LT1 - Intro

Some stuff and tings

LT2 - Models, Methods, Metal

In the ideal world we would have

- Infinite bandwidth
- Infinite speed
- Zero latency
- Zero errors

However in reality we have

- Finite bandwidth
- Finite speed
- Errors
- Latency

Latency - How long it takes for a bit to travel through the network

In the real world the lower bound for latency is the speed of light so we don't have much to worry about.

Error Rate

The following factors (and many more) contribute to data loss, corruption etc.

- Cosmic ray background
- Mains noise
- Cable and connector issues

Voice over IP for example can take low bandwidth and high errors, but it cannot tolerate high latency. Whereas **OS downloads** require reliability over everything.

Over time **link bandwidth** tends towards infinity, currently on the order of **terabits / s**, so latency is the bottleneck. Some of the contributing factors towards latency include

- Processing times
- Speed of light (lower limit, 1ms round to London, 56ms round to SF)

Raw error rates are roughly constant, we can trade off **processing power and bandwidth** for better **error correction**

It is still very common to fill a jumbo jet with hard drives for fantastic bandwidth and terrible latency. So for transferring a full data center this is great.

Packet Switching

- Split the data stream into units called *"packets"*
- Give each packet identifying information
- Switch packets over the network until they reach their destination
- Packets can get lost, delayed or re-ordered

Advantages

- Multiplexing in **time** rather than **frequency** so much simpler
- **Statistical gain** on bandwidth
- Rerouting data around failed switches for **reslilience**

Disadvantages

- Delaying packet till there is room on the line, introduces **latency**

Frequency Domain Switching

- Distinguishing two signals by their frequency
- e.g. red and blue light, low and high pitched sound

Time Domain Switching

If we have two data streams each with a bandwidth that is half that of the line, we can buffer the arriving data and when the buffer is full send it off at the line bandwidth. This means the output is idle for 50% of the time.

Statistical Gain

- Assume most users are not using full line bandwidth 24/7

- Oversell bandwidth, e.g. selling 100 x 10Mbps when line rate is 100Mbps
- Contention ratio, 50:1 -> 50 customers sharing the same bandwidth :(

Virtual Circuits

- Endpoints tell network to establish a connection
- Network sets up a path to the destination, gives back identifying token
- The token identifies a *"virtual circuit"* linking two endpoints
- Connection is torn down when endpoint has finished with it
- Network has to know about every connection
- Each packet in a connection follows the same route
- Network attempts to solve ordering and packet loss but there are no guarantees

Datagram Services

- Packet contains addressing information
- Each packet considered seperate
- Endpoints deal with loss, duplication and corruption
- Might arrive, might not
- Nedwork doesn't need to know about connections, it just routes packets

Layering

Applications generally want guarantees, knowing that if a file is sent it will arrive undamaged or you know if something went wrong.

A network can be thought of as a **stack**, a succession of interfaces starting with services needed by real applications and getting closer to transistors and volts. Each layer provides services to those above it and makes use of services below it. Each task appears in only one layer.

Table 1: *Abstraction in the networking stack*

Layer n+1	More abstract service, closer to application
Layer n	Less abstract service, closer to wire

Competing Models

OSI Model

The **OSI** model basically failed, it came long before any successful implementation.

DoD Model

The **DoD Model** was a rationalisation of already established practice, it was made partly to compare the TCP/IP architecture with the OSI proposals.

