IPv4 Don´t Fragment

- If the Don´t Fragment Flag is set in an IPv4 message, but the message needs to be fragmented to move onto the network with a lower MTU, an error message is sent by ICMP
- ICMP code 4: fragmentation needed and DF set
- Sent back to the IP message sender

ICMP

- ICMP is part of IP and used to report information on IP processing
- ICMP reports errors and other info regarding IP processing back to the datagram's source
- Examples of ICMP messages include
  - destination unreachable
  - echo request and reply (generated by ping)
  - redirect
  - time exceeded
  - router advertisement and router solicitation

TCP (Transmission Control Protocol)

Operates above IP to offer more features (addresses issues which aren't tackled by IP)

- **Reliability:** data transmitted is checked and re-transmitted when necessary.
- **Multiplexing:** two hosts can have more than 2 conversations without getting confused.
- **Flow / Congestion Control:** A receiving host/network can only accept and process a given amount of data at once, so the sender must control how fast they send data.

Multiplexing

TCP conceptually divides communications a host can be involved with into ports.

- Allows it to multiplex (simultaneous connections).
- Ports are identified by a number, used in each message between hosts so that they know which communication the message belongs to.
- Some ports are reserved for specific application controls.

FTP: 20/21 SMTP (e-mail): 25
HTTP: 80 HTTPS: 443

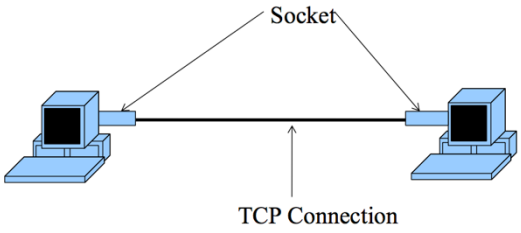
Connections

Server: a host that is ready to accept communication on one or more ports.
Client: a host that communicates with a server (our router or machine).

Users must establish a connection when using TCP to transfer data. This requires a **set-up** and a **teardown** of the connection. A TCP connection is a type of session (there are many kinds of sessions).

Sockets

Socket: a combination of a host's IP address and a port number.



137.73.9.232:8080

host ip : port number

Every communication in TCP happens between two sockets (i.e. two hosts using particular ports)

Connection Setup

Handshake: a sequence of messages sent between the client and server to set up the connection before data is sent.

1. Client sends a synchronise message to the server.
2. Server sends a message back acknowledging the synchronise, and giving permission for communication to take place.
3. Client sends a message acknowledging the acknowledgement.
4. Then they can start sending data.

"Hi can we talk? Yes let's talk. Great I received your approval."

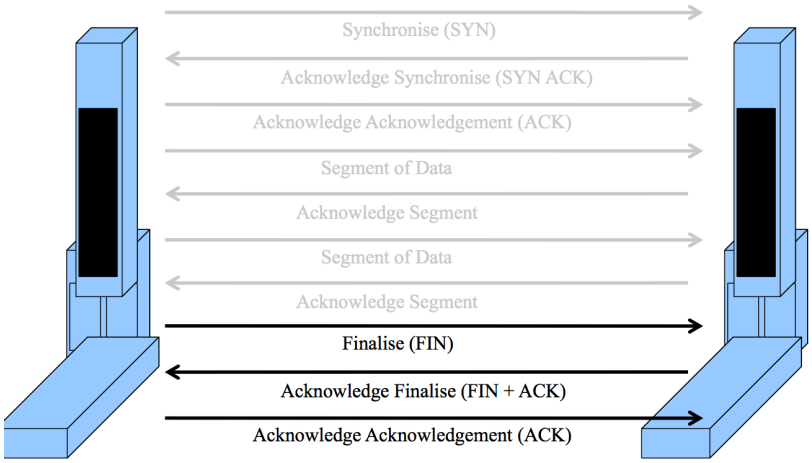
Connection tear-down

A handshake is also used to close the connection.

1. Client sends a finalized message.
2. Server sends back an acknowledgement message.
3. The client responds with an acknowledgment with an acknowledgement.

Same as connection setup but with a ***finalizing*** *message* instead of **synchronize** message.

NOTE LEARN ALL THE NEXT DIAGRAM.



Flow Control

Since a host can only receive and process a certain amount of data (this means that receiver would have to either overwrite unprocessed data or ignore new data), TCP tells the client to slow down or speed up the rate at which data is sent.

Segmentation

The messages are split and transmitted in segments. This is like a fragmentation in IP but it's different:

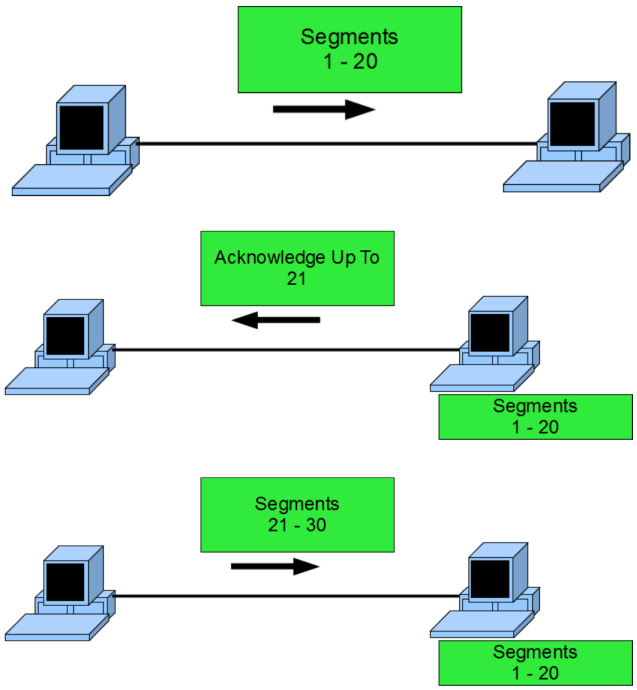
- In IP these limits are used to not exceed the limits of the physical networks and network access layer protocols.
- In TCP, segments are used to not exceed the limits of the receiving host, flow control and to aid reliability of communication.

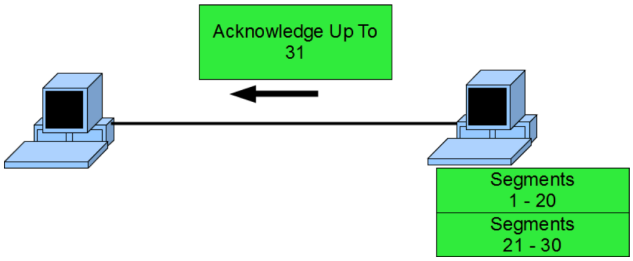
Sequence numbers

Every byte in a message sent from a client to a server has a sequence number. For every given connection and host there is an **Initial Sequence Number** for bytes it sends, this is sent in synchronisation and is incremented by 1 after every segment.

Reliability

An acknowledgement is sent back to the client for every segment the hosts receives, this means that we know if a segment has arrived or not, if not, it will try to re-send it.





If the server acknowledges with a number which is not correct (not 21 in the first time) then we know something was lost.

Even if he receives packages 1 to 20 (return 21) and 50 to 60 (return 21), it will still return 21 because it has not received the bytes in the middle.

Resending Strategies

Time-out: a client may resend a fragment if it has not received an acknowledgment for it in a certain time.

Repetition: a client will resend segment X if it receives acknowledgments suggesting other segments are being received but not X.

> If the server receives segments
> (1-20), (30-40), (41-44), (45-50)
> then it will send an acknowledgement with sequence number 21 for each segment received: 4 times acknowledgement up to 21
> The repetition suggests to the client that the segment starting 21 is lost and needs resending

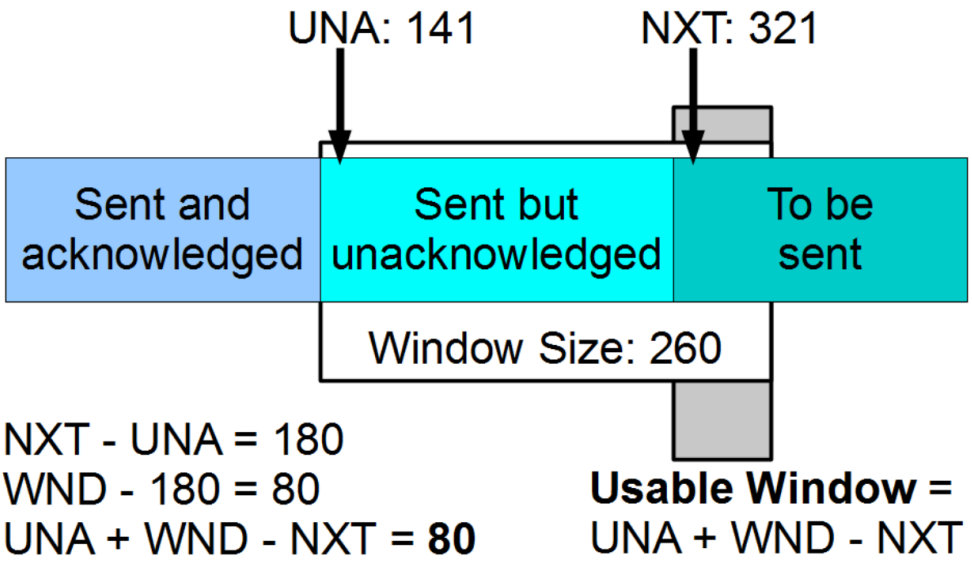
Maximum Segment Size (MSS) is the maximum amount of data allowed in one segment transmitted over that connection. (Specified by server to client during synchronisation).

Window size: the amount of data the server can receive. This can be adjusted during communication if it can accept more or less data. The client can send more than one segment at the same time independently from the segment size.

Segment Size Algorithm

UNA: sequence number of the first byte sent but not yet acknowledged.
NXT: sequence number of the next byte to be sent.
WND: window size.
Usable window: the range of bytes that have not yet been sent, but the client believes the server is ready to receive.

$$Usable\ Window\ Size\ =\ UNA\ +\ WND\ -\ NXT$$



Nagle's algorithm

Silly window syndrome: If server adjusts window size to be too small, bandwidth usage becomes inefficient because very small segments are being sent for acknowledgement. We handle it with the algorithm.

Client: doesn't send any further data until all data is acknowledged or the data that is available to be sent is the same as the maximum segment size.

Server: only informs the client about the window size (larger window size) until the new, larger window size is at least the **maximum segment size** or **half the server's maximum buffer size**.

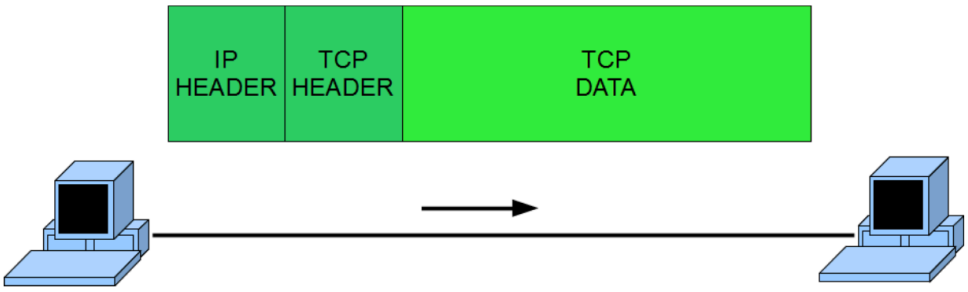
Push and Urgent

Sometimes TCP makes things wait, when we include a **push flag** then it will send it, it will not wait until the package is full. Moreover, if the server sees an **urgent flag** it will process it with priority. (These two flags are commonly used together: like an ambulance entering a hospital).

