# FIT1047 - Week 9

Networks: Network and Transport layers

MONASH University



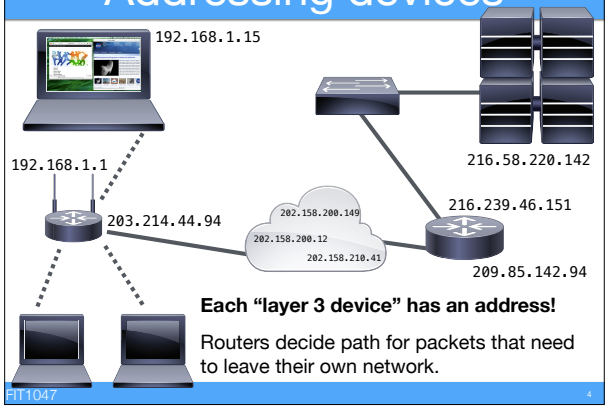
## Goals for this week

- See how **routers connect different networks**
- Understand how the transport layer makes sure messages **arrive correctly** and at the **right process**
- Study the **structure of the Internet**



# The Network Layer: **Adresses**

## Addressing devices




**Each "layer 3 device" has an address!**

Routers decide path for packets that need to leave their own network.



Remember: layer 3 = network layer. Layer 3 devices are clients, servers and routers.

Each network interface in each layer 3 device needs an IP address.

## IP version 4 addresses

**32 bit addresses**

- Written using "dotted decimal" notation
- Example: 130.194.66.43


**Hierarchy used for routing**

- You can immediately see if a destination address is in the same subnet!
- Subnet **mask**: **255.255.192.0 or /22**

no. of bits in network + subnet



**Hierarchy:** think of postal address (country, zip code, city, street, number, apartment number, name)

The "dotted" notation does not correspond directly to the hierarchy: the subnet (in this case) includes the first two bits of the 4th byte!

## Network Classes

**Subnet masks often expressed using dotted notation**

- e.g. 255.255.0.0 meaning /16

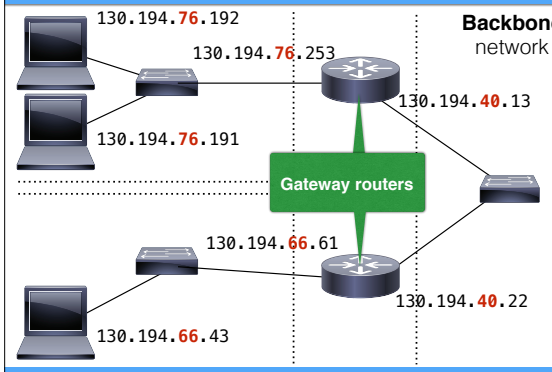
**Previously used hierarchy:**

- Class A: /8 (e.g. IBM, MIT, AT&T, Apple, …)
- Class B: /16 (e.g. Monash 130.194.0.0/16)
- Class C: /24

**Now: classless**

- e.g. /22, which can also be written as 255.255.252.0

## Subnets




FIT1047 7

Each device knows to which subnet it belongs. If a device needs to send a packet to an IP address that is outside of its own subnet, it must send it via its **gateway router** (so each device must know its gateway router!). A router is always connected to multiple subnets at the same time, and responsible for delivering the packet one step closer to the destination.

---

## IP version 6

**IPv4 addresses: 32 bits**

- In theory 4,294,467,295 addresses
- In practice probably only half are usable
- We've run out of new blocks of IPv4 addresses

**IPv6**

- "new" version of the IP protocol (from 1998…)
- 128 bits per address
- so that's four times more, right?