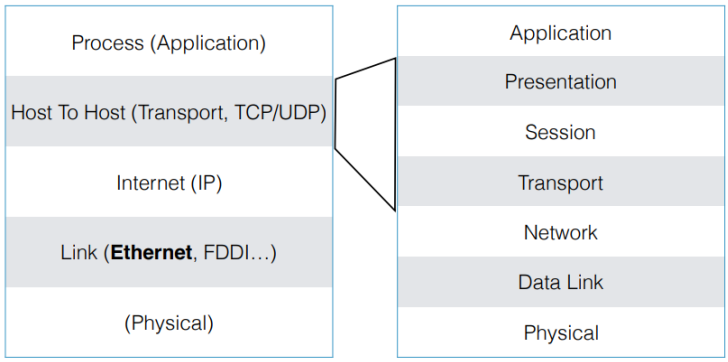


Figure 1: *DoD vs OSI Model*

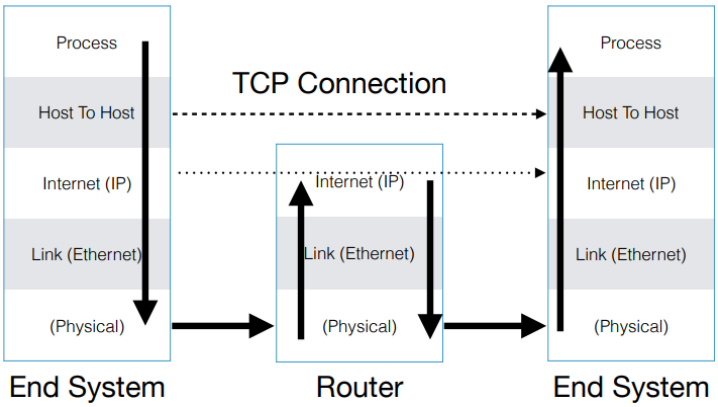


Figure 2: *The DoD Model*

The **Application Layer** runs code that does useful work e.g. SMTP, IMAP, HTTP, SSH, they need services to move data from computer to computer

The **Transport Layer** moves data between two end systems via a network with toplogy and properties that the transport layer doesn't know about. **TCP** is used for streams of data, **UDP** for packets. Properties of **reliability**, **sequenced delivery** etc. are guaranteed. The layer also permits communication between entities running on the same end systems.

The **Internet / Subnet / Network Layer** moves packets between end systems, doesn't offer any guarantees about reliability. Handles choosing which link to use to get data closer to destination. It also doesn't deal with the concept of multiple applications. e.g. IPv4, IPv6

The **Link Layer** moves data between one network element and the next. Concerned with packetisation, addressing etc. Many protocols are in use but **Ethernet** is our main focus.

The **Physical Layer** consists of Ethernet over co-axial, twister pair, fibre, radio etc.

TCP/IP Wins

- Single transport service for both connections and datagrams
- Full responsibility of the implementor to make TCP and UDP work
- Exactly one network layer – **IP**
  - **IPv4** and **IPv6** differ only in address length
- **Physical** and **Link** layers carry **IP**, if they can't then they can't carry **TCP** or **UDP**

OSI lost because it tried to provide multiple transport services, there were also two different network layers for virtual circuit vs. datagra, and none of it interworked.

Value Chain

- Telcos sell a service with a service level agreement
- ISPs are more "best efforts" and rely on statistical gain

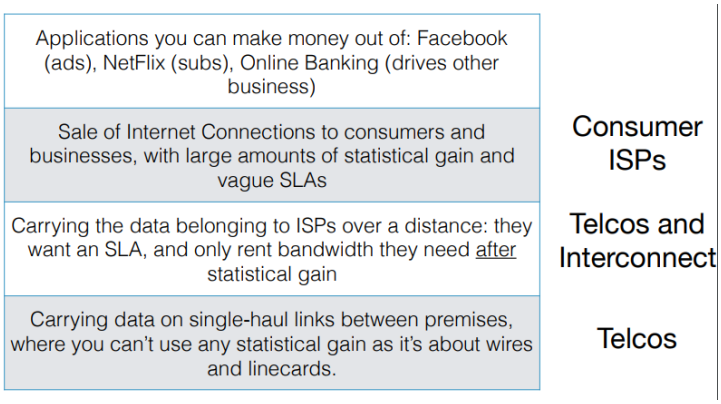


Figure 3: *The Value Chain*

LT7 - C

cdecl.org converts code into plain english.