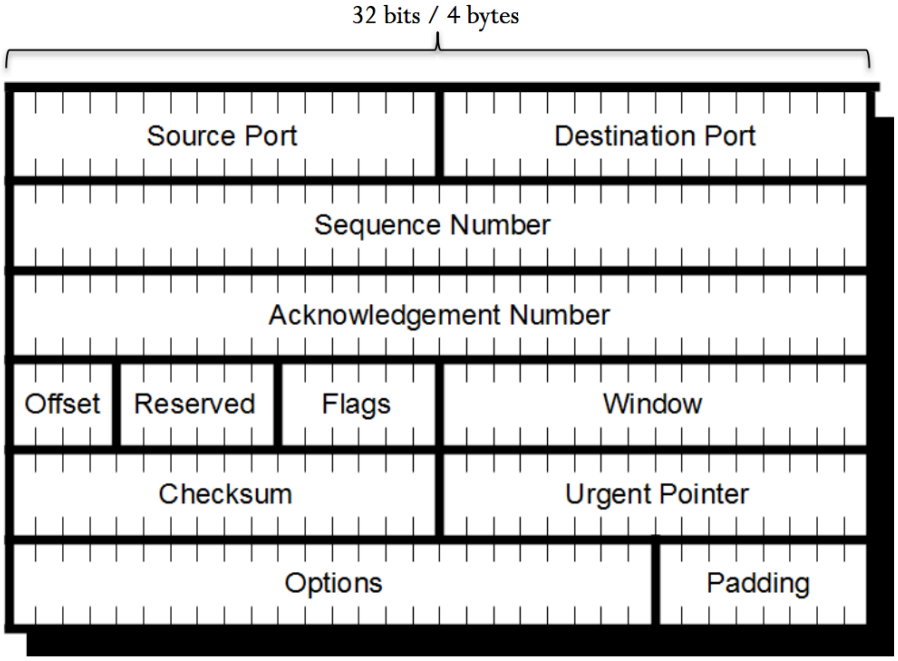
Reset

If a host or client crashes, the server will time-out and start a connection with another machine. If the client comes back up and sends the information as normal, the server will not understand this (unexpected TCP data). What it will do is it will respond the client with a **reset** message, a reset message will also be sent if the port is not free.

Combining protocols (idea of the internet).



TCP Header



Source port and destination port: ports on the client and server side respectively (16 bits each and combine with IP header to make socket address.)

IPv4: 137.73.9.232, TCP Port Address: 8080, 137.73.9.232:8080

Sequence number field (used by sender or client): will be the initial sequence number when **setting up the connection** or **the sequence number at the start of the segment** (it can be any number agreed between the client and the server).

ONLY used when the client sends data. Never when receiving

Acknowledgement Number (used by server): number for which the server has all the data in sequence.

ONLY used when server sends data, never when receiving.

Data offset: the length of the TCP header (depending if it has options or not). 4 bits, minimum value is 5.

Reserved field: 6 bits always set to 0, created for future use and have no purpose yet.

Flags

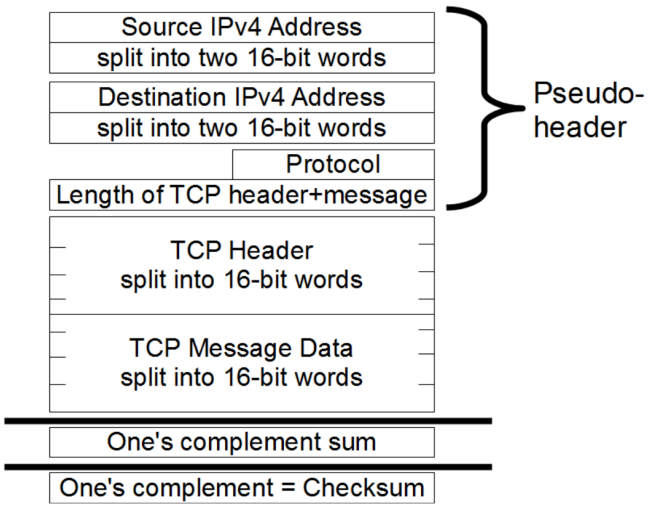
- **Urgent flag:** set flag to one.
- **Acknowledgement flag:** marks if this is an acknowledgement.
- **Push flag:** to tell client this has to be pushed to the network without waiting.
- **Reset flag:** to mark that the message is a reset message.
- **Synchronise flag:** marks that this is a synchronise message or acknowledgement of a synchronise.
    - The first two messages of the establishment of the connection the flag will have the value of 1 (I want to connect, I accept connection).
- **Finalise flag:** marks that this is a finalisation flag
    - The first two messages in the finalisation will have the flag on (not the awk awk finalise)

Window size: 16 bits (0 – 65k), it is used in the synchronisation and any acknowledgment to set the window size (uses this field to tell the client). This is only updated in acknowledgement messages and depends on the capacity of the server.

Checksum: a checksum over the segment, used to check for corruption (16 bits). Like IPv4 but it contains more data so it is more reliable.

> It includes: TCP header, message data, IP addresses, next header field and the length of the TCP message (header + data) and then flip 0 and 1's.

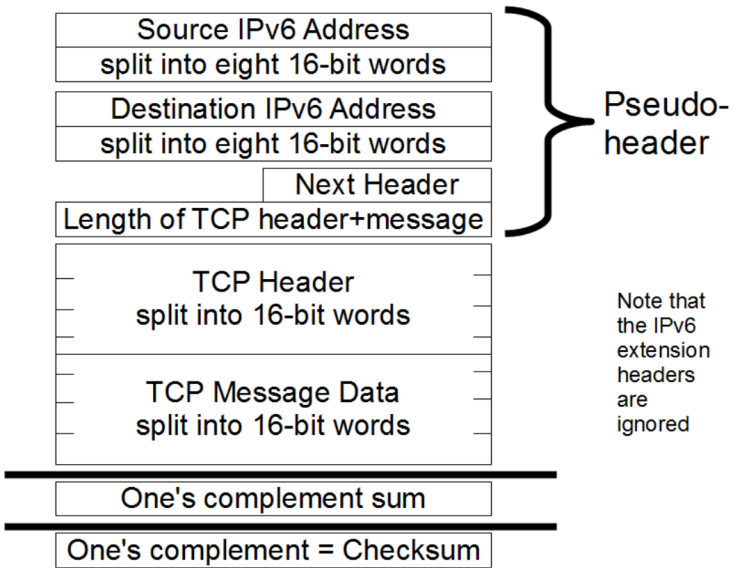
Checksum with IPv4



The use of the pseudo-header somehow violates the layering principles of the internet, since TCP should not carry information about a layer underneath it (IP). It assumes an implementation

of the internetworking layer which may not be true in the future (IP may be replaced with another protocol).

Checksum with IPv6



Urgent Pointer field: **position where the urgent data ends inside the segment.** Not all the data must be urgent. (16 bit), used when URG flag is set to 1. ***It indicates the firs byte of the urgent data.***

Options: varies in length, not useful although it can help to specify the maximum segment size.

Hypertext Transfer Protocol (HTTP)

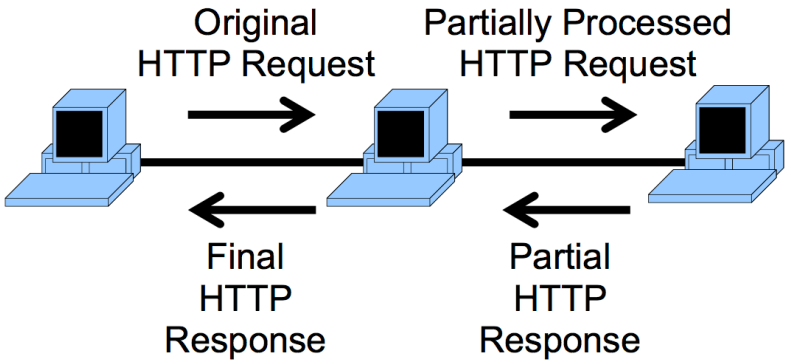
Commonly sent over TCP/IP to send and receive messages. In HTTP a client sends a request to the server and the server returns a response.

- Clients are normally web browsers.
- The client application sending a request is called the **user agent**.
- A host acting as an HTTP server is called a **web server.**

Proxy

Client knows it is using proxy to talk to server, it may transform messages en-route.

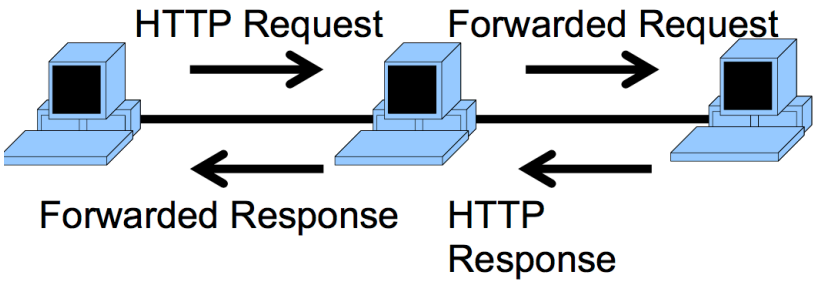
- ***An intermediate node that sits between the destination that satisfies a requirement necessary to reach to the final server.***



Gateway

Connection between a private local network and the internet.

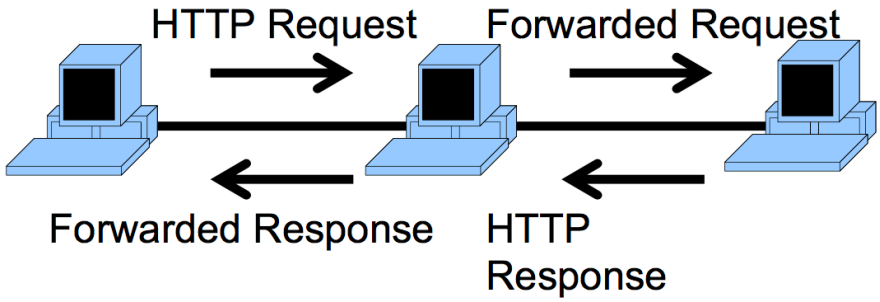
- Gateway only passes data, doesn't change it.



Tunnel

Working remotely on a machine, through a network.

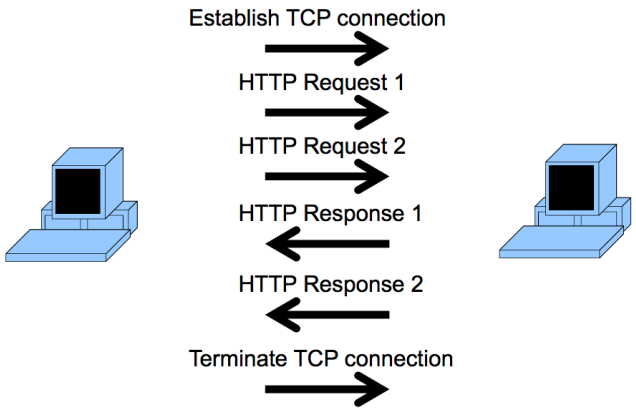
- Client talks to server as If tunnel wasn't there, the tunnel only forwards messages.



Pipelining

If a client has so many requests to a server, the server can pipeline them. This means that the client establishes a single TCP connection and sends its requests without waiting for a response.

- A client ends a pipeline with a "connection: close" command.



Resources

When we use HTTP we commonly are looking for a service: Web page, Database service, Downloadable files, Web services.

A **resource** is identified by a **Uniform Resource Identifier (URI)**

- *Not to confuse with URL (URL is a subtype)*

URI

URI is composed of two parts:

Uniform Resource Name (URN): exists as a unique identifier over the whole web. It should mean something (like an ISBN of a book).

Uniform Resource Locator (URL): what we type in our web browsers.

Syntax

<scheme>:<scheme-specific-part>

The scheme can be HTTP or FTP and the scheme-specific-part has to comply with it.

http://host[":"port]["/"path["? "query]]

http: – scheme
host – website
port – the TCP port used (normally not seen because it's put by default)
path – identifies the resource within the host
query – an optional query to send the resource

Domain Name System (DNS): translates human readable domain names (websites) to IP addresses allow us to access the resources. (Like a telephone directory, you know name but not their phone no).

What can we use in URI

In the scheme specific part we can only use:

A-Z a-z 0-9 $ - _ . + ! * ' ()

Other characters must be encoded as a % followed by the hexadecimal ASCII code.

- Space is %2-
- @ is %40

Http only supports ASCII code, everything else has to be encoded (in real life, the browser will encode this for you).