FIT1047 8

---

## IP version 6

$2^{128}=$

**340,282,366,920,938,463,463,374,607,431,768,211,456**

**(340 undecillion)**

**IPv6 addresses**

FIT1047 9

## IP version 6

**340,282,366,920,938,463,463,374,607,431,768,211,456**

**A bit excessive?**

- At least 7 addresses for every atom of every person on earth
- 665,570,793,348,866,943,898,599 addresses per square meter of the surface of the earth

**Required!**

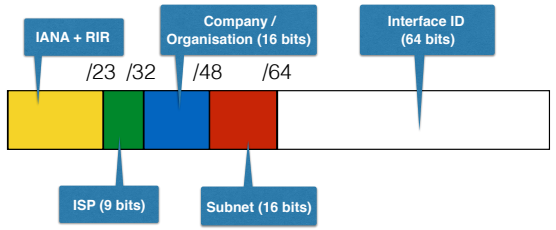
- The huge space is used to create **hierarchies**
- This makes it easy to assign whole subnets



The problem with IPv4 addresses was not only that there are not enough, but also that it's really difficult to assign hierarchical addresses (because we don't want to waste any addresses). With IPv6, we can simply give every company enough addresses to run as many subnets as they would ever want.

## IP version 6 address space

**Typical allocation:**




$2^{32}$ times more addresses than IPv4 in Interface ID alone!
Every company can run more than 65,000 different LANs



So a typical company would be allocated a **prefix** of 48 bits, and then it can use 16 bits to create subnets.
In practice many ISPs will allocate smaller packages to customers (e.g. only /60 instead of /48). But the main point is that there will be enough space for the future, even while wasting lots of addresses on creating hierarchies.

## IPv6 transition




See http://www.worldipv6launch.org/infographic/

There are other advantages to using IPv6, but the huge address space is the most important one.

# Address resolution

**Assume we browse to http://www.google.com.au**

- We have to translate www.google.com.au into an IP address: 216.58.220.99
- We send a request through the Internet to that IP address
- The router in the LAN of 216.58.220.99 needs to know the MAC address for 216.58.220.99 to deliver the frame

**This is known as address resolution.**



# Address resolution: Application Layer

**DNS (Domain Name System)**

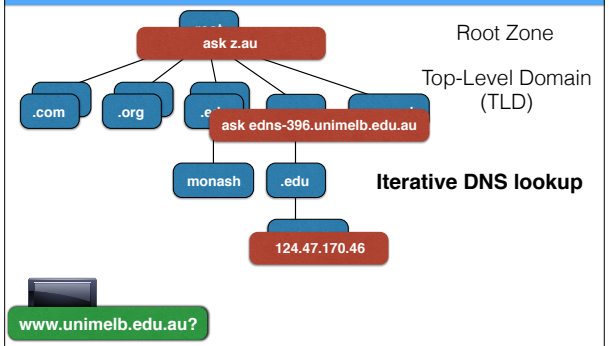
- Application layer protocol for address resolution
- Client sends request to DNS server to get IP address registered for a name

**DNS Servers**

- Implement a **distributed database** of names
- Are organised in a **hierarchy** reflecting the **structure** of the domain names

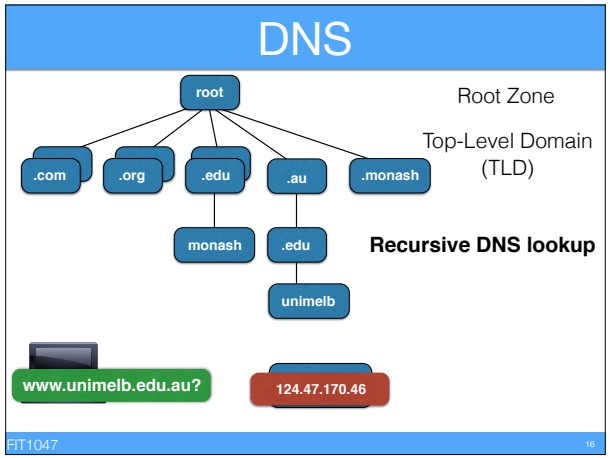


# DNS



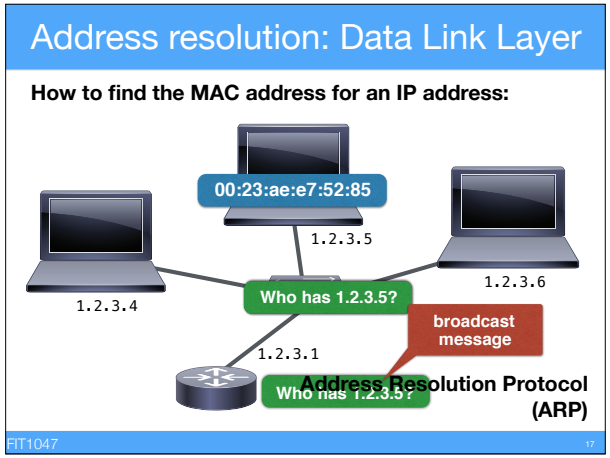



In iterative DNS lookup, the host starts asking the root zone server, which refers it to a slightly more specific TLD server, which in turn refers it to a slightly more specific server, which in turn may refer it to the server that actually knows the correct IP for the host name.




