

# Networking

# Autumn 2017

[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# Check Panopto!

- Is it running?
- Is it running?
- Seriously, is it running?

# What this course is

- Some networking **theory**
- Some networking **practice**
- Some networking **business and reality**

# What you will learn

- The main components of networking 2017-style, hopefully with an eye to 2027 (or at least 2018).
- Some background to explain why networking today looks like it does, much of which is rather contingent, and how people make money.
- Some theory and science to justify the technology.

# The key question

- Both GCHQ and Google lament that when they ask applicants the question “you click on a link, what happens?”, people can’t really give much of an answer beyond the GUI.
- This course should get you from the click to the wire.

# Who are you?

- I'm expecting people from
  - 3rd/4th years of our BSc/MSci/MEng programmes and parallel joint honours programmes
  - Advanced Computer Science and Cyber Security MSc programmes
  - Is there anyone else? ICY? Conversion MSc?

# Programming Skill

- I'd really like you to be able to program, well, in C (ie, Hayo's course, and perhaps be taking Operating Systems in parallel with this)
- However, for Cyber Security Students, I want you to be able to take this course as a pre-requisite for Network Security, which is compulsory, and you might not have C skills.
  - In which case, we can do the 10% programming exercise in other languages, but you might not get the full experience.

# Extended Course

- There will be an additional section in the second exercise towards the end of the course for “extended” students, which will look at deeper issues.

# Some Basic Assumptions

- Spoiler: Apollo 13 did get back to earth
- Spoiler: The Titanic sinks before the end of the film
- Spoiler: Ethernet lower layers and TCP/IP upper layers **have won**. There are no serious contenders on the 10–20 year horizon. Everything else is history or weird curiosity.
- So this course is unashamedly Ethernet and TCP/IP focused, as there are no other games in town.

# Week 1:

- This Introduction
- Packet v Circuit Switching, layer models (DoD 4/5, ISO 7), what's in the subnet, transport, application layer.
- Network Hardware: Switches, Routers, data/control/management plane. Software defined networks.

# Week 2

- Socket Concepts, Threading requirements, old code forks. Socket API.
  - I really want to do this in C, with Java as a distant second choice. Please talk to me if this is going to be a problem for you. There will be an exercise.
- Application: DNS
  - Worth a whole lecture, as it's a critical Internet service. I will talk about practical deployment and some security issues as well as how the protocol works.

# Week 3

- Background: LAN/WAN split, Arpanet, X.25, PSS, DEC/XNS/SNA, DoD, OSI. Why OSI Failed.
- Lower Layers: FDDI, Slotted/Token Rings, **Ethernet** (in its various forms). Touch on Transmission (SDH, WDM). ATM in passing.

# Week 4

- IP: addressing, routing, concepts. **Why IPv6 is needed.**
- IP: address allocation, bootp, DHCP, SLAAC.

# Inversion

- In the past few years, I taught the knotty details of TCP at this point.
- Strong knowledge of TCP is important and valuable: there's a real shortage, 35 years after the RFCs, 40 years after Cerf's paper, of people with good understanding of its dynamics
- But I think it needs applications to motivate it, so I'm going to switch to applications at this point and come back down the stack.

# Week 5

- Application: HTTP, SMTP, IMAP, POP3, SNMP, NTP
  - For all the application lectures, I will deal with deployment and server issues as well as the protocol, so you can go out and do stuff with them.
  - For the good of everyone's futures, I'm going to spend some time on security issues within these protocols.

# Week 6

- TCP 1: history, basic concepts. Windows, acks.
  - Previously I had a deadline here to meet Network Security, which I was teaching in the same semester. We're more relaxed this time.
- TCP 2: Detailed operation
  - This might take more than one lecture

# Week 7

- TCP 3: Options (scaling, PAWS, etc, timestamps)
- TCP 4: Nagle, silly-window.
- Implementation: how it looks in the kernel.
  - I'm going to assume some familiarity with Unix/Linux kernels, but I'll try to make it accessible if you don't have that

# Week 8

- UDP, RTSP, other transports
- NAT and its evils. IPv6 as cure. IoT. NAT security / logging / problems.

# Week 9

- Application: Voice (I will try to get Chris Gallon in), 21CN, issues and politics.
- If I can't get Chris, Jim Reid on DNS security, issues and politics
  - Marshall Rose referred to politics as “Layer 8” of the 7 layer model.
- Tutorial, Catchup, Exercise feedback and discussion

# Week 10

- Routing inside the enterprise: Interior (RIP, OSPF, IPv6 analogues). VLANs.
- Routing outside the enterprise: Exterior (BGP, tech and politics). PPPoA/E, VLAN stacking

# Week 11

- Summary and spare

# Assessment

- Exercise set w/c 2/10/2017, due 23/10/2017
  - Will be a programming task, including testing against other people's code for interoperability and bugs
- Exercise set w/c 6/11/2017, due 27/11/2017
  - Will **not** be a programming task, but a bit of shell-scripting might help it along.
- Each worth 10% of total marks, so don't sweat them too much.

# Office Hours

- Wednesdays, 10–12 in Room 132
- [I.G.Batten@bham.ac.uk](mailto:I.G.Batten@bham.ac.uk)
  - Or Canvas discussions
- <https://igb.batten.eu.org/>
- Canvas/Panopto will (I hope) contain full recordings

# Books

- “Distributed Systems: Concepts and Designs” by Coulouris, Dollimore, Kindberg and Blair is also used for Distributed Systems and covers a lot (was used in the past when the two courses were combined)
- TCP/IP Illustrated, Volume 1 by W. Richard Stevens is the essential book on TCP/IP
- TCP/IP Illustrated, Volume 2 is **not** necessary (it documents kernel implementations) but is a fascinating read (really)
- I will also expect you to read RFCs as we go along.

# Things I've left out

- There is hopefully some spare time in which we can fit in some of the following.
  - Wireless networking (802.11[abgn])
  - Network management (SNMP in more detail)
  - Network design issues

# Networking 2: Models, Methods and Metal

I.G.Batten@bham.ac.uk

<https://www.batten.eu.org/~igb>

# Check Panopto!

- Is it running?
- Is it running?
- Seriously, is it running?

# Contents

- Circuit Switching v Packet Switching
  - Netheads v Bellheads
- Layered Models
- Transport and Application Layers

# What do we want to do?

- We want to link network elements together so they can exchange data:
  - In infinite amounts
  - Infinitely quickly
  - With zero error rates
- With that, there is no need to make engineering trade-offs, and we can all go home

# Reality Called

- Just as hi-fi nerds talk about “straight wires with gain”, but then accept they can’t have it, we have to accept that data travels at finite speeds through networks of finite capacity, and errors happen.
- We can characterise a flow of data in terms of:
  - volume per second (bits per second)
  - latency (seconds)
  - error rate (errors per bit)
- With suitably large error bars on these numbers

# Volume

- Bandwidth, capacity, etc
- How many bits per second can I put in one end and have emerge from the other

# Latency

- How long does it take for a particular bit to travel through the network (which may be defined to include parts of the end stations, so we might be more interested in application-to-application latency).
  - The speed of light imposes a lower bound, a lower bound high enough we will have to worry about it.