Internationalized Resource Identifiers were used to allow non-English URIs.

Examples

http://www.inf.kcl.ac.uk/cgi- bin/people.cgi?email=simon&search=1

file://C%3A/Temp/myfile.txt

ftp://myusername:mypassword@www.ctan.o rg/pub

Relative Reference

So, starting from:

http://www.inf.kcl.ac.uk/staff/simonm/

publications

can be a relative subdirectory:

http://www.inf.kcl.ac.uk/staff/simonm/publications/

while:

..

refers to the parent directory (as in UNIX/DOS):

http://www.inf.kcl.ac.uk/staff/

URI Templates

A URI template is a compact sequence of characters for describing a range of URIs

http://www.inf.kcl.ac.uk/staff/{username}/

We can also set a pattern saying that all pages must have an FAQ:

{+ basename}/faq.html

Note: + symbol indicates that reserved characters are allowed (unsafe).

HTTP Methods

A method specifies what's the task you want to achieve with a query.

Common methods:

GET – when you try to download a resource (download a webpage browser).
POST – submits some resource to server (submitting a form on a webpage).
HEAD – used for testing purposes (only concerned with the head of the http response).

Metadata comes in the header. It says to the server: *I don't care about the data just now, I just want to check if the head is correct.*

Other methods:

OPTIONS – requests the server to send available communication options.
PUT – requests the storage of a resource.
DELETE – requests the resource be deleted.
TRACE – requests that the request itself be returned by the server (used for diagnostics).
CONNECT – used for tunnelling http requests through a secure proxy (switching a proxy host to be a tunnel).

HTTP request format

```
GET /index.html HTTP/1.1                    Request Line
Connection: close                           General Header Line
Host: www.example.net                       Request Header Lines
Accept: text/html, text/plain
User-Agent: Mozilla/4.0 (compatible; MSTE 6.0)
```

Note: you will only send a http request with connection: close if there is only one request.

- **Request line**: method, resource, HTTP version
- **General headers**: Data about the message transmission or protocol
- **Request headers**: Parameters of request

Example:

- If you type into a web browser:
  http://example.net:8080/dir/file.html
- The HTTP request will contain:
  GET /dir/file.html HTTP/1.1
  Host: example.net:8080
- The response message body will then contain the contents of **file.html**
- Default TCP port (if none given) is 80

It is important for a server to know if a request is safe or not.

- GET, HEAD, OPTIONS and TRACE are **safe methods** because they don't try to change any data in the server
- POST or PUT place information and upload things so these are not safe.

Idempotent methods: *methods which will have the same effect whether done once or many times.*

- *GET, HEAD, PUT, DELETE (one we delete a file, if we ask to delete it 100 times nothing will be deleted), OPTIONS and TRACE are idempotent (POST is not).*
- *With POST if you use it more than once on a shopping site you will purchase the product multiple times.*

Idempotent sequences: *a sequence of requests can be non-idempotent even though the methods are.*

GET X	->	Client set X = "A"
PUT X = "B"	->	Server sets resource X to "B"
GET X	->	Client receives "B"

A client should never pipeline non-idempotent methods and sequences, if connection breaks, the client may not know where the sequence reached and may need to repeat requests.

HTTP response format

```
HTTP/1.1 200 OK                                    Status Line
Date: Wed, 01 Aug 2007 11:13:09 GMT
Connection: close                                  General Header Lines
Server: Apache/1.3.31
Accept-Ranges: bytes                               Response Header Lines
Content-Type: text/html, text/plain
Content-Length: 942                                Entity Header Lines

<!DOCTYPE HTML PUBLIC = "-//W3C//DTD HTML 4.01
  Transitional//EN">
<html>
<head>                                             Entity Body
<title>Dr. Simon Miles</title>
...
</html>
```

- **Status line:** HTTP version, status code, reason phrase
- **General headers**: Data about the message transmission or protocol (exact timestamp).
- **Response headers**: Parameters of the response (server metadata)
- **Entity headers**: Description of the request's data (content type)
- **Entity body**: The data to be transferred (the resource (if any))

Connection can close because client requested it or because the server is not interested in maintaining it open.

Status Codes

Status code: a 3 digit number which tells us if the request was successful or not accompanied by a human-readable phrase.

Code	Category	Meaning
1YY	Information	General; no indication of success/failure of request
2YY	Success	Request understood and accepted by server
3YY	Redirection	Further action to complete
4YY	Client Error	Invalid request
5YY	Server Error	Unable to complete request

200: OK

300: Redirection, further interaction required

301: Resource moved permanently, the URI has changed and the client needs to request using the new identification

400: Bad request, syntax error by client

401: Unauthorised, request must include authentication

403: Forbidden, the request will not be authorised

404: Not found

500: Internal server error

505: HTTP version not supported

Multipurpose Internet Mail Extensions (MIME)

MIME provides explicit data types, ***it allows non ASCII code to be encoded and transferred over ASCII protocols***.

MIME provides a high level classification of data into 7 media types:

- text
- image
- audio
- video
- application
- multipart
- message

New types can be created but must start with "X-" to mark them as experimental

Media can also have subtypes, such as:

Full media type: **type/subtype**

- text/plain
- text/enriched
- text/html
- text/css

We can also add ***optional parameters*** to give more detail

type/subtype;options

E.g. text/plain;charset=ISO-8859-1

Content Type Declaration

In HTTP, email and other headers, the MIME type content is declared as:

Content-type: type/subtype;ooptions

E.g. Content-Type: text/plain; charset=ISO-8859-1

In the head of the HTML file in the Response Header Lines it will specify which content types it will accept.

MIME encoding

Sometimes we have to encode data to make it transferable, we have to declare what encoding was used to be able to decode it.

MIME data can be encoded in 5 ways:

- **7 bit:** ASCII compatible, fits in lines of 1000 characters
- **8 bit:** 8-bit text character encoding, fits in lines of 1000 characters
- **Binary:** Encoded directly in binary form (cannot be used for direct transmission in SMTP)
  Note that in 7bit, 8bit and binary, data is not changed to transmit.
- **Quoted-printable:** Non-standard ASCII text
- **Base64:** Binary data encoded as ASCII
  - you split every 24 bits to 6 bits, transforming it into a letter.
  - Where data does not divide into 24 bit chunks, = is used to encode missing chunks.

0	A	8	I	16	Q	24	Y	32	g	40	o	48	w	56	4
1	B	9	J	17	R	25	Z	33	h	41	p	49	x	57	5
2	C	10	K	18	S	26	a	34	i	42	q	50	y	58	6
3	D	11	L	19	T	27	b	35	j	43	r	51	z	59	7
4	E	12	M	20	U	28	c	36	k	44	s	52	0	60	8
5	F	13	N	21	V	29	d	37	l	45	t	53	1	61	9
6	G	14	O	22	W	30	e	38	m	46	u	54	2	62	+
7	H	15	P	23	X	31	f	39	n	47	v	55	3	63	/

- Encode the following bytes in base64:

  237, 162
- Convert the values to binary,

  11101101 10100010
- Split into 6-bit chunks (padding with 0s):

  111011 011010 001000 =
- Lookup values in table:

  59 -> 7 26 -> a 8 -> I
- Final encoding:

  7aI=

- Note a = sign takes the form of 6 zero's for padding aka I'm missing 6 bytes.
- We always need 24 bits, therefore if we have 12, we will obtain the letter representation, say: Sg==

Encoding declaration

Content-Transfer-Encoding:<encoding>

E.g. Content-Transfer-Encoding:7bit

MIME composite type

We can state that we might send different types of data using: **Content-Type: multipart/mixed;boundary=boundary-1**

- **"—" are used to start each part and end the list of parts.**
- **Found in the header of the html**
- **Added by the browser in the background.**

- Example: text and a GIF separated by a boundary named boundary-1

Content-Type: multipart/mixed; boundary=boundary-1

--boundary-1
Content-Type: text/plain; charset=US-ASCII
...
--boundary-1
Content-Type: image/gif
...
--boundary-1--

Web Services

Web Services are services deployed using internet and web technology

Service Oriented Computing

Applies principles behind internet and web to software functionality:

- Decentralised
- Distributed Administration
- Protocols remain constant even if servers change
- Allows companies to provide, maintain and update proprietary software on their own sites

Multiple services can be combined to provide a greater product

Extension of object / Component Principles

Interfaces

Each service states the protocol it supports

The protocols can be specific to its business function

The description of the protocols supported by a service is called its interface

Multiple services providing interchangeable functionality can use the same interface defintion

SOAP: XML based communication protocol

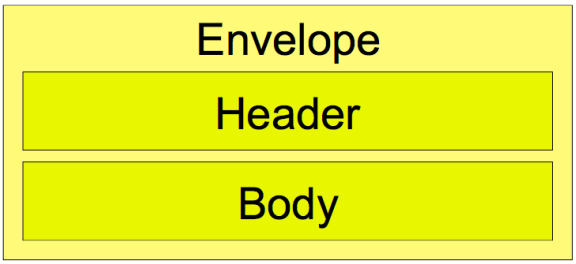
WSDL: XML based-interface definition language

Simple Object Access Protocol

Web services communication protocol, has a common structure. Has an XML element hierarchy.

Structure:

- Body contains a message which is requested by the interface (it's an xml document)
- Header contains information about the communication, message body, sender etc.
    - Authentication, authorisation information
    - Transaction context
    - Resource to where the message is addressed.



Example:

```
<soap:Envelope xmlns:soap =
     "http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <t:Transaction xmlns:t="http://www.example.com"
                   soap:mustUnderstand = "1">
      5
    </t:Transaction>
  </soap:Header>
  <soap:Body>
    <m:GetLastTradePrice xmlns:m =
                "http://example.com/stockquote">
      <m:symbol>DEF</m:symbol>
    </m:GetLastTradePrice>
  </soap:Body>
</soap:Envelope>
```

Soap over HTTP

The SOAP envelope is the entity body in the HTTP request/response

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: application/soap; charset=utf-8

<soap:Envelope xmlns:soap =
    "http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
       ...
```

SOAP actions

SOAP allows the "intent" of the SOAP message to be specified as a URI in the HTTP Content-Type field

It is extra info about how to process the message

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: application/soap; charset=utf-8;
  SOAPAction="http://www.example.com/
  GetLastTradePrice"

<soap:Envelope xmlns:soap =
    "http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
      ...
```

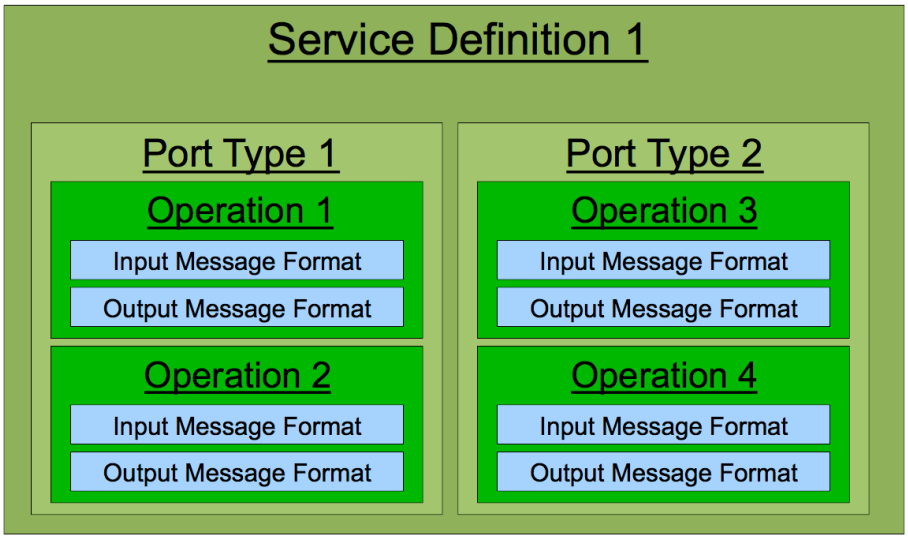
Web Services Definition Language (WSDL)

XML language that specifies the form of messages a service understands or produces. Sets the rules of the SOAP body.