Pointers

A pointer is a variable which “points” to another variable.

* - Follow the pointer and give the value it points to, **dereference**

& - Get the address of the pointer

```
// x is an integer, sets aside memory on the stack
int x;

// x is a pointer to an integer, memory is not allocated
int *x;


int x; int *x_p = &x;
*x_p = 3

int array[] = {63, 43, 32};
int *pointer = (int *) &array;
```

Arrays

Arrays are just syntactic sugar for pointers, so `int a[100]` reserves space for 100 integers and **a** is a pointer to that space.

So `a[7] = *(a+7)`

Arrays are almost always zero filled, but don’t rely on this, if you need something to be zero filled then do it yourself.

Pointer arithmetic

Pointers are assumed to point to a type twhich hasaknown size, so when multiplying they are really multiplied by the size of the type.

Debugging

Assert

```
#include <assert.h> // this should be everywhere
```

Use assert **everywhere**, asserts in, asserts out, asserts all about. If an assert fails then the code immediately exists, with a core dump if possible. Use it on arguments, return values, items in structures etc. If you can check return values and use `perror` this will help with debugging.

This helps to avoid code limping on and dying at a later time, making it difficult to trace the source of the error.

- Assert pointers are non-null
- Assert sizes of objects are what you expect
- Assert that malloc has worked properly

GDB

You have to compile with `-g`, you can also do post mortem debugging by loading the core dump with gdb.

Structures

A **struct** is a composite which contains elements of various types.

If you declare a struct you can dot access it and it is immediately stack allocated. If you declare a pointer to a struct you have to manually malloc it and you use the

It's worth using **typedef** as its tidier and makes code more readable.

You can take the address of individual elements in a struct. `&(pair_p->y)`

Strings

C strings are syntactic sugar for strings, that by convention are arrays of chars followed by a zero byte.

```
// these both represent the same string
char *s = "abc";
char s[4] = {'a', 'b', 'c', '\0'}
```

If you have an array of char which includes a zero byte then you need to use `write`, `send` or `fwrite`. The same applies to all the other string routines `strcmp`, `strcpy` etc. Instead use `memcpy`, `memcmp` and so on.

This is common with files such as PDFs so avoid hacks like trying to add a `\0` to the end of the buffer and call a string :(

Function Pointers

Functions in C are **call by value**. Changes made to arguments are not propogated back to the caller, as the arguments are copied into the function.

If you want to be able to modify the arguments then **you have to pass a pointer** to the original data.

By convention you should not pass structures to functions, this used to not be permitted. Instead you should pass pointers to structures.

Threads

Threads have their own stack, so don't access other threads stack. When using `pthread_create`, it's always best to pass in malloc'd space, not a pointer to anything else.

LT8 - IP Protocol

- Dominant networking protocol over the past thirty years
- Single network lalyer, over which all transports run
- Single network layer which can run over all available lower layers