In recursive DNS lookups, the host asks the DNS server at its own ISP, which does the lookup on its behalf. The ISPs DNS server can then also *cache* the answer, so that the next time someone asks for the address, it can return it straight away.

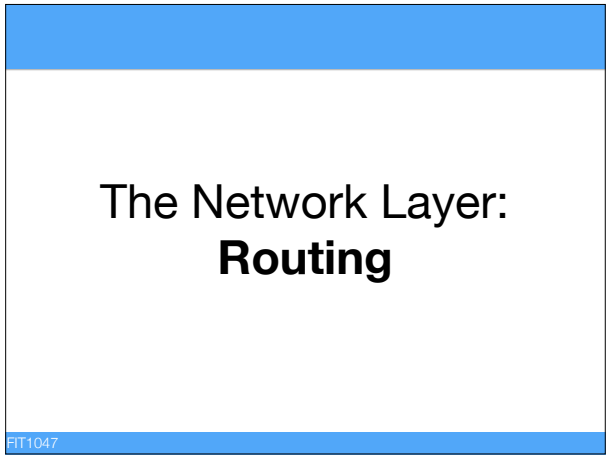
---




This is for IP version 4. For IPv6, a different protocol ("neighbor solicitation") is used.

Devices keep a **table** mapping IPs to MACs to avoid too many ARP messages.

---



# Routers

**Routers connect networks**

- Internet is a network of networks!
- Most important piece of Internet infrastructure

**A router is a layer 3 device**

- one IP address per **interface**, i.e. typically per subnet it is connected to
- Clients send packets to routers **if destination is outside their own subnet**
- Routers use IP address to determine **where the packet is sent next**



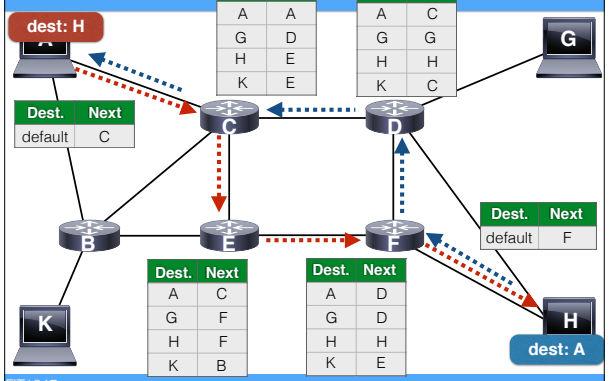
# Routing tables

**For each incoming packet, the router**

- looks at the packet's **destination IP address**
- consults the **routing table:**
  *to which other router should I send a packet for this destination, or can I deliver it directly?*
- if destination not in table: send to **default gateway**
- if no default gateway configured: **packet can't be routed**



# Routing example






**Many possible routes** from A to H: ACDFH, ACEFH, ACBEFH, ABCDH, …

Each router makes a **local** decision where to send the packet. The reply from H to A doesn't have to use the same path backwards!

How to make sure that routes are **efficient** and **don't contain loops**? That would be quite difficult if we had to set up all the routing tables by hand for a large network.

# Types of routing

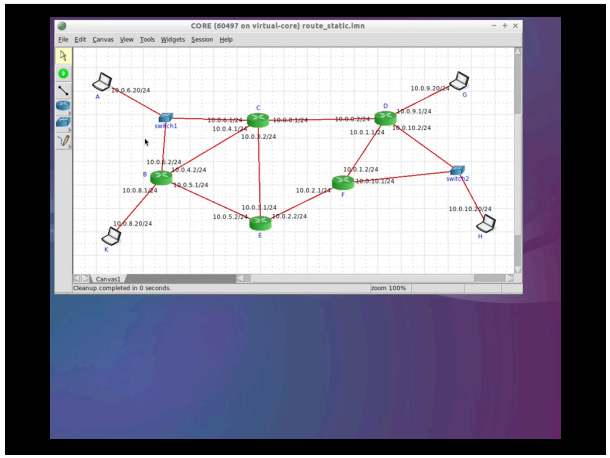
**Static routing**

- Network manager prepares **fixed routing tables**
- Manually updated when the network changes
- Used in simple networks that don't change a lot