A service interface is split into **port types**, each one has a set of input and output message definitions.

- A calculator might be a port type
- An interest calculator will be another (although it uses the calculator)

An **operations** combines an input message and output message (this input creates this output).



Port Types

Port types are like interfaces in OOP languages

A port type groups a set of operations of a particular kind

A port specifies which messages can be received and produced by one port

Operations

An operations is something that can be performed on a service like a method on OOP

An operation consists of an input request message and an output response message

Web service operations are assumed to be asynchronous, so the response may be received any time after the request

```
<portType name="StockQuotePortType">
 <operation name="GetLastTradePrice">
  <input
      message="tns:GetLastTradePriceInput"/>
  <output
     message="tns:GetLastTradePriceOutput"/>
 </operation>
</portType>
```

Messages

These are XML documents (contents in SOAP bodies). They **must** conform to the schema so that the client can know the expected structure of the request. One message can respond multiple operations.

ListAllEntries request returns ServiceList response
SearchByName request also returns ServiceList response

```
<schema>
 <element name = "PriceRequest">
  <complexType>
   <all>
    <element name="symbol" type=”string"/>
   </all>
  </complexType>
 </element>
</schema>

<message name = "GetLastTradePriceInput">
 <part name="body" element="PriceRequest"/>
</message>
```

A whole WSDL interface document consist of multiple messages, port types and operations. The interface can be shared.

- A company can serve the same functionality with different qualities.

The root node (WSDL)

```
<definitions name = "StockQuote”
  targetNamespace = "http://example.com/stockquote"
  xmlns = http://schemas.xmlsoap.org/wsdl>
…
…
…
</definitions>
```

Implementation WSDL

WSDL is also used to give details of how to use an abstract interface with a given service

Implementation details include

- URL of the service's web server
- Underlying protocol to use (HTTP)

While both the abstract definition and specific implementation details can be in one file they are often split into two so abstract interface can be re-used

Bindings

A binding describes a concrete binding of a port type component and associated operations to a particular concrete message format and transmission protocol

One may specify the use of SOAP while another the use of JAVA RMI

Within a binding, further transport and encoding info is provided for each message of each operation of the port type

You normally have an interface document and then instances that implements the interface. They *import* them.

Bindings

Binding: describes a concrete binding of a port type component and associated operations to a particular concrete message format

```
<binding name="StockQuoteSoapBinding"
         type="tns:StockQuotePortType">
  <wsoap:binding style="document"
       transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
      <wsoap:operation
       soapAction="http://example.com/GetLastTradePrice"/>
      <input>
            <wsoap:body use="literal"/>
      </input>
      <output>
            <wsoap:body use="literal"/>
      </output>
  </operation>
</binding>
```

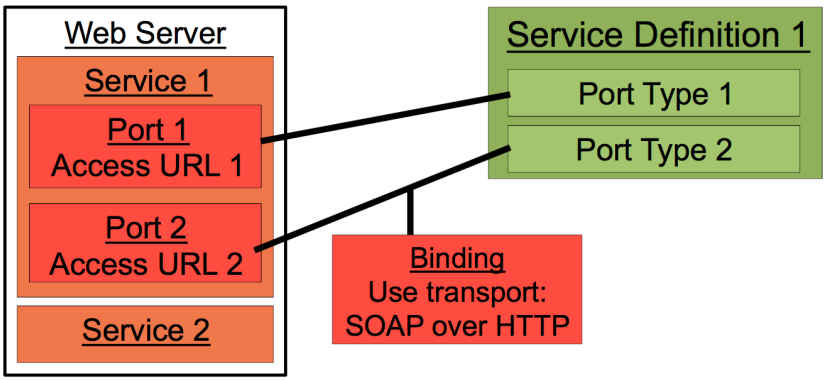
Port

It is a channel of communication to which messages can be sent and received from. Each port has a URL and a binding.

Services

A service is a collection of ports (each port is relevant to each port type, it's an instance). It ties all together the parts.

```
<service name = "StockQuoteService">
 <port name = "StockQuotePort"
       binding = "tns:StockQuoteBinding">
  <soap:address location="http://x.com/sq"/>
 </port>
</service>
```



Universal Description Discovery and Integration (UDDI)

To use the web services, you have to find them, UDDI is a directory in which people can look into it to use it.

Mark-up Languages

We add annotations around certain pieces to explicitly indicate the properties of that piece. It creates a hierarchy of annotations.

We might express a chapter titles as: <chapter><title>Chapter 32</title> …

XML

eXtensible Markup Language (XML): a general purpose mark-up language. An HTML doc can be a specific kind of XML. It uses opening <open> and closing </open> tags and the combination creates an **element**.

Empty elements can be abbreviated with <NAME/> instead of <NAME></NAME> (drawing a line in the middle of the page).

Attributes

Elements can have parameters called **attributes.** These consist of a name and a value in quotes. It specifies info about the element it annotates (put inside the opening tag name).

```
<chapter    columns = "two-column"
            author = "Samhar Mahmoud">
   ...
</chapter>
```

Special attributes: **xmlns** namespace declaration and **xml:lang** the language in which the text is written.

The xml document has a single element, the root of the hierarchy. After this there should be two special tags:

- XML Declaration: <?xml...?> it has a set of attributes that specify the version, encoding and standalone (can a document be parsed alone or does it need other documents to be parsed first).

```
<?xml    version = "1.0"
         encoding = "UTF-16"
         standalone = "yes"?>
```

- Document Type Declaration (optional) – used to specify the **schema** of the doc.

Note that special characters are used in the mark-up language itself therefore we have to represent them using **entities**, for instance & is & which will be transformed when displayed.

Comments are written in the form <!—Comment -->, they must be after the XML declaration.

Namespaces

A single tag name may have different meanings and serve different purposes depending on the application. **Namespaces** are used so that software knows which one to use. They are identified by a URI and every element and attribute has a namespace.

To assign a namespace to the document we do this using the **xmlns (xmlns = 'NS-URI')** element and we add an attribute to it. This will indicate that by default all the element use the attribute as namespace.

<chapter xmlns = "http://book.namespace">

Indicates that the chapter will follow the namespace of the book.namespace.

We can also add it globally by using prefixes. **xmlns:book = "http://book.namespace"** will set book to be from that namespace by default.

XML Schema

XML schemas are written in XML and it's normally refered to as the XSD (XML Schema Definition) and given the extension .xsd.

- The roo element is <schema>
- **Target namespace**: namespace of elements/attributes defined in the schema.

```
<xs:schema targetNamespace = "http://www.example.org"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
 ...
</xs:schema>
```

-

Simple types

For text in side elements we can use: string, integer decimal, Boolean, time, date, dateTime.

To show in a schema that an element has a given type we will use:

Schema format

```
<element name="ELEM-NAME" type="TYPE"/>
```

Schema example

```
<xs:element name = "numberOfPages"
type = "xs:integer"/>
```

XML conforming to schema

```
<numberOfPages>25</numberOfPages>
```

Attributes always have a simple type and are defined in a similar way.

Schema format

```
<attribute name="ATTR-NAME" type="TYPE"/>
```

Schema example

```
<xs:attribute name = "numberOfPages"
type = "xs:integer"/>
```

XML conforming to schema

```
<book numberOfPages = "56"/>
```

We can also give a fixed value or default value of an element/attribute:

```
<element   name = "ELEM-NAME"
           type = "TYPE"
           default = "DEFAULT-VALUE"/>
```

```
<element   name = "ELEM-NAME"
           type = "TYPE"
           fixed = "FIXED-VALUE"/>
```

Enumerator simple types

We can also create types of our own as enumerators which limit the content of al element.

```
<xs:simpleType name = "clothSizeType">
    <xs:restriction base = "xs:string">
        <xs:enumeration value="S" />
        <xs:enumeration value="M" />
        <xs:enumeration value="L" />
        <xs:enumeration value="XL" />
    </xs:restriction>
</xs:simpleType>
```

Pattern simple types

We can also create simple types using regular expressions which values must match.

```
<xs:simpleType name = "postCode">
    <xs:restriction base = "xs:string">
        <xs:pattern
            value = "[A-Z]{2}[0-9]{2}[A-Z]{2}"/>
    </xs:restriction>
</xs:simpleType>
```

Embedded complex types

To specify the hierarchical structure of an element, a complex type is used

```
<xs:element name = "ELEM-NAME">
        <xs:complexType>
                ...
        </xs:complexType>
</xs:element>
```

We can also create these and use them independently, we can reference them.

```
<xs:complexType name = "TYPE-NAME">
        ...
</xs:complexType>
```

```
<xs:element name="NAME" type="TYPE-NAME"/>
```

Elements and types defined in a schema must have a target namespace, reference to types/elements must use namespaces.

```
<xs:schema          targetNamespace = "http://red"
                    xmlns:tns = "http://red"
...
        <xs:element         name = "book"
                            type = "tns:bookType"/>
```

Sequence Complex Type

The **sequence** requires sub-elements to be **present** and **in order**.

```
<xs:element name = "book">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref = "tns:title"/>
            <xs:element ref = "tns:section"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

```
<book>
    <title>Introduction</title>
    <section>Some text...</section>
</book>
```

The **All Complex Type** requires elements to **appear** but in any order.
The **Choice Complex Type** requires one of a choice of sub-elements to appear.

Min and max occurrences

We can restrict how many times a element is used using the *minOccurs* and *maxOccurs* attributes (if not specified then by the fault both are set to 1).

- Zero or one occurrence:

```
<xs:element    ref = "tns:content"
               minOccurs = "0"
               maxOccurs = "1"/>
```

- Two or more occurrences:

```
<xs:element    ref = "tns:content"
               minOccurs = "2"
               maxOccurs = "unbounded"/>
```

Any

Self explanatory.

```
<xs:any/>
<xs:anyElement/>
<xs:anyAttribute/>
```

Note: **attributes are defined after sub-elements in a complex type**

```
<xs:element name = "book">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref = "tns:title"/>
            <xs:element ref = "tns:section"/>
        </xs:sequence>
        <xs:attribute name = "author" type = "xs:string"/>
    </xs:complexType>
</xs:element>
```

To interleave text and a mark-up element we must declare the complex type as mixed.

```
<xs:element name = "book">
    <xs:complexType mixed = "true">
        <xs:sequence>
            <xs:element ref = "tns:title"/>
            <xs:element ref = "tns:section"/>
            ...
```

```
<book><title>...</title>The following sections will...
<section>...</section></book>
```

Hypertext Mark-up Language (HTML)

The root element of every HTML document is ***<html>,*** then the document is split into two elements. The ***<head>*** which contains metadata about the document (including the title) and the **<body>** which contains the content of doc of website.

- <h1></h1> headers which we can customize.
- paragraphs of text.
- <SECTION> element to distinguish a section of a document.
- <CENTERr></CENTER> centres text

We can also add presentational attributed to multiple elements like *align = "center"*

We can use Cascading Style Sheets (CSS) to define presentation rather than presentational HTML elements. This is a quicker, better way of defining how a webpage should look. It can be a separate document (link included in the head) or in the same document. This will give the website a uniform look.

Fragment anchors

Fragment anchors are useful to link to a position in a large document (like if we link to chapter 3 for easy navigation). It uses <a> with the NAME attribute.

```
<a name="cats"/>This section of the page describes cats...
```

- To link to an anchor of a page at a given URL, add #anchor to the end of the URL

```
<a href="http://animals.com/all.html#cats">cats</a>
```

HTML Forms

Pages can be filled in and submitted, the data is sent to a HTTP server. To do this we use a <form> element with attributes **method** and **action**.

```
<form    method = "POST"
         action = "http://ex.com:8080/process">
```

A form has normal html and **<input>** specifying the boxes to be filled in.

- Input text box with space for 32 characters:

```
<input type="text" name="name" size="32"/>
```

- Input box with hidden entry for passwords:

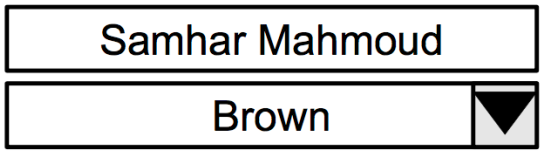
```
<input type="password" name="pass"/>
```

The submission button is then added: **<input type = "submit" />** the data is then send in an encoded form whose MIME is **application/x-www-form-urlencoded**

Encoding

- For every input on the form, a string is created: **InputName=InputValue**
- Spaces in the input values are converted to +
- Any other characters are converted into the hexadecimal form.
- The input strings are concatenated with &.