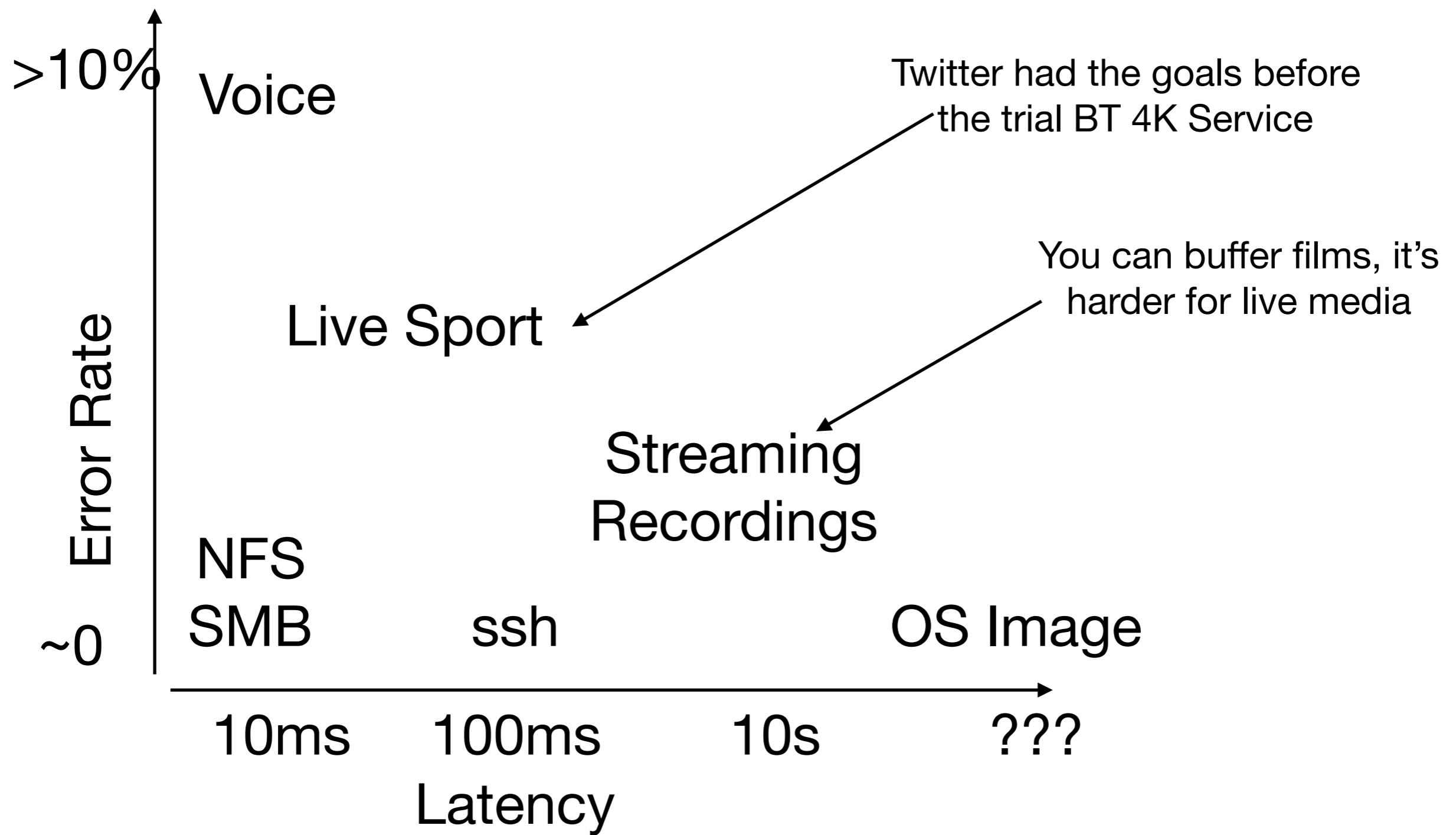
# Error Rate

- Will be a property of the medium, technology and potential interference
  - Data loss is an error
  - Cosmic Ray, mains impulse noise, cable and connector issues, etc, etc.
  - Determines how much error checking and correction we can justify, and ultimately there will be an uncorrected error rate we have to accept.

# Different Data: Different Requirements

- Voice can accept low bandwidth and high errors, but latency is a problem
- File Transfer of operating system images is about reliability above everything else
- You can look at similar trade-offs for different types of data

# Trade Offs



# Over time

- Link bandwidth tends to infinity (terabits per second is trivial with DWDM), but is still limited by cost
- Latency is affected both by processing times, which decrease, but also the speed of light which is always there

So what is the effect of the speed of light on a link between London and Birmingham? London and California?  $c = 3 \times 10^8 \text{ ms}^{-1}$ .

- Raw error rates remain about the same, but we have more processing power and spare bandwidth so can do better error correction
- So let's look at some hardware.

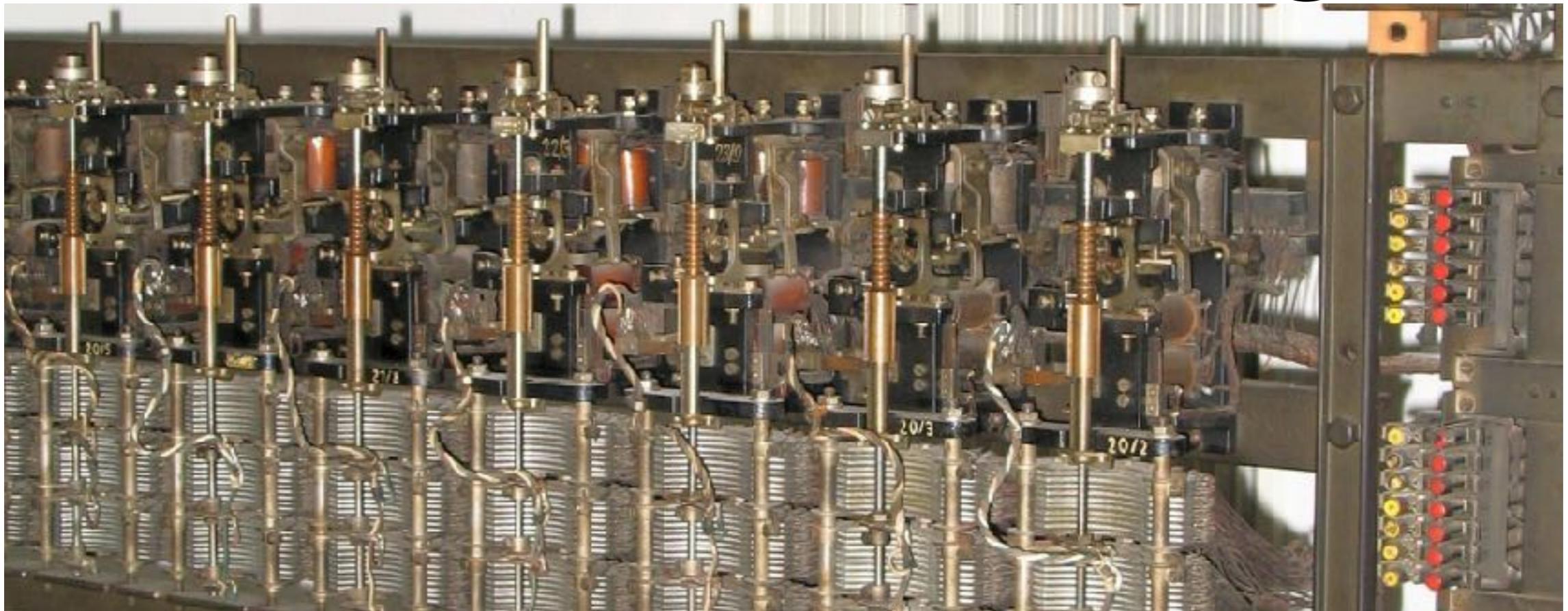
# “Never underestimate the bandwidth of a jumbo full of tape”

- Standard 2.5" SSD laptop drive is  $10 \times 7 \times 0.9$  cm, so volume is  $63\text{cm}^3$ , weighs 130g.
- $1\text{m}^3$  will therefore hold 15800 such drives, total weight 1.6t
- Revenue payload of a 747 is  $\sim 300\text{t}$ ,  $\sim 500\text{m}^3$ , so we are weight limited to  $180\text{m}^3$  of disk drives.  $180 \times 15800 = 2.8 \times 10^6$ . Recall, 1TB is  $10^{12}$  bytes.
- $2.8 \text{ million} \times 1\text{TB} = 2800\text{PB}$ ,  $2.8 \times 10^{18}$  bytes. Over an 8 hour flight,  $2.8 \times 10^5$  seconds, that's  $\sim 10^{13}$  bytes per second,  $\sim 10^{14}$  bits per second.  $\sim 100\text{Tbits}$  per second is pretty fast: it's the whole UK/US bandwidth, and then some. 1t, 1 hour =  $\sim 2.5\text{Tbps}$ , which makes an estate car to London pretty handy for backups.
- Latency not great for playing Call of Young People's games.
- Check my maths...all those zeros.
- And of course, most of the weight is the casing. If we did this with M.2 PCIe drives, the numbers would be larger.

# But...

- Waiting for the plane to come isn't great for anything
- You could have a conversation by exchanging tapes of voice, or tapes of data, but it isn't going to be quick.
- So doing it electrically is useful
- Let's skip over a manual plugboard exchange, and move straight to...

# Circuit Switching



- Old telephones: exchanges linked physical wires together so that the microphone at one end was continuously connected to the speaker at the other end, via suitable amps and multiplexors

# Problems

- Very inefficient: ties up a duplex circuit even when one or both parties are quiescent
- Multiplexing is very complicated and expensive using 1950s technology
  - You have to treat the wire as a radio and use multiple carriers, which is hard to do reliably

# Packet Switching

- Proposed independently by Paul Baran in the USA (RAND) and Donald Davies in the UK (NPL).
  - Davies had worked as an assistant to Klaus Fuchs at Birmingham on Tube Alloys
  - Baran was working on survivable architectures for post-nuclear communications



# Packet Switching

- Divide the data stream up into small units, called “packets”.
- Give each packet some identifying information
- Switch the packets over your network until they get to the destination
  - “Switch” is American railroad usage for what are “points” or “turnouts” in UK railway usage.