# Static routing demo






The initial routing tables have been set up statically to route packets from K to G via B, C, and D. When we modify the static routing table in router B, and change the path of incoming packets for the 10.0.9.0. network to go via E, the traffic immediately takes that new route.

## Types of routing

**Static routing**

- Network manager prepares **fixed routing tables**
- Manually updated when the network changes
- Used in simple networks that don't change a lot

**Dynamic routing**

- Routers **exchange information** to build routing tables **dynamically**
- Initial tables can be set up by network managers



Problem with static routing: need to set up correct routing tables, and update **all routers** when the network changes.

## Dynamic routing algorithms

**Distance vector**

- Exchange information about **distance to destination**, choose **shortest route**
- EIGRP (Enhanced Interior Gateway Routing Protocol)
- RIP (Routing Information Protocol)
- BGP (Border Gateway Protocol)

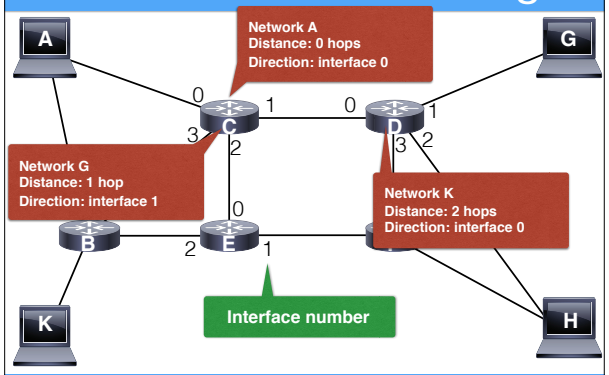
**Link state**

- Exchange information about **quality of links**, choose **fastest route**
- OSPF (Open Shortest Path First)



We'll only look at distance vector routing algorithms here in detail.

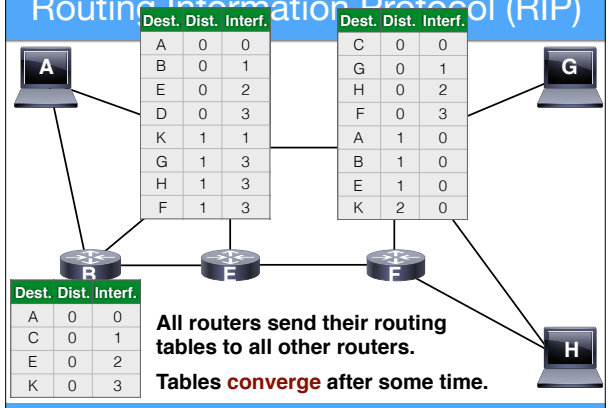
## Distance vector routing






In Distance Vector routing, each router knows, for each destination network, the **distance** (in number of hops) to reach it, and the **direction** (the vector) where it must send packets for that network (i.e., the interface). It then uses that information to build a routing table automatically.

## Routing Information Protocol (RIP)






The concept behind RIP is that routers can learn the distance and vector simply by exchanging routing tables with their neighbours. Each router incorporates the routing information from its neighbours for networks it doesn't know about yet, adding 1 to the distance (because there's an additional hop to the neighbour).