```
<input type = ”text” name = ”name” size = ”32”/>
<select name = ”eyecolour” size = ”1”>
  <option>Blue</option>
  <option>Brown</option>
</select>
<input type = “submit”/>
```



name=Samhar+Mahmoud&eyecolour=Brown

The method attribute in <form> specifies the HTML method for submission: GET or POST. If it's get then the data is added to the end of the action URL with a ? (query).

http: //example. com: 8080/process?name=Samhar+Mahmoud&eyecolour =Blue

```
GET /process?name=Samhar+Mahmoud&eyecolour=Brown HTTP/1.1
Host: www.example.com:8080
Accept: text/xml,application/xml,application/xhtml
Accept-Language: en-gb,en-us
Accept-Charset: ISO-8859-1,utf-8
Connection: keep-alive
```

With post it is sent concatenated to the URL in action.

```
POST /process HTTP/1.1
Host: www.example.com:8080
Accept: text/xml,application/xml,application/xhtml
Accept-Language: en-gb,en-us
Accept-Charset: ISO-8859-1,utf-8
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

name=Samhar+Mahmoud&eyecolour=Brown
```

Web Servers

Anything that can respond a http request over the internet.

Can be:

- A program
- A pre-packaged software/hardware appliance
- Embedded in a consumer product (ADSL modem/router)

Web server algorithm:

1. Set up TCP connection
2. Receive an HTTP request
3. Process request
4. Access resource referred to by request
5. Construct response
6. Send response
7. If requested, connection closed
8. Log transaction

Web servers need to resolve the resource URIs in requests to retrieve the correct documents.

The **document root** on a host contains all the files available to the web server. The document route and the request resource URI are combined to find the resource.

Document root:

/dcs/web

Request line:

GET /teaching/times.html HTTP/1.1

Resource retrieved from:

/dcs/web/teaching/times.html

Virtual Hosts

- Virtual hosts are host addresses that appear to be different, but use the same web server

http://www.inf.kcl.ac.uk
http://ais.inf.kcl.ac.uk

- The web server is configured to use a different document root for each virtual host

http://www.inf.kcl.ac.uk	/dcs/web
http://ais.inf.kcl.ac.uk	/dcs/web/research/groups/ais

IP and domain name virtual hosts

Name-based virtual hosting:

- You will have one host that operates multiple IPs.
- Each domain corresponds to different virtual host, so has a different document root.

IP-based virtual hosting: \

The Semantic Web

The idea of the semantic web is to include computer-readable information on the web alongside the current human-readable information.

Resource Description Framework (RDF)

Is a data structure to make compute-readable statements.

An RDF is made out of a set of statements, a statement says something about a resource (sometimes how it's related to another source).

They ALL contain:

- Subject: what the resource is about.
- Object: what resource or value the subject is related to.
- Predicate: how the subject and object are related.

We can read an RDF in the form:

Subject predicate object.

Resources

A resource is identifiable by a URI, these give unique identifiers for the predicates of RDF statements.

We write the URI in the statement between < >.

Vocabulary

A vocabulary is a set of terms defined together to allow descriptions in a particular domain (~namespaces)

Each term is a URI and all the URIs in a vocabulary start with the same string

i.e. vocabulary http:/putl.org/dc/terms/ describes things in terms such as:

* vocabulary http:/putl.org/dc/terms/**creator**

* vocabulary http:/putl.org/dc/terms/**publisher**

*vocabulary http:/putl.org/dc/terms/**isReplacedBy**

Turtles and Prefixes

Turtle document: a way to specify multiple statements which end in a full stop. You include the prefixes at the top, then you add the statements further down.

RDF statements can be encoded in different formats including XML

The format we are using is called a **Turtle**

URIs are long to write, we can abbreviate them using prefixes where the prefix replaces the vocabulary URI

- For example, we may say:
  - Prefix **ex:** is mapped to **http://www.example.org/**
  - Prefix **dc:** is mapped to **http://purl.org/dc/terms/**
  - Prefix **inf:** is mapped to **http://www.inf.kcl.ac.uk/staff/**
- The previous Turtle RDF statement then becomes:

```
ex:index.html   dc:creator   inf:samharm .
```

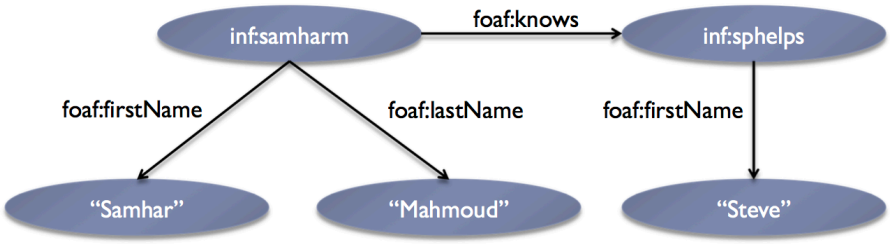
- Prefixes are declared at the start of the Turtle document, e,g.

```
#prefix ex: <http://www.example.org/> .
```

Values

RDFs don't have URIs but data values, a turtle document contains many data values (in the form of statements).

RDF documents can be displayed in a graph. With resources and values at the nodes and predicates at the edges.



Ontologies

RDF allows us to make statements about resources that can be read by software

It doesn't allow the software to reason to determine how best to apply the info

To allow this we need to encode something about the meaning of the resources such as what kind of thing a particular resource is an what is known about resources of that type

Web Ontology Language

The Web Ontology Language (OWL) is a language for encoding ontologies in RDF

An OWL ontology defines a vocabulary of terms but also says how those terms realte to each other in order to allow software to reason over using components caller **reasoners**

OWL is itself a vocabulary for defining the meaning of terms

Classes and individuals

- For example, the statement below says that inf:simonm is a kind of person, i.e. an instance of the class ex:Person

```
inf:samharm  rdf:type  ex:Person .
```

- The URI ex:Person is a term in our ontology, representing the class of all people
- In the statement above, inf:samharm is said to be an **individual**, because it is a specific thing in the world
- rdf:type is so common it is abbreviated as **a**

```
inf:samharm  a  ex:Person .
```

It is presented: instance type and class

- A resource can be an instance of multiple classes
- inf:samharm is not only a person, but a lecturer and a man

```
inf:samharm  a  ex:Person ;
             a  ex:Lecturer ;
             a  ex:Man .
```

- However, we would do not want to have to state that every resource that is a man (or woman) is also a person
- Instead, our ontology can say that ex:Man is a subclass of ex:Person, so reasoners can automatically determine that any resource that is a man is also a person

```
ex:Man  rdfs:subclassOf  ex:Person .
```

You can then build a hierarchy, for instance a Mother is a sub class of a Woman, then we can infer these relations.

```
inf:mary  a  ex:Mother .
```

implies

```
inf:mary  a  ex:Woman, ex:Person .
```

Properties

- Consider an RDF statement about OWL individuals

```
inf:samharm  ex:worksIn  ex:London .
```

- In OWL, the predicate above, ex:worksIn, is called a **property**

We can say more about a property's meaning with OWL, we can say that ex:worksIn only makes sense if it's relating a person to a city

```
ex:worksIn  rdfs:domain  ex:Person ;
            rdfs:range   ex:City .
```

Datatypes

In RDF objects of statements can be values that are not resources: strings, integers etc.

To define the range of one of these properties we cannot refer to a class like ex:Person or ex:City

Instead the range of these properties are **datatypes** such as string or integer

OWL uses XML Schema data types to state the range in these causes, ie: xs:string, xs:dateTime etc

```
ex:hasName   rdfs:range   xs:string .
ex:hasAge    rdfs:range   xs:integer .
```

Social approach to the semantic web: we combine all ontologies (ontology mapping) of small groups since it is very difficult for everyone to agree on a global ontology. To do this we can use class equivalents. This is part of mapping ontologies.

- We can say that one class is equivalent to another class, so any instance of one is an instance of the other

```
my:Person  owl:equivalentClass  your:Human .
```

- where my: and your: are two different ontologies
- We can similarly say that one individual is the same as another individual, just identified with a different URI

```
inf:samharm  owl:sameAs  ex:samharMahmoud .
```

SPARQL

Language used to access information in stores of RDF data. It is an SQL query language for RDF. It has an accompanying protocol (SPARQL Protocol) which sends quires to online triple stores and returns query results.

SPARQL Queries

Finds all the statements or combination of them following a particular pattern and returns some subject and/or objects of those statements

Variables in SPARQL are created in the following ways $var or ?var.

Querying: *any resource for which we have a name and an email address.*

```
{ ?x  foaf:name  ?name .
  ?x  foaf:mbox  ?mbox  }
```

The actual query will be made in the way:

Input:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
      { ?x  foaf:name  ?name .
        ?x  foaf:mbox  ?mbox  }
```

Output:

?name	?mbox
Samhar Mahmoud	samhar.mahmoud@kcl.ac.uk
Steve Phelps	Steve.phelps@kcl.ac.uk

RDFa

RDFa allows RDF to be embedded in HTML

If we add a property attribute to an element marking up text, the attribute value is a predicate relating the webpage to the text

- If http://www.example.org/index.html contains:

```
<h2 property="http://purl.org/dc/terms/title">My first story</h2>
```

- Then the following RDF statement is embedded in the page:

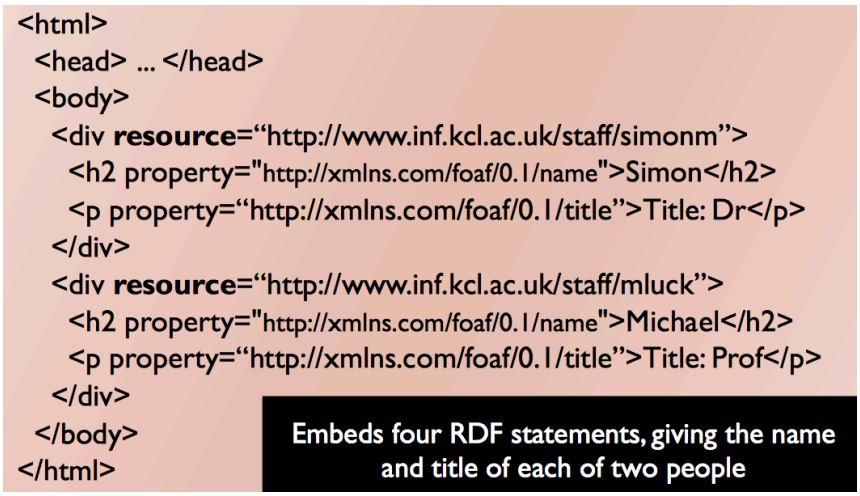
```
<http://www.example.org/index.html>
      <http://purl.org/dc/terms/title>  "My first story" .
```

The text that comes in the tags is the object and the link is the predicate.

Multiple Subjects

We can also embed arbitrary RDF with any subject (instead of the webpage instead like above)

To do this we use the resource attribute to say which subject resource we are making statements about within a given HTML element



Means the statements embedded in this <p> element are not about the webpage but about http//www.inf.kcl.ac.uk/staff/simonm and http//www.inf.kcl.ac.uk/staff/mluck

DBpedia

Huge repository where there are millions of RDFs statements of all the Wikipedia pages to discover relations between them.

Security on the internet

Conditionality

Data is only given to people who has the authority to access it.

Integrity

Data integrity: whenever you try to exchange data, you have to make sure it has not been altered or manipulated.

Software integrity: whatever you try to install on your machine, you have to make sure it has not been altered.

Availability

Data that is available must be usable all the time.

Internet attacks are easy because:

- Action happens at a distance
- Technique propagation:
  Automation: tools that can break security operations.

Vulnerabilities and exploits

Most software's have certain vulnerabilities which can be exploited. Vulnerabilities have many exploits.