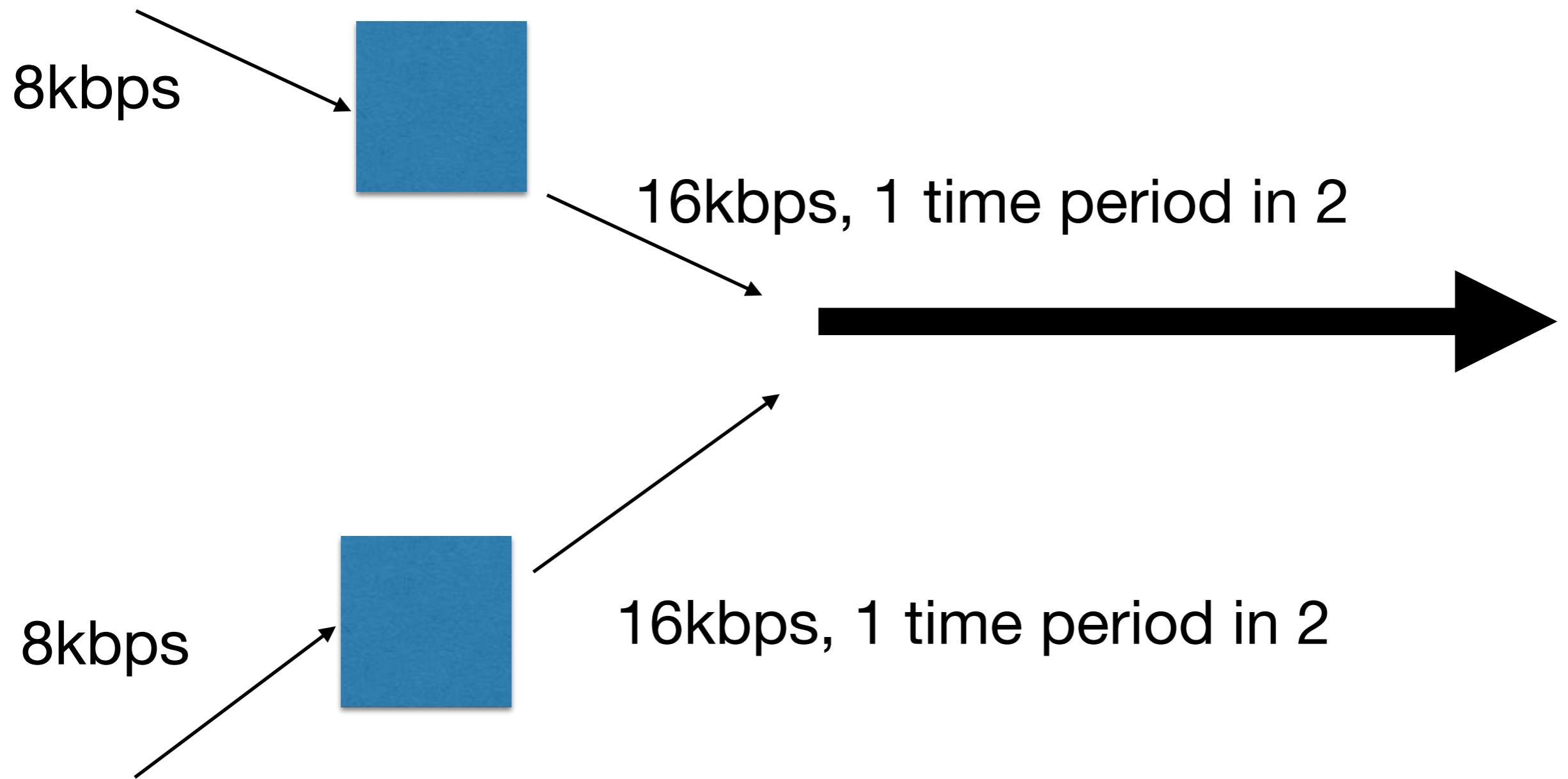
# Advantages

- You are multiplexing in the **time** domain, rather than multiplexing in the **frequency** domain. It's easier to build, in general.
  - You delay the packet until there is room on the line, introducing **latency**.
- When no data is being sent, no (or at least few) resources are consumed.
- You therefore get **statistical gain** on bandwidth.
- You can re-route data around failed switches, for resilience.

# Time Domain

- Suppose I have some data streams, each of 8kbps (8000 bits per second).
- I have a line which runs at 16000 bits per second
- I can send them all by buffering data arriving at 8kbps, then when the buffer is full sending it on at 16 kbps (the output will be idle 50% of the time).
- I can do this a second time, using the other 50% of the line.
- There will be some latency: the first bit in each block is delayed by the time taken until the last bit arrives, and then the time taken to send the block at the higher speed.

# Time Domain



# Contrast, Frequency Domain

- I can distinguish two signals on a medium by their frequency.
  - Low and high pitched sound in air, red and blue light in fibre, 96MHz and 98MHz radio signals in a vacuum or on a wire.
  - I can modulate that signal (or “carrier”) with some data by varying the amplitude (“AM”), the frequency (“FM”), the width of some pulses (“PWM”).
  - On the receiving side I separate the two carriers, and then decode the modulation.

# Frequency Domain



- Imagine two lighthouses flashing messages in morse, one in red, one in green, close enough you can't visually separate the lights.
- You would be able to distinguish the two different messages by using appropriate filters on your telescope.

# FDM v TDM

- Doesn't introduce systematic latency (which is why, amongst other reasons, Radio 4 on FM runs ahead of Radio 4 on DAB).
- But is harder to engineer and, for physics reasons ("sidebands", if you're interested), the use of available capacity is less efficient with real signals.
- All the networking we'll be talking about, apart from at the transmission level, is time-domain.

# Statistical Gain

- Not many people hammer full line rate 24x7: people pause to sleep, catch breath, etc.
- Even hammering at full line rate is limited by the other end and other network delays.
- So you can sell 100 people 10Mbps each, and only provision  $(100-x)\%$  of  $(100 \times 10M)$ ,  $x$  in  $[0, 100]$ .
- Typically, for residential,  $x$  might be as high as 95 (ie, you only provision 5% of the aggregate bandwidth), and has been as high as 98 (2%, 1:50).
- Also called “contention ratio”, “overbooking”, “over commitment”.

# Problems

- Packets can get lost, delayed, re-ordered.
- You (usually) have to break data up into packets and stick them together when they arrive

# Presentation

- The carrier network, the service you buy in to convey your data, can offer a variety of services.
- Let's simplify wildly and look just at **virtual circuits** and **datagrams**.
  - Amusingly, the telegram, from which datagrams take their name, hasn't been available commercially for decades. I have never seen or sent one, and I don't think my parents (born 1935) have either.

# Virtual Circuits

- Endpoints tell the network to establish a connection
- The network sets up a path to the destination, and gives back some token to identify it
- For the duration, that token identifies a “virtual circuit” linking two endpoints
- It is then torn down when the endpoint has finished with it
- Network has to know about every connection in progress
- Each packet in a connection follows the same route
- Network tries to sort out ordering and packet loss/duplication, but doesn’t always guarantee it
- The user of the circuit doesn’t have to worry about the fine details, but a checksum would be a good idea once in a while.

# Datagram Services

- Each packet contains complete addressing information
- Each packet is considered as a separate item by the network (conceptually, at least: more later)
- Endpoints are responsible for dealing with issues of loss, duplication, corruption: your packet might get delivered at some point, that is all you know.
- Network doesn't (need to) know about connections: it just routes packets

# Netheads v Bellheads

- “Bellheads” (people who work for telcos) like virtual circuits: they can shape and groom traffic, provide added value, run complex and interesting protocols
- “Netheads” (people who go to IETF meetings) like datagram services: it stops the telco from shaping, grooming...
- Last thirty years are a history of conflict between the two camps.

# Layering

- What I've just described is the very basic services made available by networks: virtual circuits and datagrams.
- But applications generally want something with guarantees: you send a file, it gets there undamaged or you know something went wrong.
- And you want your programs to be able to operate over different sorts of network without radical changes.

# Layering

- So the idea arises of thinking of a network in terms of “layers” or “a stack”: a succession of interfaces which start with the services needed by real applications, and progressively get closer and closer to volts and flashing laser diodes.
- Each layer provides services to those above it, and makes use of services from those below it. And each task only appears in one layer.

# Layers

Layer  
 $n+1$

More Abstract Service, closer to Application

*Like catch/throw*

Request

Solicited  
Response

Unsolicited  
Response

Layer  
 $n$

Less Abstract Service, closer to Wire

Anything that obeys the requests can  
be used as a replacement

# Competing Models

- “OSI” (Open Systems Interconnect) was a failed project, mostly European, to build a standard suite of networking protocols from within the telecommunications community. We will look later at why it failed. It was competing with proprietary systems and with...
- ... the DoD, aka ARPA, aka TCP/IP suite that we will be using as our main case study