---

## Routing Information Protocol (RIP)

**Distance = hop count**

- Max. hop count 15
- Avoids loops

**Only used in small networks**

- At most 15 hops
- Updates transmit whole routing table
- Can be slow to converge



---

## Link-state routing

**Routers exchange information about connectivity**

- not just routing table (best routes)
- but **all** of the network it knows about
- use a **metric** (usually link speed) to describe the **quality** of each connection
- each router can **independently** compute best route to every subnet

**Most common protocol: OSPF**
(video available on Moodle in case you're interested)

# Transport Layer



## Transmission Control Protocol (TCP)

**Connection-oriented**

- A **virtual circuit** is established between two devices
- To the application it always looks like a **point-to-point full duplex** connection
- Messages split into **segments** for transmission

**Reliable**

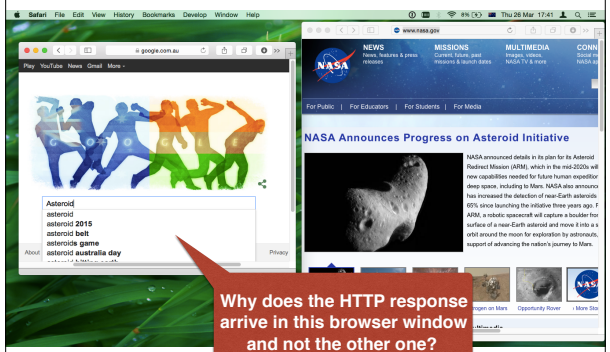
- Errors are **detected** and **corrected**
- Segments are re-assembled in the **correct order**

**Used by e.g. HTTP, SMTP, IMAP, SSH**



Segmentation: application can send long message (e.g. a video), but data link layer frame size is limited (e.g. 1500 for Ethernet). TCP splits message into segments and reassembles at the receiver.

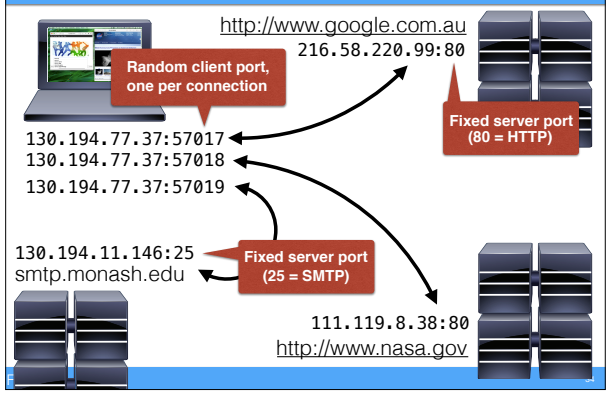
## Addressing applications






Since many applications (processes) are running on the same machine, the TCP/IP system needs a way of sending each packet to the correct process. IP addresses are not good enough: they only identify an *interface* (and therefore one device), but not a *process* on that device.

## Addressing applications




The **port number** (or port address) is used to address an individual application (or process).

Server processes use **standard port numbers** so that clients know which port to connect to (e.g. port 80 stands for http, port 25 for SMTP).

Clients create **random port numbers** for each connection (which can be reused when the connection is closed). A connection is therefore **uniquely identified** by four numbers: source IP, source port, destination IP, destination

---

## One address per layer

**Application Layer**

- URL (e.g. http://www.csse.monash.edu)

**Transport Layer (TCP)**

- Port number (e.g. 80 for HTTP)
- identifies the application that handles a message

**Network Layer (IP)**

- IP address (e.g. 130.194.66.43)
- used for identifying devices across networks

**Data Link Layer (Ethernet)**

- MAC address (e.g. 00:23:ae:e7:52:85)
- used for sending frames in a LAN



This is just to summarise the addresses at the different layers. Make sure you understand their different functions.

---

## TCP ARQ

**Error control**

- Data Link Layer **discards** frames that have errors
- Frames may not arrive at all
- But TCP should be a **reliable channel!**

**Solution:** **A**utomatic **R**epeat Re**Q**uest

- Exchange **acknowledgements (ACK)**, letting sender know that packets were received correctly
- Sender re-transmits if no ACK within certain time

## TCP ARQ

**When we send a TCP packet, it includes two numbers.**

**Sequence number:**

- how many bytes we've already transmitted (before this one)

**Acknowledgement number:**

- how many bytes we've received from the other side

**Sender can therefore check how many bytes have been received correctly!**



Note that TCP is symmetric and "full-duplex": both parties can send and receive. Think of a typical application like HTTP. The client needs to send a request, and the server then sends the response. Both client and server acknowledge each others transmissions to make sure nothing gets lost.