Exploit: a piece of code or replicable procedure that you target at a weak point.

- All software will have weak points (although they may not be known)
- When exploits are found, software vendors are informed but there is always a window of time between the time it is detected and when it's fixed.

Zero Day Vulnerabilities

Vulnerabilities that have not been publicly announced are called "zero-day". Anyone that has this is very powerful.

Authentication

The process of determining the true identity of a user.

To defend ourselves from unauthorised users we must distinguish between authorised and unauthorised users (sign in). The process of determining the true identity of a user is called.

- *The simplest technique is to use a secret password or pass-phrase*

Access Control

After the server identifies you, the server says: these is the data you can access. A client may only use services whilst an admin may be able to change settings.

Preventing illegitimate access can be split into two issues:

- Authentication (determining who is trying to gain access to your host)
- Access Control (determining whether that individual is allowed to access a resource on your host.

Web Authentication

Authentication mechanisms of web servers prevent illegitimate access to resources

Resources are often grouped into named realms, which users can be allowed to access

Authentication HTTP status

If a client tries to access a secured wen server with no authentication it receives a response with **401 Authentication Required**

The response will contain a field **www-Authenticate:…**

Proof of Identity

Common forms are:

- Username plus password
- Public key cryptography and Digital Certificates

Software applications (agents) must also authenticate themselves.

HTTP Basic Authentication

Sends username and passwords as plain text in http request.

- You get a 401, authentication required, and basic authentication

- The response will contain a field:
  WWW-Authenticate: ...
- specifying the authentication scheme required
- The client provides authentication in new request

-
- The answer is of type

  username:password (note that the colon IS encoded as well)

  We encode the data in base 64 (encode NOT encrypt)

  Answer would be:

  Authorization: basic QWxhZGR=

Hash Functions

A hash is a data transformation mechanism that creates something which is not understandable by humans. It's a one way function (you cannot go back to the original test).

You will send the hash and the data, then hash the data again and check if the two hashes are the same, this will make sure that the data has not been corrupted or changed.

The strength of the hashing algortithm depends on:

- How hard is it to go back to the original text if was hashed from (should be very hard)
- How many collisions does it have (we want it to make as less collisions as possible)
  - Since it creates a string of a specific size, there can be a collision (when two different texts result in the same hash function)

Password hash

They don't store the original password in their database, they store the hash value of the password. They hash it when you are creating your account and store the hash (different websites will have different hash functions). They also have a "random salt" (or nonce) to prevent rainbow attacks.

&& Rainbow attacks: people figuring out the hash

On unix, the file /etc/shadow contains hashed passwords.

Password authentication

A client sends a server a password and username. This might be dangerous due to eavesdropping. If the channel is unencrypted this data is easy to spy on.

Algorithms for Hashing

MD5

MD5 stands for message Digest (another word for hash)

It takes an arbitrary long string and produces a string of fixed length, the hash value or digest. Collisions are possible, can try it out with UNIX command md5sum

'digest' **3g3gh3ghgdhj3dg3hkdg3io4**

SHA-2

After realizing that MD5 is not secure enough, they then created SHA, currently SHA-2 family of hashing functions (there are many functions that exist under this one).

MD5 is used for legacy applications but the current standard is SHA-2.

HTTP digest authentication

1. Client tries to access a realm on the server
2. Server responds asking for HTTP 401 using digest and providing a unique identifier that can only be used once (nonce)
3. Client sends a digest of the concatenation of username,realm,password,URL,request method and nonce id
   - Digest authentication is indicated by the Digest scheme passed in the 401 response:

     WWW-Authenticate: Digest
     realm="somerealm" algorithm="MD5"
     nonce="564dsd" ...

   - Create digest of password plus other data
   - Digest put into Authorization request field:

     Authorization: Digest Username="Simon",
     response="6629fae49393a053",
     realm="somerealm", nonce="564dsd" ...

- Note that the response field in the response, includes:
- ***username,realm,password,URL,request method and nonce id***
- all of this hashed

How will the server know the correct hash if they only know the hash of the password.

Eavesdropping

Can happen at any point in route between hosts, the whole physical network cannot be realistically secured.

To secure this, we turn to encryption. Hashing was good to make sure your information cannot be read. But it is not revesible, it cannot be un-hashed.

With encryption we are able to decrypt.

We use cryptographic ciphers which are what encrypts the data.

Link encryption: encrypts all communication across a physical link. Very computationally expensive and unrealistic over large scale.

Document encryption: documents are encrypted, sent then decrepted.

Transport Layer Security: TLS encrypts all messages at the TCP Layer.

Hashes and Encryption

Using both hashing and encryption you cannot go back if you do not know the right information (that is the public and private key).

Ciphers

Have two functions:

Encryption function:

$$C = E(M,K)$$

Decryption function:

$$M = D(C,K)$$

- Both has and encryption of a message are a transformation of that message into some new data that gives no clue to the original
- The original message can be computed from the encrypted data (decrypted)
- The encrypted message is called the cipher-text.
- The decryption function can only be computed if you have access to the correct secret key K.

Same key is used to encrypt and decrypt (symmetric encryption [using the same key]), the most famous type for this is Advanced Encryption Standard (AES).

Single key encryption

Less secure type of encryption, a key is a secret piece of data than the sender and receiver can use to encrypt and decrypt the message, since no one else knows the key no one else can access the message.

Its called symmetric cipher and the most famous one is Advanced Encryption Standard (AES)

Key Generation

- You mustn't use trivial keys, they must be hard to guess
- They should be randomly chosen
- It is very hard to generate genuinely random data using deterministic digital computers
- Usually the best we can do is to use a pseudo Random Number Generator (PRNG)

Key Sizes

The space of possible keys has to be large - to ensure that someone cannot use all the possible keys and be able to decrypt your data. To avoid a **BRUTE FORCE attack.**

For AES (example): maximum key size is 256 bits, which represents 2 to the power of 256 different combinations.

Key Management

We need separate keys for every possible pair od used.

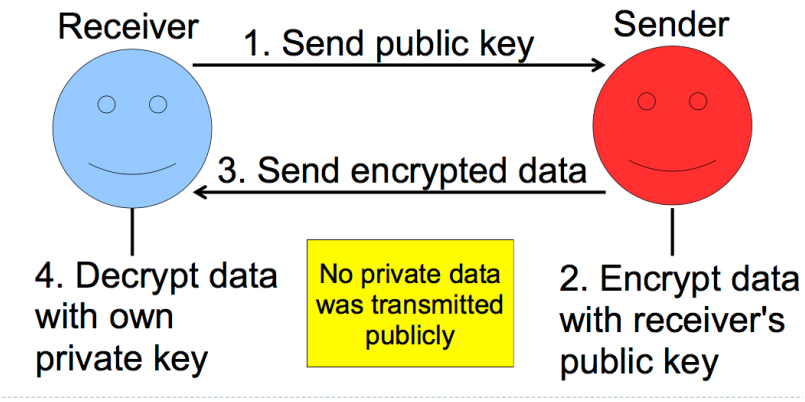
We need to agree on a key with the other party.

If we use the same, then everyone on server1 will know how to access server 2 from my account since the key is the same. **We cannot send the key, because of eavesdropping.**

Asymmetric Encryption, Public Key Encryption

When we create a key, we create a key pair. A public and private key (called asymmetric ciphers)

The public key is of public knowledge, used by anyone who wants to communicate with me. **When we send a message its encrypted by the public key**, **it can only be decrypted by private key.**



RSA

Encryption form which uses two very large prime numbers to generate public and private keys

1. Generate two large primes: p, q
2. Calculate the product: n = pq
3. Calculate the totient: m = (p - 1)(q - 1)
4. Find a co-prime to totient m: e
5. Choose integers d, i so that: d = (1 + im) / e
6. The public key is (n, e)
7. The private key is (n, d)

Just for illustration we are using 7 and 11, IRL we would use very large prime numbers.

1* Generate two large primes: 7,11 - (p,q)

2* Calculate the product: 77 – n = p*q

3* Calculate the totient: 60 – m = (p-1)(q-1)

4* Find a co-prime to the totient: 7 - e (Count up each prime, and find the first not divisible into totient)

5* Choose integer I,d: 5,43 (d = (1+im)/e (you can choose i randomly as long as the you can 1 + im is divisible by 7)

6* Public key is (n,e)

7* Private key is (n,d)

E is 7 and m is 60, we have to find I and d.

Steps 4 and 5 are both ways in which we can randomise our output.

RSA Encryption Algorithm (how to use public and private keys)

You breakdown your message into large array bits. It will represent a particular number.

We change the data into Binary, we then change this into a number and then encrypt that number.

1. Obtain receiver's public key (n, e)
2. Convert message to an array of bits representing a large integer
   $M < 2^n$
3. Compute encrypted values $C = M^e \bmod n$

---

1. Obtain receiver's public key $(77, 7)$
2. Represent data as +ve integer 6
3. Compute encrypted value $6^7 \bmod 77 = 41$

RSA Decryption Algorithm

1. Use private key (n, d)
2. Receive encrypted message C
3. Calculate original value $M = C^d \bmod n$

---

1. Use private key $(77, 43)$
2. Receive encrypted message 41
3. Calculate original value $41^{43} \bmod 77 = 6$

RSA Signing

Encryption is good but it does not confirmed that the person encrypting the data is the person they say they are. RSA can be used for digital signature.

You send a message to the server and the server answers you asking for authentication. How do you know this is the server and not someone else.

TO avoid this, the sender uses their private key to produce a signature.

I send my response and I attach a signature. I hash the response and then sign it (encrypt it) then send it.

At server side, we decrypt to get the hash representation of the message, you then check that the message that was decrypted is hashed and checked with the decrypted hash that was sent in the signature.

USER:

Sends encrypted data, encrypted hashed data.

Server:

Decrypts data and to check it was not modified, decrypts certificate, hashes data and checks two hashes.

The signature S of a message M is:

$$S = M^d mod\ n$$

To verify the signature, the verifier checks that:

$$S^e = M\ mod\ n$$

We typically hash the message, and then sign the hash.

MD5 is not suitable for digital signatures- use SHA-2 or SHA-3.

You can later use a single key encryption (its more computationally efficient) but to exchange this unique key you use the public private key way to transfer this. Usually, this symmetric key (AES single key) is only used for that current session to add extra security.

Hybrid Crypto Systems

For large messages RSA is generally used with AES. RSA can be used to encrypt messages containing the symmetric keys which can then be distributed over a public channel. The symmetric key is typically a temporary key that is only valid for that particular session.

Transport Layer Security

SSL (Secure socket Layer) operates between TCP and HTTP, will use the most secure supported method by both the client and the server. TLS is the "official" accepted name of SSL.

TLS Negotiation Protocol

TLS initialized a cryptographic protocol between hosts with a hello method. In this message, they will declare the cryptography methods that they both support. Then the most advance one that is shared is picked for the communication.

TLS sharing certificates

The host shares certificates that provide verifiable host data for authentication

They also provide public keys for encrypting the communication.

Man-in-the middle attacks

TLS tackles this attack, if data is not encrypted the man in the middle can take this data. Else if it is encrypted thanks to the TLS layer this data is rendered useless since its encrypted.