What is IP

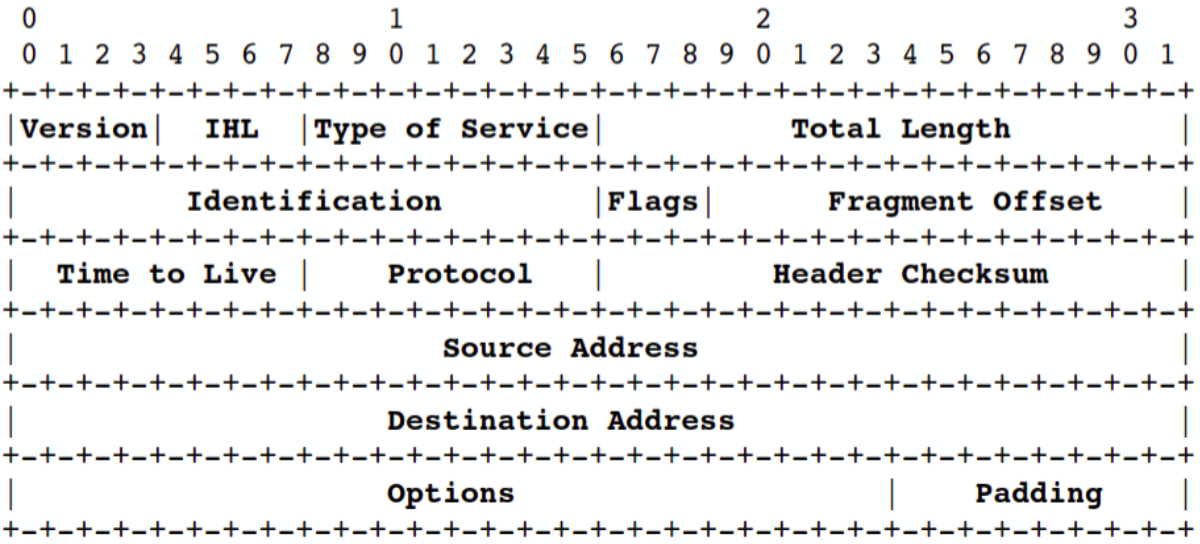
- Unreliable, unsequenced datagram service
- Put an address on a packet, the network attempts to deliver it somewhere, hopefully the right place
- No checksum covering the data (there is one for the header), corruption protection is the responsibility of the upper layers
- Barebones

Why did IP win

- Easy to implement
- Works over all mediums e.g. long haul radio to high speed fibre
- Support of US DoD, ARPA, NSF

Packet Format

- Each row is 32 bits, four bytes, 1 word
- Typical header is 20 bytes, 5 words
- Fields aren't byte aligned
- Options are optional



32 Bit Addresses

- At the time, insanely large
- Original concept: 8 bits indicating the site, 24 bits specifying a machine at the site
- Obviously 256 sites was not enough

Notation

- Four decimal numbers, each encoding one byte (wastefully) seperated by dots
- Typically not zero padded

  `192.168.1.1`

  TODO

Routing Decisions

- Identify network using leftmost part of address
- Identify host on network with the rest of the address
- Router ignores host for all non-local addresses
- Network is looked up in a routing a routing table, chooses which interface to send a packet through
- “Default network” in all other cases

Routing problems

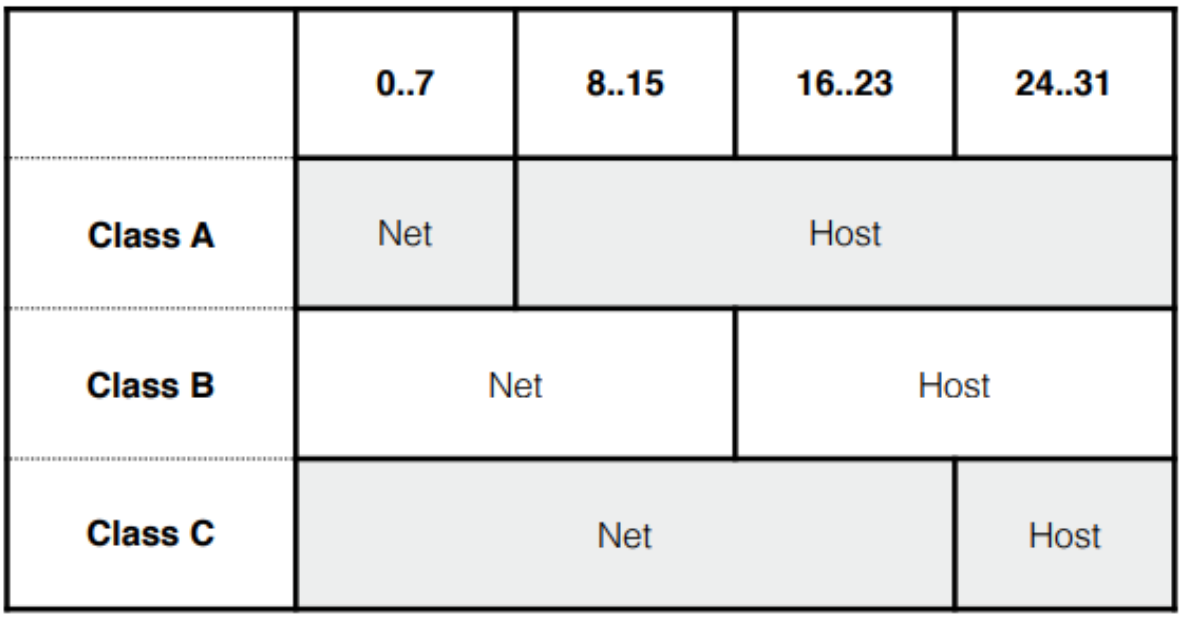
- Speeds and ram limitations of the time made building large distributed routing tables hard
- Limiting to 256 networks was unreasonable
- 32 bits, ~ 4 billion addresses, utilisation was anticipated at a fraction of a percent