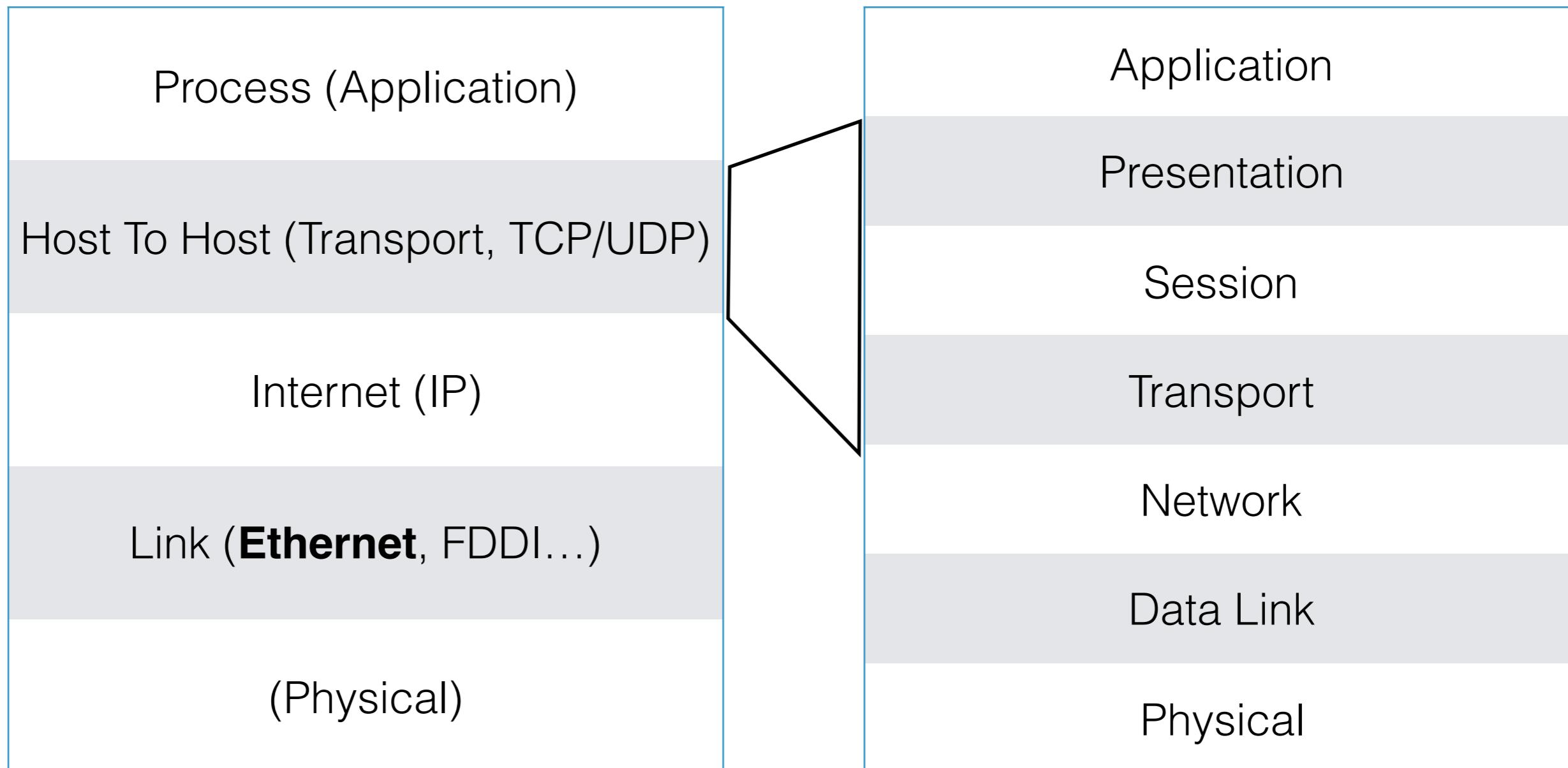
# OSI Model

- The model came long before any successful implementation.
  - A ~~cynic~~ historian would argue there were never any successful implementations.
- But the model achieved widespread traction as the model of how computer networks either *should* or *do* work.

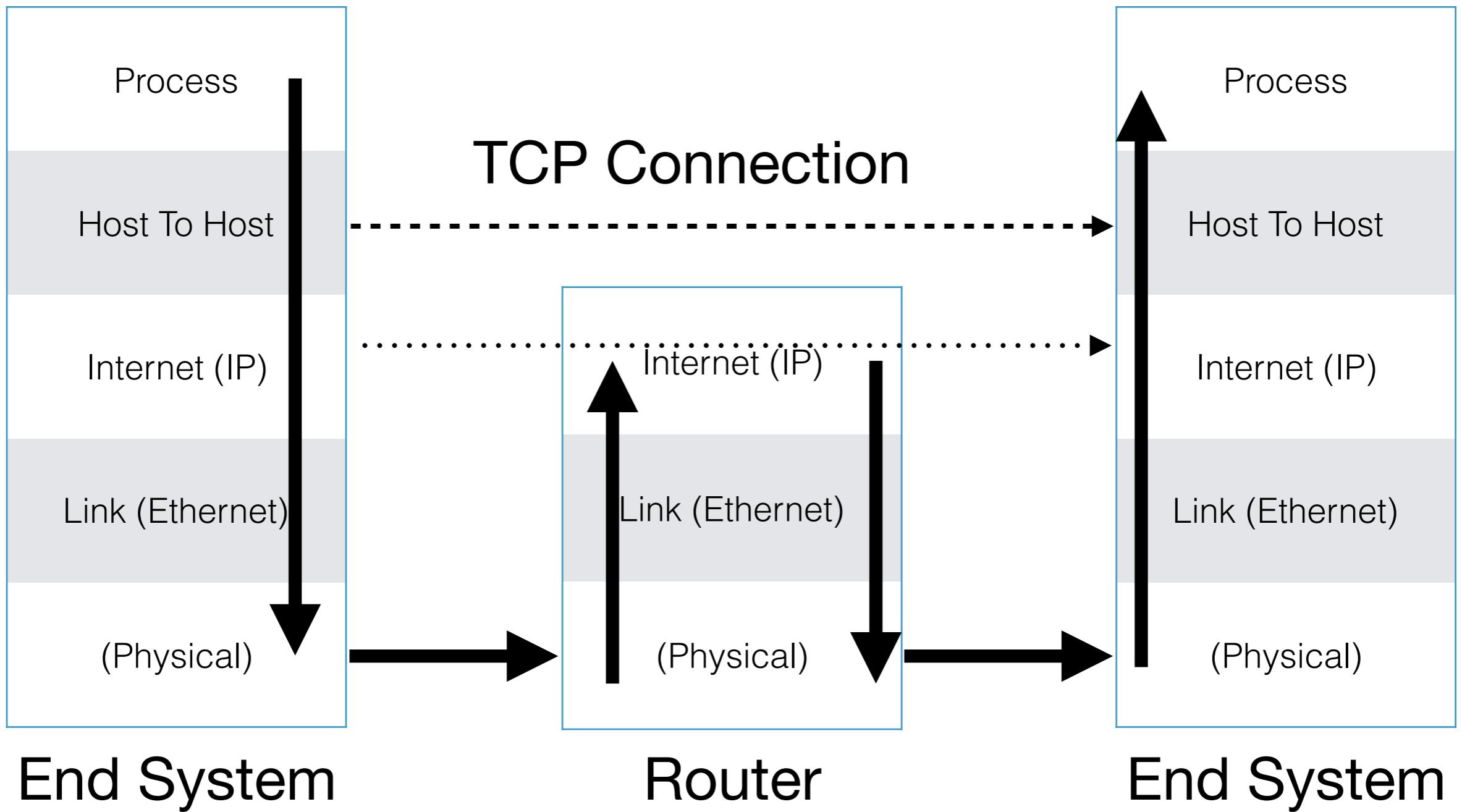
# DoD Model

- The DoD model is a post-hoc rationalisation of established practice, in part in order to provide a way to compare the TCP/IP architecture with the OSI proposals.
- Practice and implementation came first, the model long afterwards.

# DoD v OSI



# The DoD Model



# The DoD Model: Applications

- Applications are running code that do real, useful work, and the protocols that they use.
  - SMTP and IMAP for mail, HTTP for web, ssh for remote logon...
- They need services to move data from computer to computer.

# The DoD Model: Transport

- A transport layer moves data between two end systems via a network whose topology and other properties the transport layer doesn't know much about.
  - TCP for streams of data, UDP for packets
- The transport layer will guarantee various properties: reliability, sequenced delivery, etc, etc (or will disclaim responsibility for these things).
- The transport layer will permit communication between multiple entities (applications, usually) running on the same end systems.

# The DoD Model: Internet / Subnet / Network Layer

- This layer moves packets between end systems, but doesn't offer any guarantees about reliability. It handles choosing which link to use to get the data closer to its destination.
- This layer also doesn't deal with any concept of multiple applications: separating the data between two different applications is the layer above's responsibility
  - IPv4, IPv6

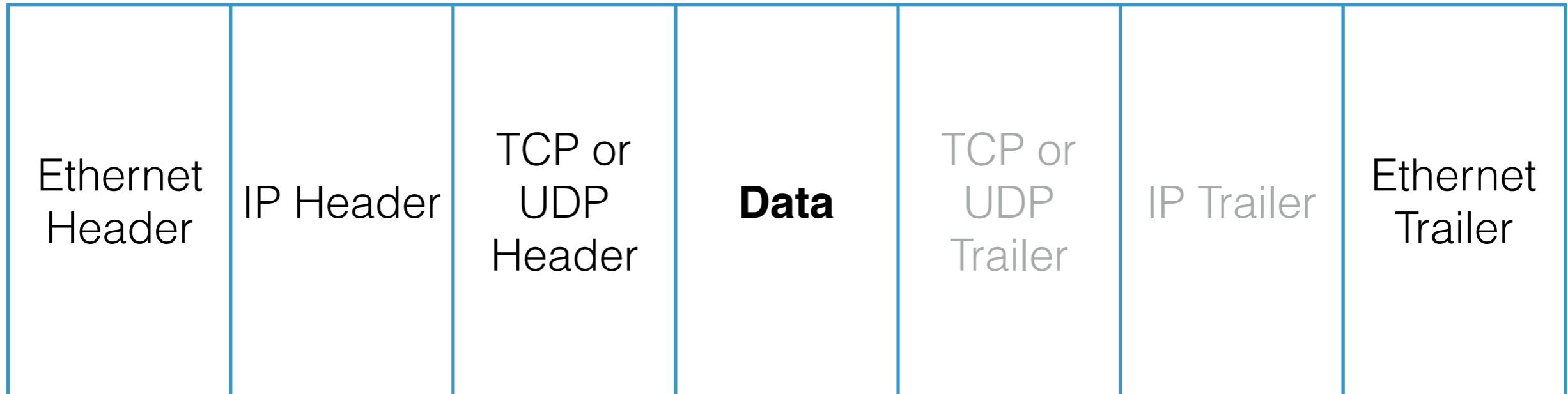
# The DoD Model: Link Layer

- This layer moves data between one network element and the “next” element. This layer is concerned with packetisation, addressing and so on.
- There may be a variety of protocols in use
  - Ethernet is going to be our main focus, but ATM will sometimes still appear

# (Physical Layer)

- This is the actual encoding used to shift bits over a distance. Ethernet can run over co-axial cable (if you are old), twisted pair, fibre optics of various sorts, radio...