TLS and the internet architecture

TLS operates over TCP but under HTTP, TFL is the official name for SSL/

HTTPS

- HTTPS is HTTP over SSL/TLS
- Uses its own URI scheme
- https:---
- has different default TCP port (443)
- otherwise everything else is the same as HTTP
- An https WEB SERVER MUST HAVE A DIGITIAL CERTIFICATE

Digital Certificates

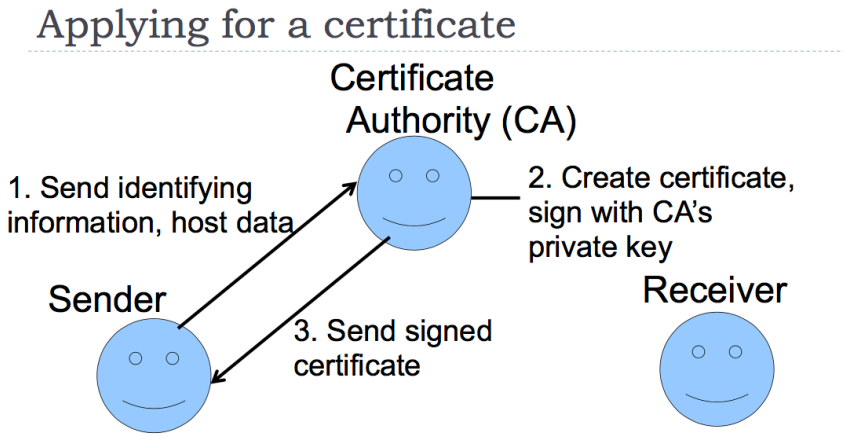
A digital certificate is a block of data about a communicating host that is signed, this means that an encrypted hash of the host data is added so that other hosts can check that you are who you say you are and that the data has not been modified.

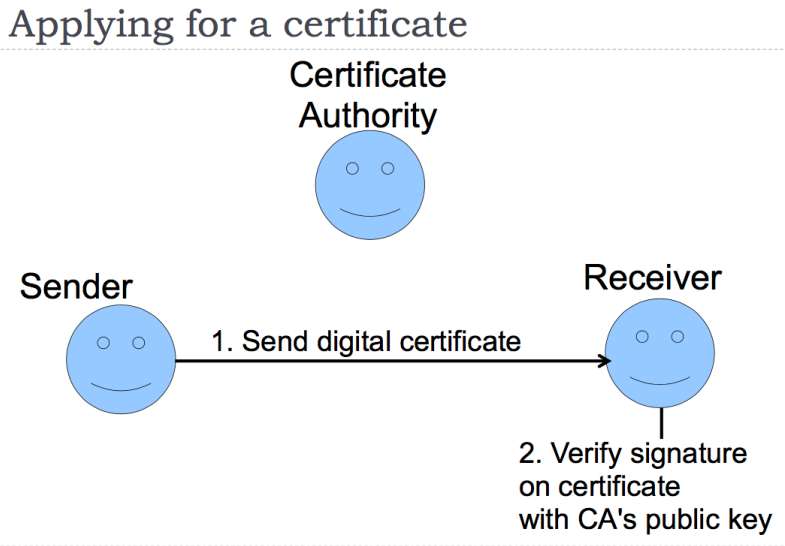
Certification Authorities

A certification authority is an organization responsible for issuing certificates and verifying their correctness

If a host's certificate is signed and a CA then another host trusting the CA ay reliably allow that the certificates public key belongs to the host stated

Publicly trusted CA are VeriSign and CertCA





X.509

One of the most popular certificates will contain 3 parts:

- Certificate details
- Signature of certificate
- Algorithm to sign certificate

The details include:

- Unique serial number
- The period that is valid for
- Name of issuer
- Unique identifier for the issuer
- Name of certificates owner
- Public key of owner

Signing a certificate says: I am who I'm saying I am, you can check it.

Internet Paradigm Shift

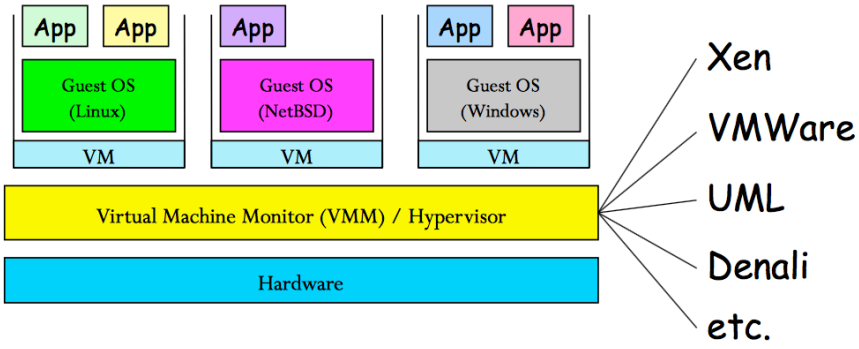
Virtualisation

Virtualisation is used to run multiple virtual machines on a single physical server to provide the same functionality as if there were many machines running.

Software that does this is named **hypervisor**. It is very similar to an operating system. This is dangerous since leaks in one operating system may help violate another one.

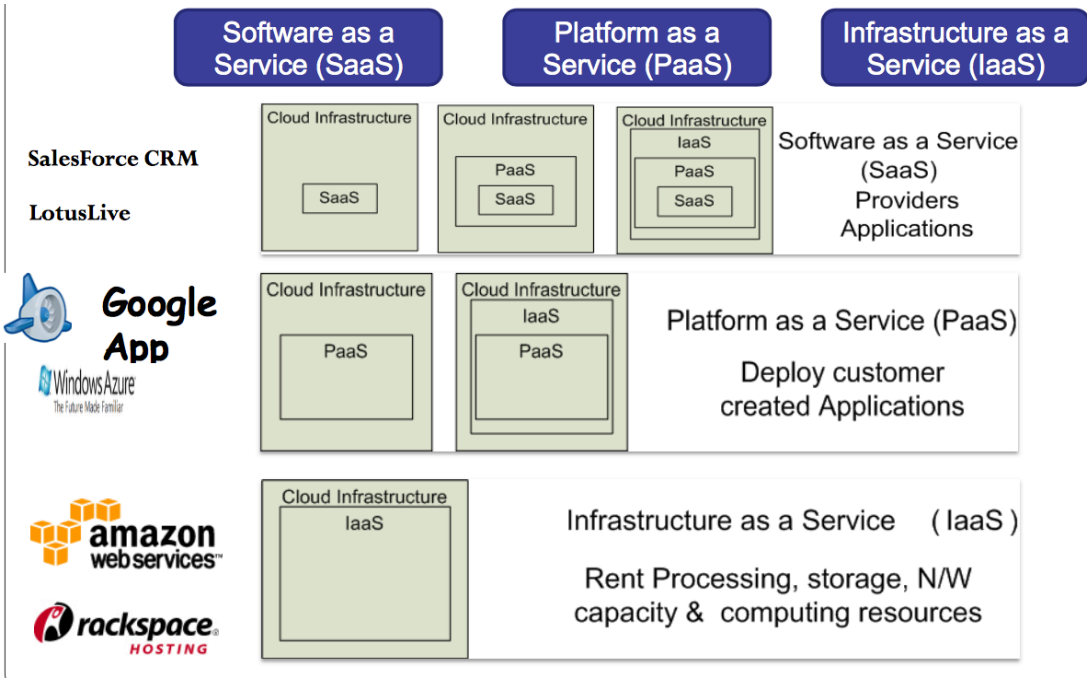
Virtual Machines

- VM technology allows multiple virtual machines to run on a single physical machine.



Cloud Computing

- Virtualisation is the key to the success of Cloud Computing
- Cloud Computing is Internet based computing, whereby shared resources, software and info are provided to computers and other devices on demand
- Hides the complexity and details of the underlying infrastructure from users and applications by providing a simple graphical interface or API (Applications Programming interface)
- Invoked only when you need them, no permanent parts of your IT infrastructure
  - On- demand resources
  - No need to have dedicated resources waiting to be used



Software as a service (SaaS)

Tool for clients to use remotely, we create an application and we host it in our server where many people can access it remotely. People often have to buy a license to use these services.

- Software is managed from central location
- Users don't need to upgrade the software, you can do it locally.

Platform as a Service (PaaS).

Computing platform that allows the creation of web applications quickly and easily and without the complexity of buying and maintaining the software and infrastructure underneath it.

- Services to develop, test, deploy, host and maintain applications in the same integrated development environment
- Web based user interface creation tools help to create, modify, test and deploy UI scenarios
- Multi-tenant architecture where multiple concurrent users utilize the same development application.
- Integration with web services and databases via common standards
- Support for development team collaboration (like project planning and communication tools)

Doesn't provide the software but an infrastructure that allows people to create software using it. You don't need to buy and maintain local machines. Provide templates and other services. Eg. WordPress, Google App.

Infrastructure as a Service (IaaS)

Rather than buying a datacentre to store it, you can store it on the cloud for a few months. It is distributed as a service and can be scalable. If I rent 100GB and then need more I can do that quickly (you don't have to go out to buy actual HDDs).

- Amazon Elastic Commute Cloud: allows you to quickly configure what are the settings for the data centre and they make optimal access and solutions for the demands of their clients. They have datacentres in different places and different sizes. Add or remove computer resources and distribute traffic.

Cloud Computing Pros

- Elasticity and scalability: reduce or increase resources at any time.
- Workload movement: you can distribute workload to make optimal use of resources (no one machine is doing so much work and another none.)
- Resilient: if you have a failure of a component you can redirect the work to another machine without the user even noticing.
- Multi-tenancy: allow you to have multiple applications on one machine and make optimal use of computational power.

Cloud Computing Concerns

- Performance, reliability and SLAs
- Control of data and service parameters
- Application features and choices
- Interaction between Cloud providers
- No standard API – mix of SOAP and REST
- Privacy, security, compliance, trust….

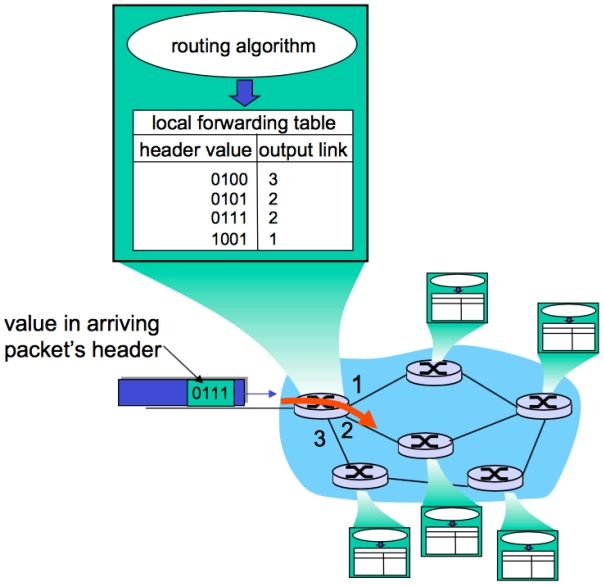
SDN

Typical Networking Software:

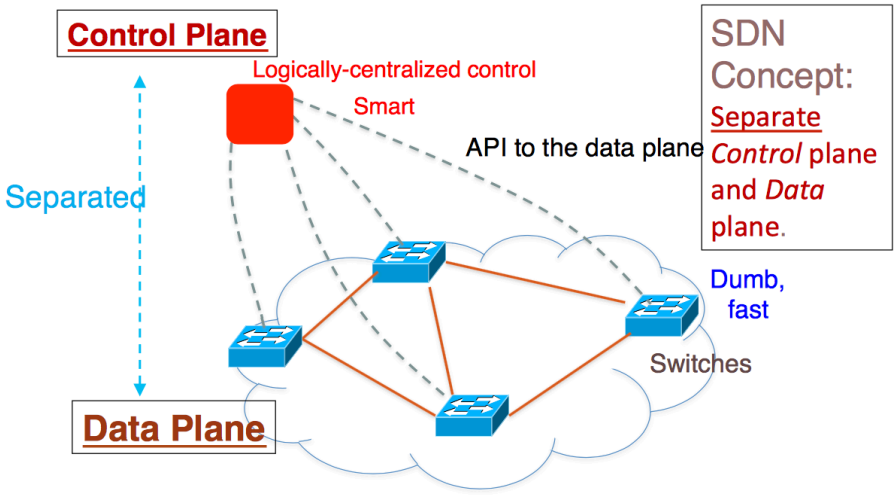
- Control Plane – brain decision maker
- Data plane – packet forwarder

We used to have specialized closed systems that work only with other specialized components, we try to make this open to use different control planes and different apps.

Nowadays we have a routing algorithm that checks what's the best way to forwards packets. Then we have a local forwarding table that given a header value it will forward it to a specific link.