“Classful” Addresses

If the address starts with a 0, the first 8 bits identify the network, the remaining 24 identify the host *1.x.y.z* through *127.x.y.z* So there are 128 “Class A” addresses for large sites, they get 2^24 (~16 million) hosts each.

If the address starts with 10, the first 16 bits identify the network, the remaining 16 bits identify the host *128.x.y.z* through to *191.x.y.z* So there are 2^14 (16384) “Class B” addresses, they get 2^16 hosts each.

If the address starts with 1010 etc. there are also classes C, D and experimental class E.



Wasteful allocation

- Totally reachable space is 87% of the available addresses
- Class A space is very sparsely occupied, as is class B
- In both cases very few hosts are externally accessible
- Class C space was initially available to everyone
- Likely that < 25% of address space is usefully deployed
- Huge shortages outside USA, Canada and west Europe

Easy for routers

- Most of early internet was class A addresses
- Look at the address to see if it's local, send straight to the destination
- Otherwise Look at first bit of address, if zero then lookup first byte in 127-entry "next hops" table
- With 32 bit addresses a 128 entry table takes 512 bytes, easy.
- If found, send packet to that router
- Otherwise look up remaining bytes in more complex table
- Otherwise use default route

Routing vs Memory

- Fully populated routing table for every /24 in 128MB of RAM

- Every unique destination for every IP number in 32GB of RAM
- Routing not a big computational problem, can just index into a sparsely populated array instead of trees or hash tables
- No need for caching

Sub-Netting

- Using large amounts of address space to plan internal networking by structuring “host” part of the network
- Class B treat 2^16 addresses as 256 networks of 256 hosts
- Later you could “subnet” on non-byte boundaries
- Systems that don’t support this are now obsolete
- Outside world sees one network, internally everyone knows extra information about address layout