# Layers are Encapsulated in Packets



# The OSI Model: extras

- The OSI model's main difference is that there are three layers where the DoD has one: transport's equivalent is presentation, session, transport.
- Presentation is intended to handle data encoding (converting integers for a neutral format everyone can use, for example).
- Session is meant to handle the relationship between multiple connections (say, restarting a failed file transfer)
- Transport is transport.
  - The DoD model pushes all the above up into libraries
  - Experience and history says the extra layers are useless: applications require things that are too specific to provide as generic services. But that's another lecture.

# Abstraction is hard

- The problems come when a layer closer to the application is tasked with providing a service which is a poor fit to the lower layers (like trying to write operating systems in Haskell).
- If your network just does datagrams, a strong transport service is hard (and will take us three/four lectures to describe)
- Conversely, if your network just does virtual circuits, sending a single datagram is very expensive.

# TCP/IP Wins...

- Because there is one transport service for connections and one transport service for datagrams
  - TCP/IP has various experimental transports for special purposes, but they are not widely deployed.
  - It is the responsibility of the implementor to make TCP and UDP work, in full, over whatever lower layers they are proposing.

# TCP/IP wins...

- Because there is exactly one network layer, IP.
  - IPv4 and IPv6 for these purposes differ only in address length.
  - It is the responsibility of physical and link layers to carry IP, and if they can't, they can't carry TCP or UDP.

# OSI loses...

- Because it tried to be “efficient” and provided multiple transport services, ranging from a very thin one to put on top of virtual circuits (“TP0”) to a very complex one for datagrams (“TP4”).
- Because it was about telcos protecting existing business models and capital plant
- So there were **two** different network layers, CONS for connection-orientated (virtual circuit) services, and CLNS for connection-less (datagram) services. This was to keep telcos with different infrastructures happy.
- None of it interworked.

# The Value Chain

- Telcos (largely) sell a service with a service level agreement
- ISPs (again, largely) are much more “best efforts”
  - They are relying on statistical gain, which works most of the time, but in Selly Oak, not so much

# So...

Applications you can make money out of: Facebook (ads), NetFlix (subs), Online Banking (drives other business)

Sale of Internet Connections to consumers and businesses, with large amounts of statistical gain and vague SLAs

Carrying the data belonging to ISPs over a distance: they want an SLA, and only rent bandwidth they need after statistical gain

Carrying data on single-haul links between premises, where you can't use any statistical gain as it's about wires and linecards.

**Consumer  
ISPs**

**Telcos and  
Interconnect**

**Telcos**

# Equipment is Layered

- In two different ways
  - Different pieces of equipment do different jobs in the stack (although the distinctions are becoming blurred)
  - There are aspects of different layers in the same piece of equipment, and separating those makes design, construction and security easier.

# Planes within Elements

- You can view equipment as having a management “plane”, a control “plane” and a traffic (or data) “plane”.
- This is telecoms language, and originally these were literally separate elements in the hardware, “plane” being jargon for one printed circuit board in a rack linked by some interconnect.

# Planes

**Management plane:** where the GUI/CLI runs, where statistics and error reporting happens, where the device is configured. Implemented in software, running today on some general purpose operating system, typically Linux, or something similar.

**Control plane:** where decisions about routing policy are made. In voice switches (which we aren't going to talk about much) this is where calls are set up and cleared down. Almost always software, but might be running on a real-time executive.

**Data plane:** where the actual traffic is shipped. Might be done with special purpose hardware, might be done with exotic software running on exotic hardware ("network processors")

# Different Hardware

- We can divide network hardware up by which part of the model they implement.
- “**Switches**” (hubs, bridges) understand the **Link Layer**. So an ethernet switch (we will talk about what makes it a switch later) switches ethernet packets between links. The interfaces are in some sense the same (might be different speeds, might be different media).
- “**Routers**” understand the **IP Layer** and move packets between potentially very different links.
- “**Hosts**” understand the **transport layer** and **application layer**, and are usually general purpose computers.

# Blurring

- Hosts in fact understand everything, and make pretty good routers as well.
- These days, routers tend to have a switch integrated into them.
- And switches are increasingly “L3 aware” and make switching decisions based in part on the contents of IP headers.
- This is not exact science.

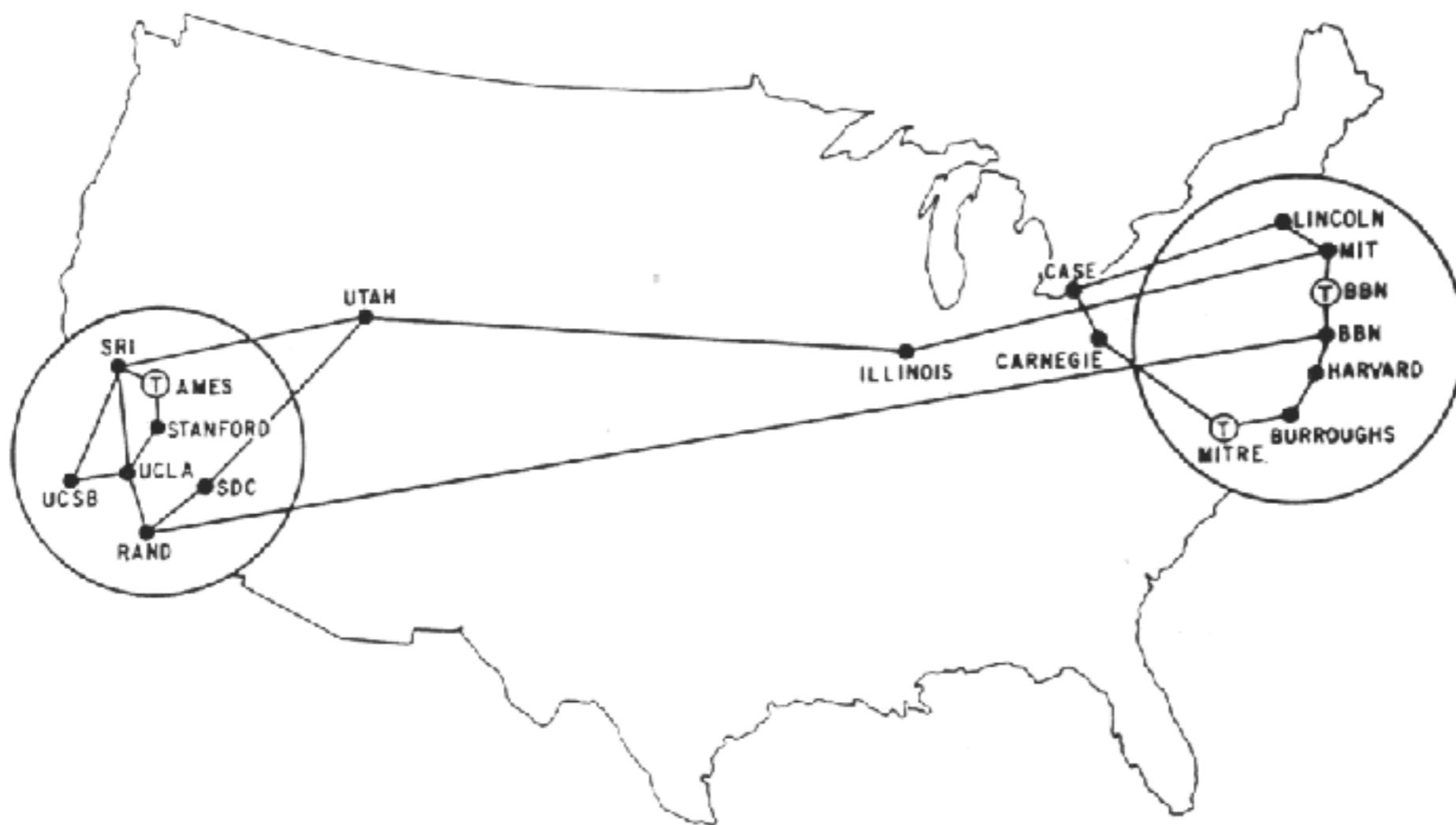
# Networking: Lower Layers

[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# Check Panopto!

- Is it running?
- Is it running?
- Seriously, is it running?

# ARPANet, Sep 1971



MAP 4 September 1971

# LANs, MANs, WANs...

- Division of networking space into **Local** (buildings, campuses), sometimes **Metropolitan** (city) and then **Wide Area Networks**. LANs, MANs, WANs.
- Also (not this course) sometimes **Personal Area Networks**, PANs, mostly today Bluetooth.
- Theoretically, different technical solutions to different engineering problems.
- Rapidly converging.

# LANs and WANs

- Historically, wide-area networks were very slow.
  - a fast long-haul network of the 1980s might be a few tens of kilobits per second
  - Technology existed to go faster, issue was mostly cost and practical availability.
- Even though local-area networks were hitting 10 Mbps by 1980 and 100 Mbps by 1990.
  - Although many were much slower (X.25, RS232 with Kermit, etc, etc)

# LANs and WANs

- Historically, Local Area and Wide Area networks were different in technology, purpose and protocols.
- In Europe and to an extent the US, telco monopolies limited what WANs could do.
- And outside a small number of research environments, LANs were almost entirely proprietary: interworking was mostly with WAN technology.