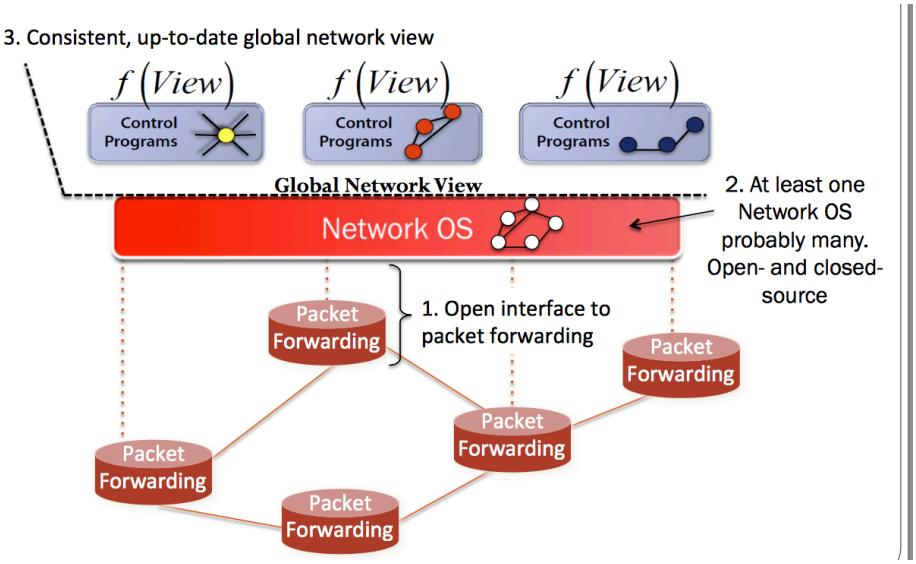


The SDN concept separates the Control plane and Data plane with the objective of speeding up the process.



Software Defined Network Architecture

It's able to construct a global network view for the control programs which can see this view and do a particular calculation task.



SDN Basic Concept

Separate Control plane and Data plane entities with centralized Network intelligence and state.

To control not just a networking device but an entire network

Use commodity servers and switches

Decoupling all the complicated task from the networking hardware

SDN helps reducing financial costs and delays (you do not have to manually update multiple devices) by building their own cheap simple hardware and created their own software for their data centre.

Consequences

- More innovation in network services
- Lower barrier to entry for competition
- Lowe cost of infrastructure and Management

Network OS & Control Program

Network OS

- Distributed system that creates a consistent up to date network views. It runs on servers in the networks.
- Example: HyperFlow, Kandoo, Maestro …

Control Program

- Operates on view of network
- Input global network view graph/database
- Output configuration of each network device.

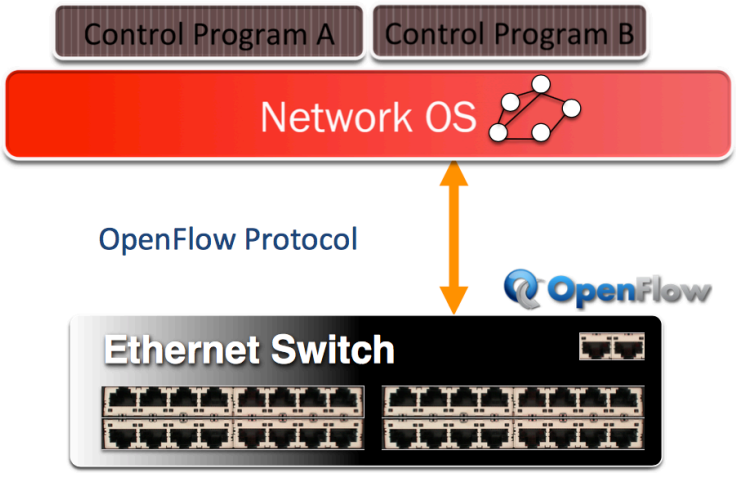
Forwarding

Flow: application flow, all http traffic

Actions taken: allow or deny, route or re-route, isolate flow, make it private or remove it.

Open Flow Switch

A protocol that allows the SDN to talk to switches. It allows the separation to happen. Provides centralized control and takes advantage of routing tables in Ethernet switches and routers.



SDN IS NOT OPENFLOW

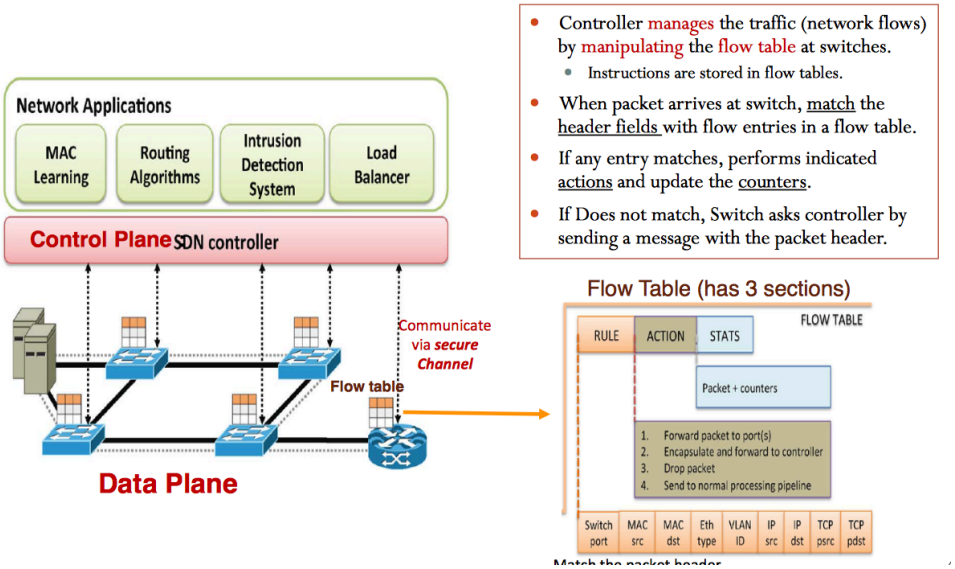
SDN: a concept of physical separation for the network control plane form the forwarding plane, and where a control plane control several devices.

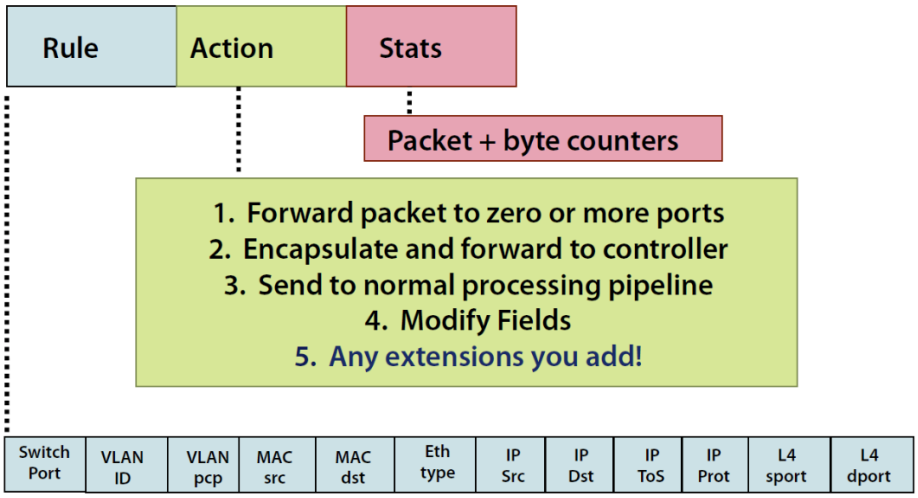
OpenFlow Protocol is an agree standard with SDN.

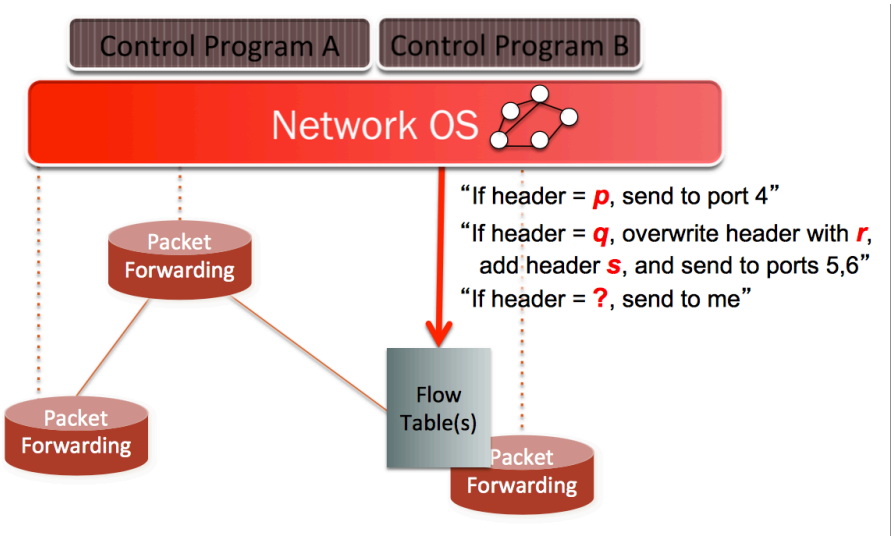
Open Flow

Communication interface between the control and data plane of an SDN architecture

- Allows direct access to and manipulation of the forwarding plane such as switches and routers (physical and virtual)
- Think of it as a protocol used in switching devices and controllers interface







OpenFlow Table: Basic Actions

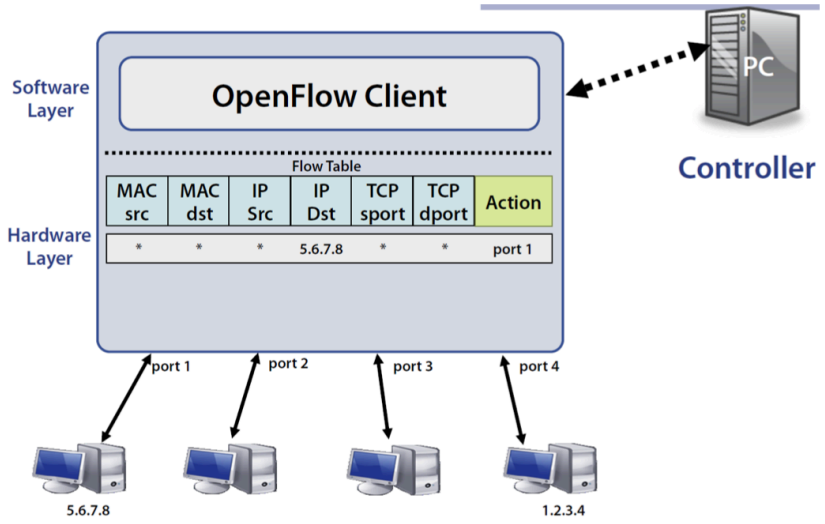
- All: To all interfaces except incoming interface.
- Controller: Encapsulate and send to controller.
- Local: send to its local networking stack.
- Table: Perform actions in the next flow table (table chaining or multiple table instructions).
- In_port: Send back to input port.
- Normal: Forward using traditional Ethernet.
- Flood: Send along minimum spanning tree except the incoming interface.

OpenFlow Table: Basic Stats

Per Table	Per Flow	Per Port	Per Queue
Active Entries	Received Packets	Received Packets	Transmit Packets
Packet Lookups	Received Bytes	Transmitted Packets	Transmit Bytes
Packet Matches	Duration (Secs)	Received Bytes	Transmit overrun errors
	Duration (nanosecs)	Transmitted Bytes	
		Receive Drops	
		Transmit Drops	
		Receive Errors	
		Transmit Errors	
		Receive Frame Alignment Errors	
		Receive Overrun erorrs	
		Receive CRC Errors	
		Collisions	

- Provide counter for incoming flows or packets.
- Information on counter can be retrieved to control plane.
- Can be used to monitor network traffic.

Example



Virtualisation in Networks

Instead of putting network functions on high end switches, put it on every machine we have and create software that allows for these services. Virtualise the switch on the machine.

Advantages of virtualisation in other domains:

- Better utilisation of resources

- Programmability: able to change behaviour on the fly. You can change it at a software level not a hardware level.
- Dynamic Scaling: ability to change size, quantity.
- Performance: optimising network device utilisation.