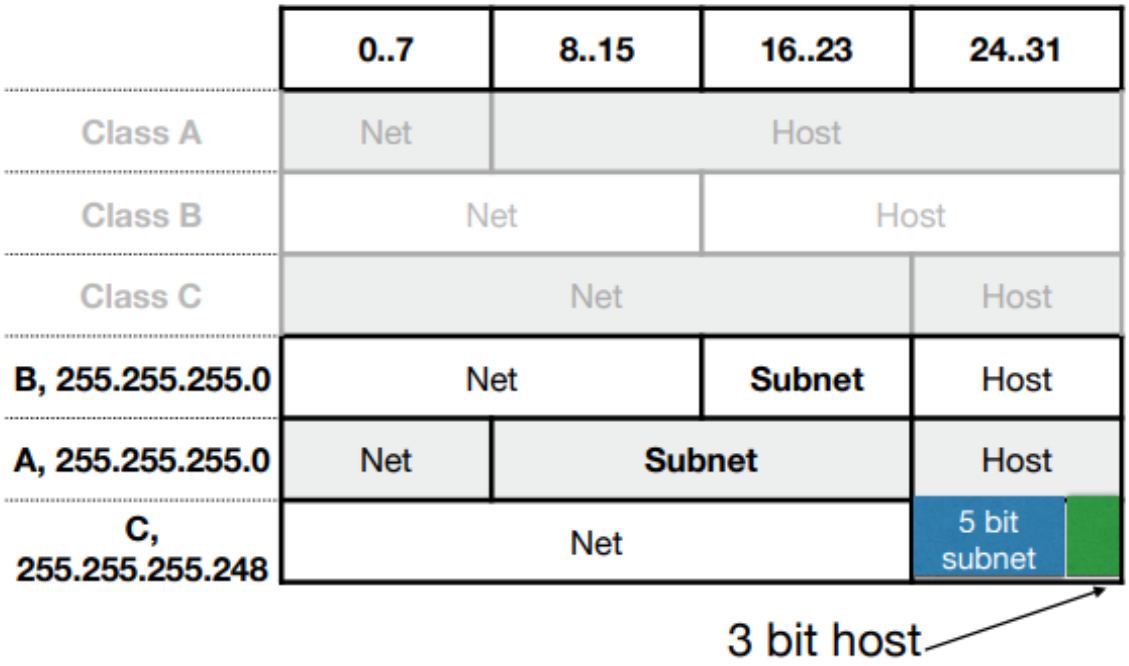
Netmasks

- The bit pattern which can be bitwise and’d with an address to yield a network number

```
Class A (/8)) - 255.0.0.0

Class B (/16) - 255.255.0.0

Class C (/24) - 255.255.255.0
```



CIDR and Slash Notation

- Classful networking was wasteful and needed more flexible sub-netting
- Hence, Classless Interdomain Routing, **CIDR**
- Every network address has a netmask which describes how much of it is network and how much is host
- Class A now */8*
- Class B now */16*
- Class C now */24*

```
18.0.0.0/18
// First 18 bits are network number, // rest are host
```

- *Super netting* rarely used to group lower classes into blocks
- Eight contiguous class Cs placed under a */21* for example