# WANs

- Used to connect computers together, between buildings more than (say) 1km apart, or sometimes just when crossing a road if the local laws grant a monopoly to telcos.
- Historically there were three main applications:
  - File transfer (lots of problems of format conversion, as even byte-size varied)
  - Job transfer (for use of national facilities for super computers; batch mode)
  - Remote login (when you interactive access to remote systems, which was not always available).
- UUCP very influential and STILL SHIPPED ON MAC!
- ARPANet in the US restricted only to people with government contracts

# WAN Technology

- The key point about the WAN is that for most of its history it is slow.
- Very slow.
- UofB JANET connection 1985: 64Kbps.  
cs.bham.ac.uk JANET connection 1987: 9.6Kbps.  
US/UK ARPAnet connection 1986: 2.4Kbps (yes,  
seriously).
- ARPA/NSFNet backbone 1987: 64Kbps
- 2Mbps links emerge (for most of this) by about 1990.

# WAN Technology

- This means that efficiency is very important: wasting tens of bytes is a significant performance problem
- So if you are going to use the same protocols on WAN and LAN, the protocols in use on the LAN has to consider working over slow-speed, lossy links as well as fast networks inside buildings.
- The crucial issue here is when developed, the LANs were much faster than the WANs; today, that is in many cases precisely reversed.

# Packets and Circuits

- Real circuits involve electrical connections from end to end
- Packet switching involves putting an address on packets and sending them to the destination individually
- Each packet can contain full destination information, or can be associated instead with a virtual circuit.
- Virtual circuits make objects that look like circuits out of a stream of packets
- Assumption: network is a mesh of routers (switches) linked by some sort of medium.

# Packet Switching

- Each packet has addressing information
- A router looks at incoming packets, decides where to send it, sends it on its way
- Router “complexity” scales by the number of packets processed, and possibly other things.

# Connection Orientated / Virtual Circuits

- If your underlying network supports virtual circuits, you can ask the network to send a stream of packets to a specific destination
- The network decides a route, tells all the routers along the way what is happening, and give you some sort of token to identify the flow (“virtual circuit”)
- You then send data with that token attached, and it arrives at the other end complete and in order. The assumption is that there are fewer connections than there are endpoints, so this token is smaller and easier to look up.
- Upside: network is doing a lot of the heavy lifting of ensuring all the data gets there, so your network stack is simplified.
- Downside: the routers are much more complex, as they scale by bandwidth **and** number of circuits **and** rate of circuit creation / destruction.
- Historically, X.25 and (to a lesser extent) Frame Relay. Today, ATM (on its way out, but still present in many networks), and Multi Protocol Label Switching (MPLS) (core of big provider networks).

# Connectionless / Datagram

- Underlying network offers packet switching “in the raw”
- Individual packets are processed by the network and may or may not arrive, and may or may not be damaged. Network is “best efforts”, no guarantees.
- User addresses packets with complete destination information on each and every packet.
- All responsibility for more complex services rests with the end-points: the network is not going to help (although obviously the less damage it does to packets the better)
- This is the basic requirement to put IP over
- You can use a virtual circuit as a link in a connectionless network, but not vice versa.

# Lower Layer Technology

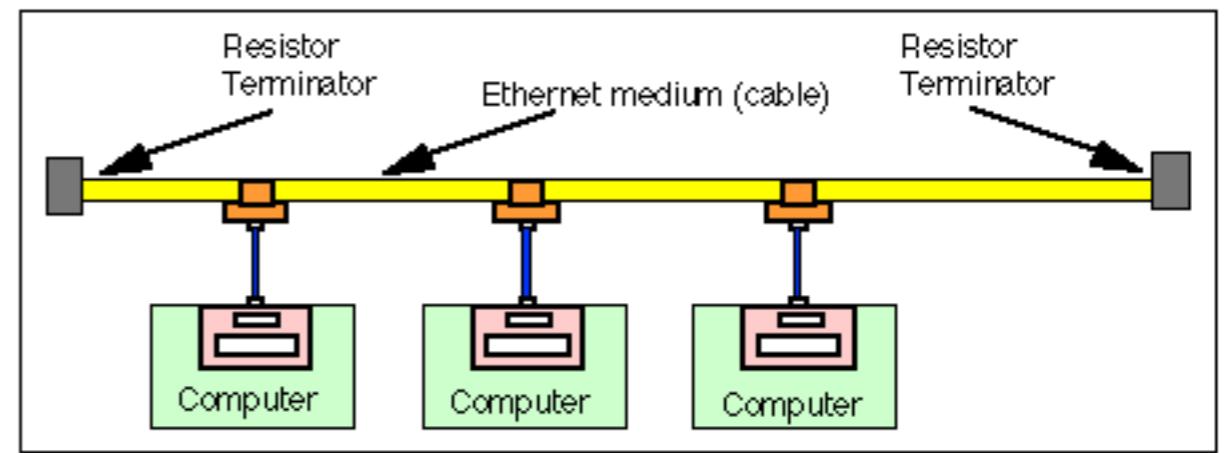
- Ethernet
- Token Ring (IBM Token Ring, FDDI)
- Slotted Ring
- ATM
- MPLS
- SDH
- DWM

# Ethernet

- Developed by Metcalfe and Boggs at Xerox Palo Alto in the 1970s.
- Named after the luminiferous aether that supposedly carried light and radio until disproved by the Michelson-Morley experiment
- Takes inspiration from earlier radio packet networks, notably AlohaNet in Hawaii.

# Topology

- The topology of Ethernet was originally a bus: a single cable with computers connected to it.
- (Early versions are 3Mbps, but for practical purposes “yellow hose” is always 10Mbps).
- Maximum length is 500m (both for reasons of resistance and timing as we will see); can be amplified and regenerated to go 1500m max.



# Format

- 7 bytes of **preamble** (0x55) to allow receivers to synchronise.
- 1 byte **start of frame delimiter** (0x5d)
- 6 byte **source address** (48 bits)
- 6 byte **destination address**
- 4 byte **VLAN tag** (optional)
  - First two bytes 0x8100 to keep older equipment happy
- 2 byte **type or length**
  - If  $\leq 1500$ : length. If  $\geq 1536$ : type, with length found by looking for end of the packet
- 42–1500 bytes of **payload**
- 4 byte **CRC**
- 12 byte-time **inter-packet gap**.

# Finding the end without a length

- Checksum is computed continuously, so when you have a set of bytes where the last four bytes are the correct checksum for the whole packet, you know you have reached the end.
- CRC calculation of (data + CRC) generates the magic number 0xC704DD7B - google this number for the gory GF(2) details.
- Or wait until the inter-packet gap
- Or both

# Basic Logic

- Only one station can talk effectively at a time, as every station can see what every other station is saying and multiple transmitters will interfere.
- Each station waits until no-one else is talking, and then start transmitting.
- What could possibly go wrong?



# Collisions

- Ethernet is formally known as “**CSMA/CD**” — Carrier Sense Multiple Access Collision Detection.
- The magic comes from what happens when there is a collision.

# Collision Detection

- As a station transmits, it also listens to the ether and checks the ether only contains the signals that are being sent
  - this has to be done in hardware, as it is mostly an analogue problem.
- If there is a mismatch, someone else is transmitting at the same time.

# When Collisions Happen

- First action is to “jam” the network: send a set pattern so everyone knows a collision is in progress.
- Critical that the whole ether knows about the collision before the packet has finished being sent
  - Imposes a minimum packet size (64 octets), which is a function of the maximum diameter of a collision domain (1500m). Jam pattern pads packets to this length at least.

# Recovery from Collision

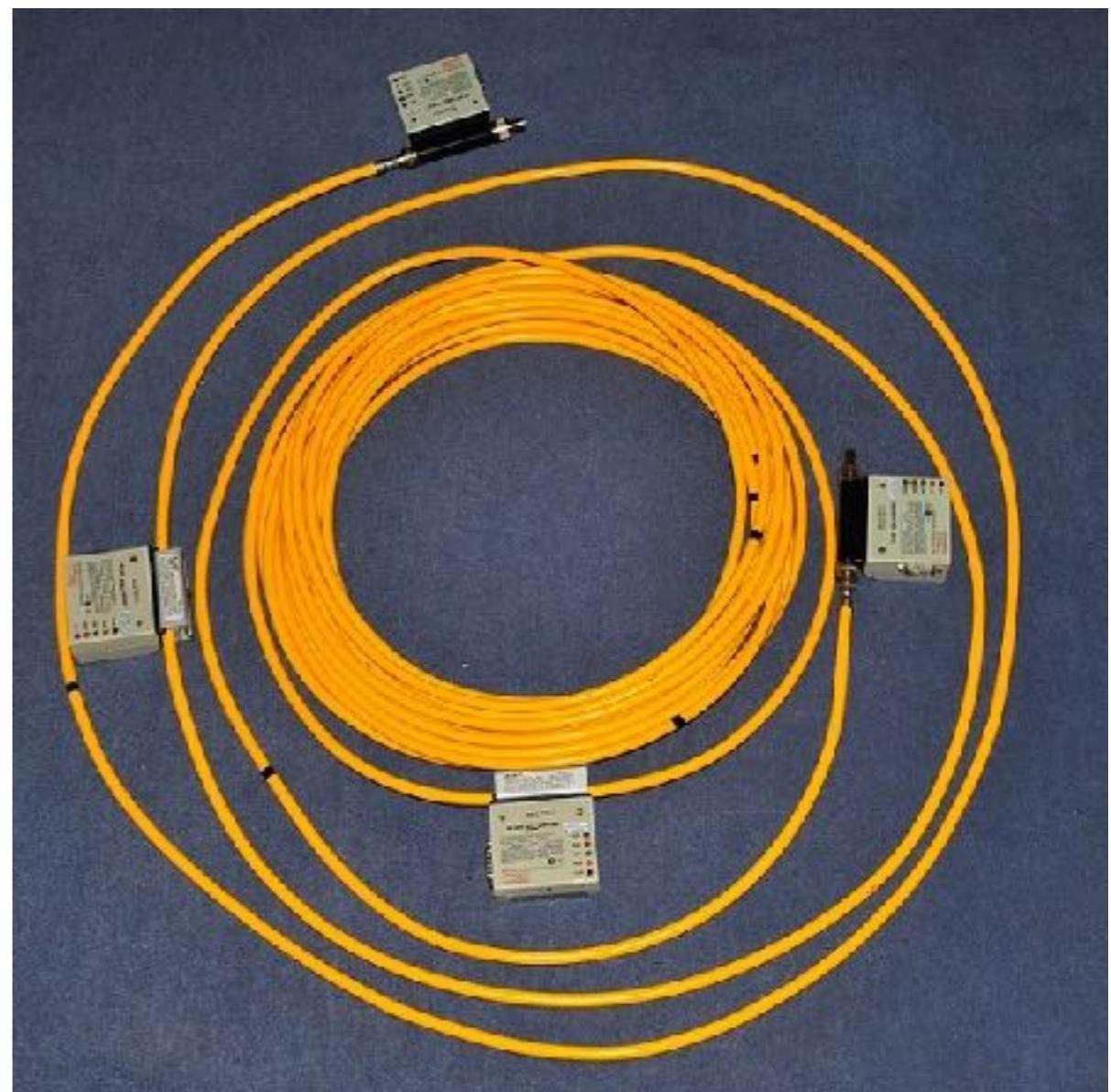
- On the first attempt, choose a random number  $k$  from  $\{0,1\}$  and delay  $k \times 512$  bit periods before trying again.
- More generally, on the  $n$ th attempt, choose a random number  $k$  from  $\{0..2^n\}$  and delay  $k \times 512$  bit periods before trying again.
- After 10 attempts, give up.
- Randoms come from things like serial numbers; they don't need to be very good quality.