## Establishing a connection

**Three-way handshake:**

- Client sends a **SYN** packet with **random sequence number *A***
- Server replies with **SYN, ACK, acknowledgement number *A+1,*** and **random sequence number *B***
- Client sends **ACK** with **sequence number *A+1*** and **acknowledgement number *B+1***



After the three-way handshake, the server and client both know each other's random sequence number. That way they now have shared knowledge and can start transmitting the actual data.

## Closing a connection

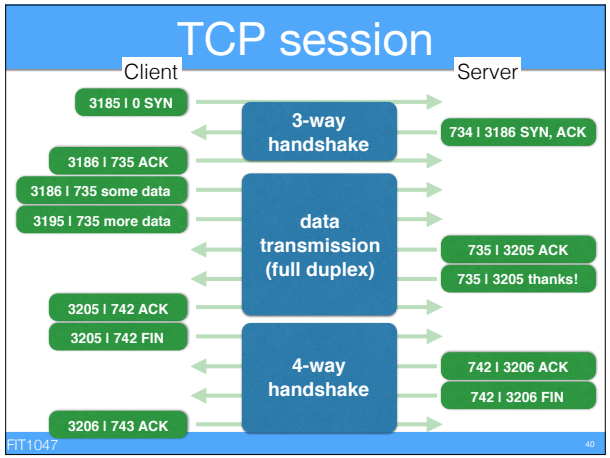
**Four-way handshake:**

- Computer A (client or server!) sends a **FIN** packet
- Computer B acknowledges with an **ACK**
- Computer B sends a **FIN** packet
- Computer A acknowledges with an **ACK**
- **Can be simplified to three-way** (combining a FIN/ACK)

**Necessary because TCP is full duplex!**



We don't know in advance whether e.g. the client or the server is finished sending data first, so both need to be able to close the connection individually. If computer A sends FIN first, it needs to wait until computer B also sends a FIN.




Sequence is always previous sequence number + number of bytes transmitted

Eg 3186 + "some data" = 3195, 3195+"more data"=3204

So first server ACK number is 3205 (next byte **expected** to be sent)

Wireshark shows **relative** sequence/ack numbers (i.e. subtracts initial random number) to make them easier to read

---

## TCP parameters

**TCP implements segmentation**

- large application layer messages are split into segments

**How fast to send segments?**

- Sending too many at once may overload receiver or intermediate path with lower bandwidth

**How to decide the segment size?**

- Sending too large segments requires IP to **fragment**
- Large segments also increase errors

FIT1047 41

E.g. when you download a large image or movie, it is split up into segments of (typically) around 1500 bytes each. Remember that an Ethernet frame can only be a bit more than 1500 bytes, so we're trying to fit a segment into a single frame.

The network layer (below TCP) actually has to check if one segment (or packet) fits within a frame, because it talks directly to the data link layer. If the packet doesn't fit, it needs to be split up again (this is called

---

## TCP segment size

**Two approaches:**

- use “reasonable” MTU, accept that IP sometimes needs to fragment
- use **Path MTU Discovery** (PMTUD):
  - send IP packets to destination, asking routers to **never** fragment
  - if router would have to fragment, it sends back an error message
  - increase packet length until error occurs, then use last known error-free MTU

FIT1047 42

# TCP congestion control

**How fast to send?**

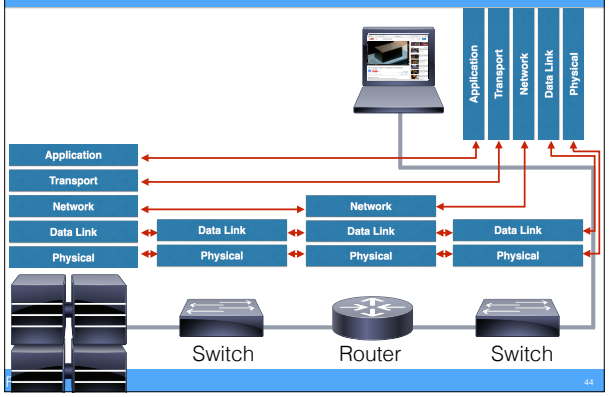
- Receiver transmits its maximum buffer size
- Sender sends segments without waiting for ACK up to buffer size

**What if network cannot cope?**

- Start slow: wait for ACK after each segment
- Increase with every ACK: send two, four etc segments after each ACK
- Fall back to slower speed when no ACK arrives



# Layers of Abstraction



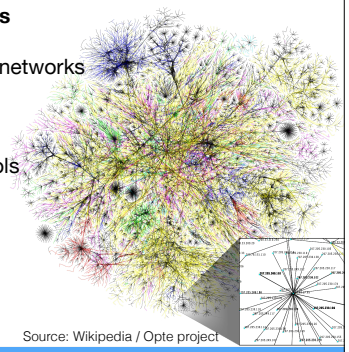



We've now seen the entire TCP/IP "stack", and the basic networking technology behind the Internet.