Non Byte Masks

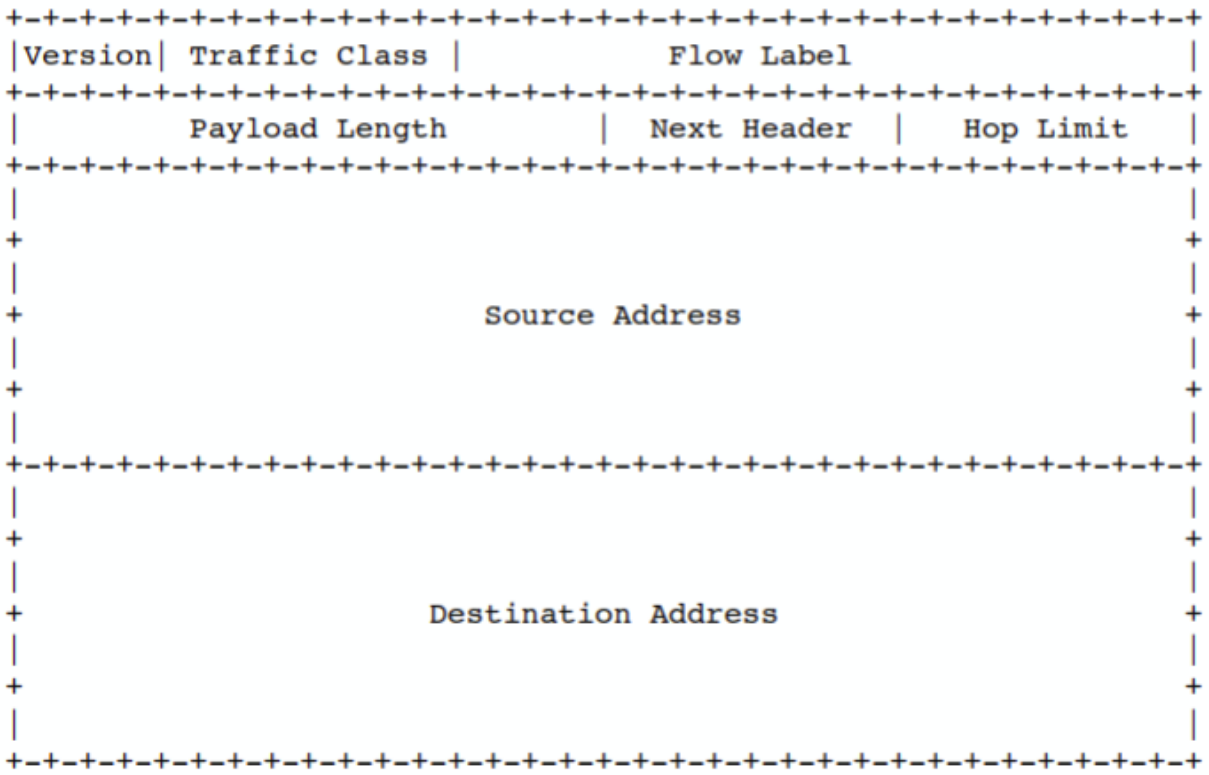
- Extends to boundaries that are not classful
- A */28* would give the user 16 IP addresses

Recovering Class As

- Some of the class As were recovered (by threats, bankruptcy etc.)
- Allocation stopped at 57 anyway
- Remainder and returns were broken up into different clsases

IPv6: IP with big addresses

- 32 bit addresses have run out
- 128 bit addresses gives 2^96 more addresses
- Minimum allocation is a /64, you're mobile phone will probably get a /64
- IPv6 is "really" a 64 bit protocol
- 2^64 gives 2^30 per person, ~ 1B



- Hex string, broken into 16 bit chunks, width colons, no leading zeros - Certain addresses are reserved

IPv6 Routing Tables

- In the long-term will require complexity originally used for Ipv4
- Very sparsely used currently

- 2^64 addresses = 2^24 TB, obviously a way off but so is 2^64 allocated networks

IP in Operation

- Sender: choose an interface believed to be closer to the destination, send the packet
- Recipient: if the packet is for us, process locally, otherwise send it on

IP on Ethernet

How do we find the address of the next "hop" on an ethernet?

- IPv4 uses **ARP** - Address resolution protocol
- Simple, old, insecure
- Ask "who has" a particular IP
- Station with that IP or someone who knows tells us
- Ripe for exploitation
- IPv6 uses "Neighbour Discovery Protocol" to do a similar job

IP To Gateways

TODO

IP Data Transfer

On Point to Point Links

- Just send a packet down the link
- Might be physical, might be a tunnel

To Gateways

- Look up gateway address using **ARP**
- IP destination remains the ultimate destination
- Gateways don't appear on IP header
- Gateway at other end of point-to-point links just involves sending the packet

Hop Counts

- Each packet has time to live **TTL**
- Decremented each time packet is processed, or router holds it for more than a second
- When it hits zero, packet is discarded "ICMP Time Exceeded" error report is generated
- Typical initial value **30 - 60** depending on estimates of "diameter" of internet
- Prevents packets circulating endlessly
- Requires re-computation of header checksum in IPv4
- Fast algorithms can recompute based on knowing what's changed (not a secure hash).
- IPv6 doesn't have a header checksum, it instead relies on lower layers being reliable and upper being sensible
- Good reason for routers and switches to use ECC ram, to avoid flipped bits