# Problems

- Collisions increase non-linearly with load, and the precise curve depends on the exact traffic mix
  - “Ethernet capture effect”
- Latency for a single packet is unpredictable, because some number of collisions may delay it.
  - This can be overstated by advocates of other protocols.

# Sizes

- Maximum frame size 1500 bytes payload plus 22 packets of header (larger ends up slowing down stations wanting to exchange small packets)
- Minimum frame size 64 bytes (slightly wasteful for, say, telnet, but making it smaller reduces maximum diameter of network)
- Maximum “diameter” 1500m (from complex rules surrounding number of permissible repeaters).
- 500m and 10Mbps gives name:  
**10Base5**.

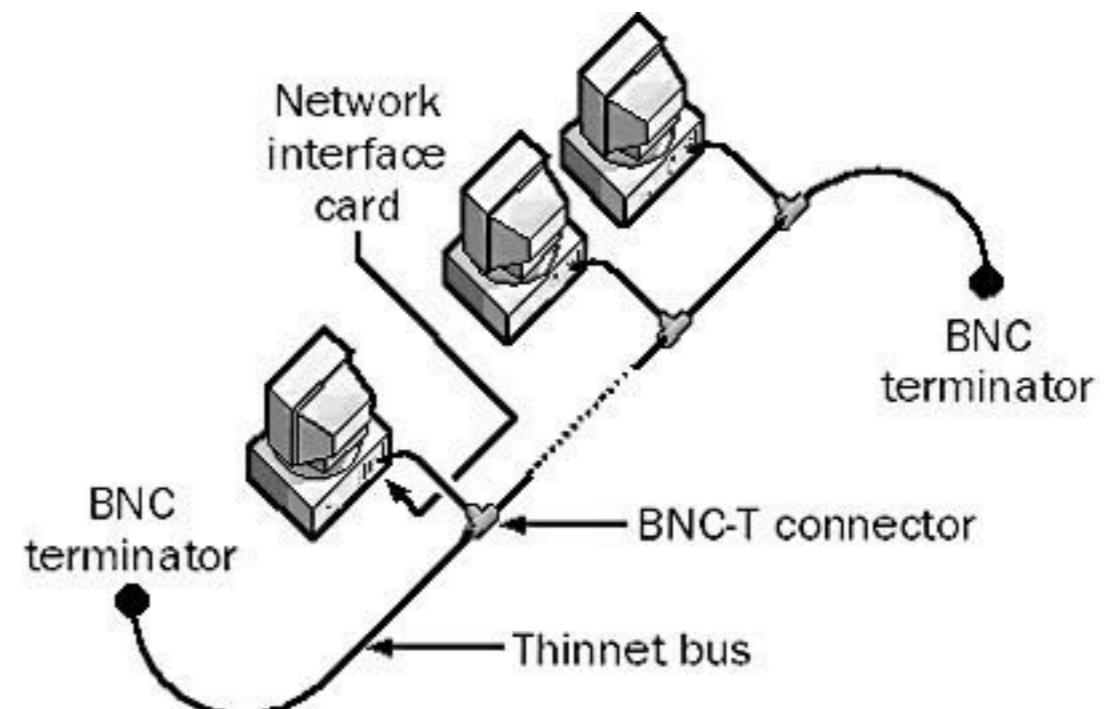


# Problems

- Cable is heavy, expensive and difficult to install (tight, or more to the point loose, minimum bend radius requirements).
- Installing taps for transceivers involves drills, and risks damaging the cable.
- Need for transceivers adds cost and complexity.
- Performance issues lurking in the background

# An interim: 10base2

- Instead of using thick co-ax, use thin coax. Higher resistance, so limited to 185m: **10base2**.
- Instead of using transceivers, simply bring the coax to the computer and attach it with a tee-piece.
- Cut the cable, rather than drilling into it.



# 10Base2

- Otherwise it works much the same
  - much smaller maximum diameter of <600m
  - Different terminators
- Can be mixed electrically and logically with 10Base5 (rules are complex and only of historical interest)
- Probably the dominant networking of the 1980s and early 1990s: older buildings still full of it.

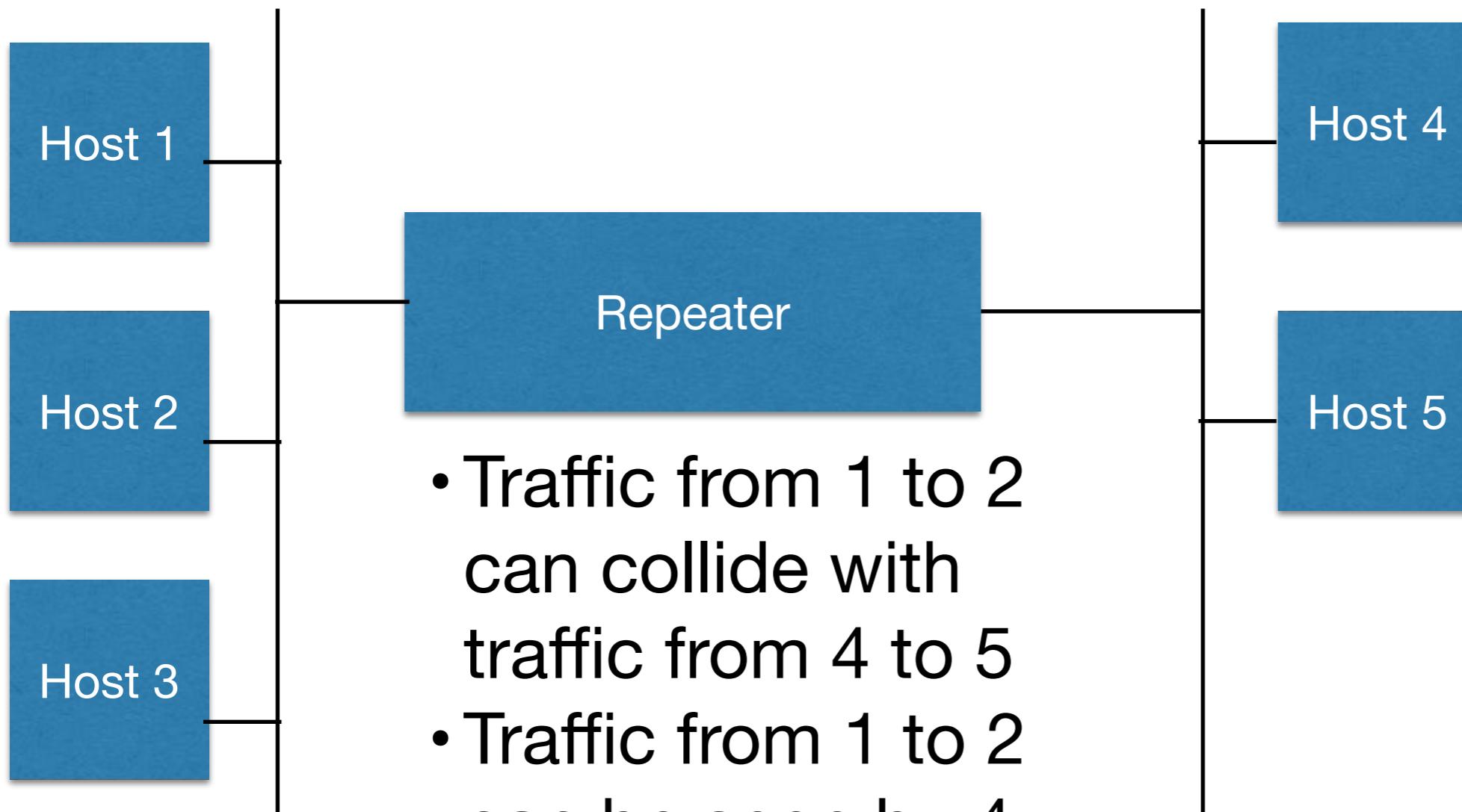
# 10BaseT

- Coax cable still a pain: expensive, awkward to install, easily damaged.
- 10BaseT looks like “modern” ethernet: Up to 95m of twisted pair (four conductors in two pairs) using RJ45 connectors to a **hub**. Originally “Category 3” cabling, basically voice.