# The Internet

# The Internet

- A **network of networks**
- Connecting millions of networks and billions of devices
- Based on a common, standard set of protocols



Source: Wikipedia / Opte project



Recall: The Internet is nothing special in terms of the networking technology: every lab at Monash, e.g., is not *connected* to the Internet, it is *part* of the Internet! We'll see how this works in the coming two weeks.

# Autonomous Systems

**Networks operated by a single organisation**

- e.g. Monash University's or your ISP's network

**Interior routing**

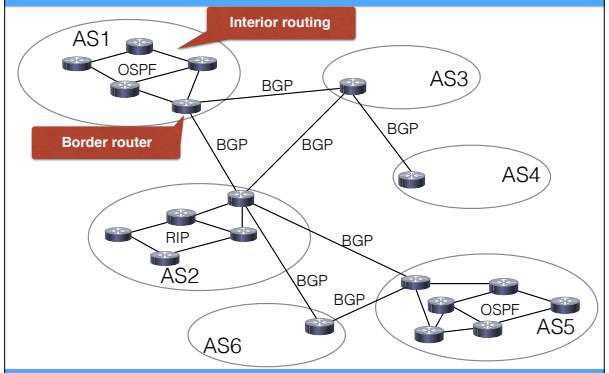
- for routing packets **within** an AS
- uses RIP, OSPF, EIGRP

**Exterior routing**

- for routing packets **between** different AS
- Internet uses BGP (Border Gateway Protocol)



# Internet Architecture



# Internet Structure

**Internet Service Providers**

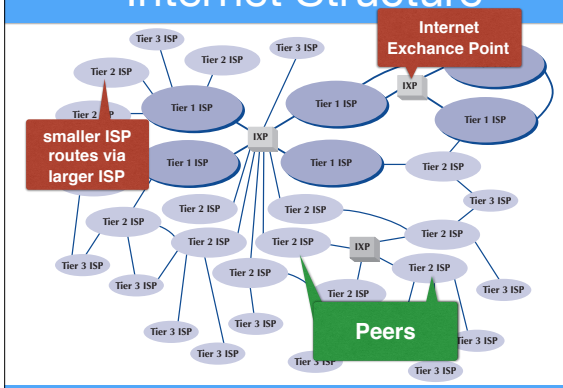
- Commercial companies
- You connect via ADSL, 4G, cable, NBN, …
- How are they connected with each other?

**Hierarchy of ASs**

- Each ISP operates an AS
- Routing information shared between ASs using BGP
- ISPs connect at **IXPs**



# Internet Structure






Internet Service Providers (ISPs) connect with each other at Internet Exchange Points (IXPs).

# Peering

**Tier-1 ISPs**

- Large ISPs with large WANs
- Charge smaller ISPs for routing their traffic

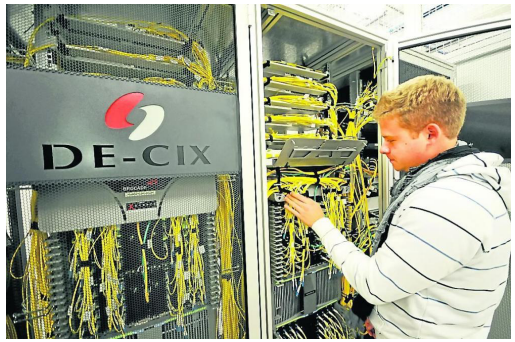
**Peering agreement between two ISPs**

- accept each other's traffic without charge
- usually because both are similar size (similar amount of traffic)
- connect at an IXP

**IXP**

- provides the hardware for several ISPs to connect
- often owned by a consortium of ISPs