Reading

- EEEE Vinton Serf, a protocol for packet network intercommunication
- RFC791 # LT9 - IP Protocol 2

Ethernet Addresses

- Ethernet Addresses (MAC Addresses) are normally inherent to the machine
- 48 bits (6 bytes)
- 3 bytes identify manufacturer
- 3 bytes identify device
- Can be changed
- Good reasons: Older protocols modify address to match higher layer addresses
- Bad reasons: MAC spoofing to evade access control
- Debate on whether machines with two interfaces should have one or two addresses: standard says one, reality says two

Network Numbers/ DHCPv6

- Where to get an IP network?
- Small allocations from your ISP, belong to ISP (business ISP)
- If you change ISP you need to change your IP numbers
- Larger allocations can be obtained from **regional registries**
- Need a very good case to get these

Local IP numbers

- Static allocation
- bootp
- DHCP
- SLAAC (IPv6 only)

Static Allocation

- Endpoint given an IP in some sort of config file
- Every time device boots it gets exactly that IP, even if it's wrong or clashes etc.
- The machine might notice that it is a duplicate but otherwise has very little protection
- Essential for routers and infrastructure devices that need to be up in very early stages after power failure
- Often used for servers that need to have fixed addresses (www.my.domain, mail.my.domain etc.)

bootp

- Device broadcasts its ethernet address (MAC)
- "bootp server" hands back an IP number and DNS servers, default router etc.
- Can only be used for static assignments, as no means to reclaim addresses
- Large table statically mapping MAC addresses to IP numbers
- Obsolete

DHCP

- Dynamic Host Configuration Protocol (RFC2131)
- **Lease** a temporary IP number for a specified duration
- Also obtain address mappingss for routers, DNS, time servers etc.
- Can also be used to inform devices of a statically allocated IP number
- Main means of allocating IPv4 addresses on LANs in 2017

Handing out leases, a temporary IP number with a time, only allowed to use it for that period, after you have to give it back or renew the least. DHCP servers are much more complicated than bootp servers, as they have to handle an address pool.

Initial Operation

- *DHCPDISCOVER* - Client broadcasts a request for an IP number, including its MAC address
- *DHCPOFFER* - Server(s) reserve an available IP number, broadcasts an an offer of it with a lease time
- *DHCPREQUEST* - Client chooses from amongst offers, broadcasts reply containing chosen IP number
- If it matches the offer, the IP number is locked for the duration of the lease
- If the IP or MAC doesn't match they do nothing
- *DHCPACK* - Server that offered the IP finishes reserving it and acknowledges; other servers see their offer has been declined and unreserve their offer.

A client can directly request to a known server, in order to renew a lease. Conventionally, renewal attempts start after half the lease has passed. Leases typically last between 30s - 1 year

Static vs Pools

- DHCP can work like bootp, always handing out the same IP number for the same MAC address, to configure static machines
- Or it can manage a pool of temporary addresses
- Smart DHCP servers store the assignments so if you ask for an address you always get the same one from the pool, unless it has been allocated to someone else in the meantime
- Very smart DHCP servers can update DNS servers to record the name to IP binding

Redundancy

- Loss of DHCP server will wipe out your network
- But DHCP servers contain delicate state
- Simple home routes can be careless about this, so a route restart may require client device restart to get state into agreement, hence the typically short lease times
- Choices include:
- Two DHCP servers managing disjoint poolts, let client select TODO
  - One server lags the other by a second, only to be used in an emergency
- Complex failover protocol so both servers make the same offer and commit to the same lease (no standards, implementations are rare)
- Using higher-level redundancy protocols to have one virtual server (probably best)

Relaying TODO

DHCP relay agent totally stateless, one per network or more, can be buried in routers

Relaying

- Relay agent hears a broadcast packet
- Is configured to send all requests to a known DHCP server, after filling in its own address TODO