## Inside an IXP





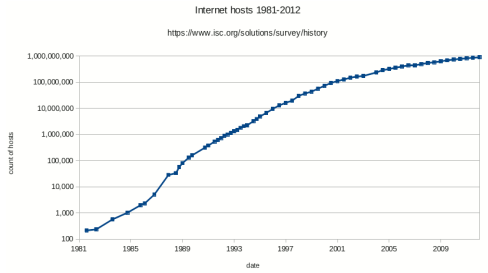

Similar to Monash switching racks, but using fibre optic connections instead of UTP

## Delivering Content over the Internet



## The problem

**The Internet has grown**



## The problem

**The Internet has grown**

- not only the number of hosts, but also their **distance**
- many applications rely on **low latency** (video, streaming music, modern web sites)
- some services have millions of users
- the main protocols (HTTP, TCP/IP and BGP) were not designed for this growth

**So why does it still work?**

- Load balancing
- Content Delivery Networks (CDNs)



## Load Balancing

**Many services impossible for single server**

- Google processes over 40,000 search queries per second
- 6000 tweets are sent per second
- Netflix streams around 10.2 Tbps on average

**Spread load over multiple servers**

- DNS-based: host name maps to multiple IPs
- Special hardware: load balancer accepts requests, routes them to different servers



http://www.internetlivestats.com/google-search-statistics/

http://www.internetlivestats.com/twitter-statistics/

According to Netflix, 10billion hours watched in Q1 2015 (at 1GByte/h), 10Gh/90d/24h/60m/60s = 1,286 h/s = 1.286 TByte/s = 10.2 Tbps

## DNS-based load balancing

**Inside Monash network:**

- PING www.google.com (216.58.220.132)
  64 bytes from 216.58.220.132: time=13.752 ms

**From Optus network:**

- PING www.google.com (74.125.237.209)
  64 bytes from 74.125.237.209: time=52.074 ms

**From Germany:**

- PING www.google.com (173.194.112.176)
  64 bytes from 173.194.112.176: time=1.43 ms

**From France:**

- PING www.google.com (74.125.21.105)
  64 bytes from 74.125.21.105: time=104 ms



As you can see, DNS servers will return very different IP addresses for the same application layer address (the host name www.google.com), depending on where you are located. But even from the same location the addresses could differ every time you ask. That way your request will hit a server that's close to you (and therefore faster), and requests from different users will be directed to different servers.

## Content Caching

**Store web data closer to users**

- replicate web pages etc. in *caches*
- can be implemented *transparently:*
  - user makes request
  - router on path to server queries *cache engine*
  - if content available, serve from local cache

**This is explicitly supported by HTTP**

- GET requests can be cached
- HTTP headers can contain `Expires:` field
- Cache only serves GET requests that are not expired



Caching works particularly well for static content like css, javascript and images

## Content Delivery Networks (CDNs)

**Load balancing only solves half the problem**

- once the requests arrive in your network, you can distribute them to all your servers
- but the requests and responses need to be routed through the Internet

**CDNs**

- operate servers in multiple locations
- operate their own high-bandwidth network
- locate points of presence close to end users



CDNs are run by independent commercial companies, but also by telecommunications companies, and sell their services to other companies that need to deliver large amounts of data to end users.

## CDNs and Peering

**Get close to your customers**

- improves user experience (fast page load times)
- network inefficiencies are not blamed on you

**CDNs are present at IXPs**

- peer with anybody for free
- small ISPs avoid paying e.g. for YouTube content downloaded from the upstream ISP
- Example: Netflix peers with Australian ISPs, which can offer “unmetered” access



https://peering.google.com/about/

https://peering.google.com/about/delivery_ecosystem.html

https://openconnect.netflix.com

# Summary (I)

**Network layer**

- subnet mask identifies other devices in same subnet
- routers use routing tables to determine next hop
- static vs dynamic routing
- IPv4 vs IPv6

**Transport layer**

- splits up long messages into segments
- uses ARQ for error control
- uses ports to identify applications



# Summary (II)

**The Internet**

- consists of Autonomous Systems
- connected by routers that use BGP for sharing routes
- ISPs run autonomous systems, connect at IXP
- Load balancing and Content Delivery Networks enabled the massive growth