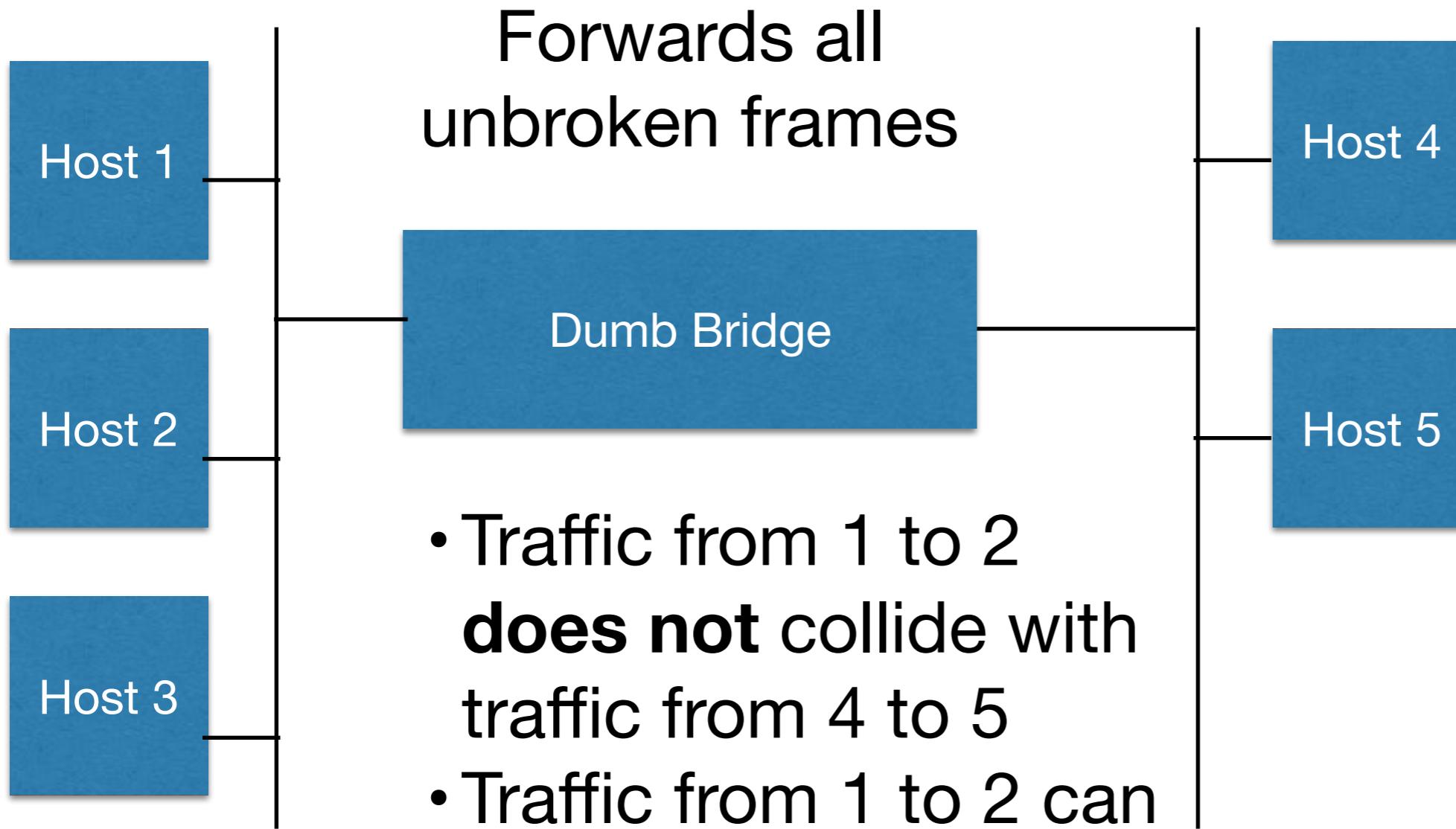
# Hubs, Repeaters, etc

- A repeater is just an amplifier: collisions are seen on both sides
- A bridge receives, buffers and transmits frames, so collisions are not propagated
  - “learning” or “filtering” bridges only send frames that belong on the other side; stupid bridges just propagate everything.
- Ether hubs are **repeaters**, not bridges. There are collisions when two stations talk.

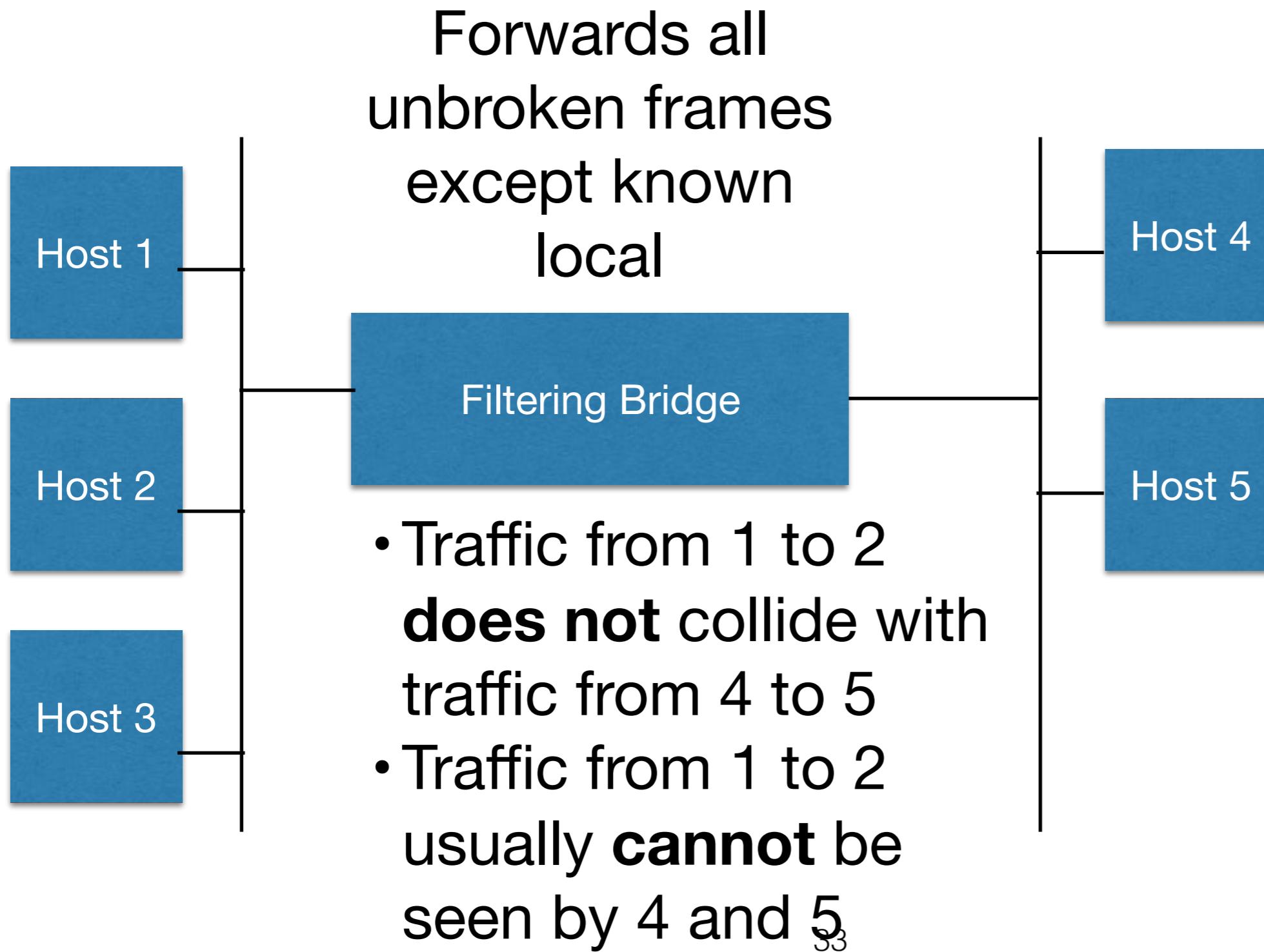
# Repeater



# Dumb Bridge



# Filtering/Learning Bridge



# Faster and Faster

- 10BaseT is no faster than 10Base2, but cheaper and more flexible to install.
- 100BaseT raised the speed, but still had potential for collisions
- Full duplex and switching made 100BaseT much faster, following by 1000BaseT (GigE) and then 10GigE, 40GigE and the nascent 100GigE.
- Technology similar, but stricter wiring rules (“Cat5” for 100BaseT, “Cat5e” or “Cat6” for faster).

# Ethernet Switches

- A switch is a set of learning bridges in a box.
- Each interface is its own collision domain.
- Packets to unknown destinations are sent out of all ports, otherwise only traffic for devices plugged in to the port is sent.
- “Full Duplex” means traffic goes in and out without colliding as **each direction** is a separate collision domain.
- Large buffers internally deal with congestion.
- Result: no collisions.

# Cut-Through Switches

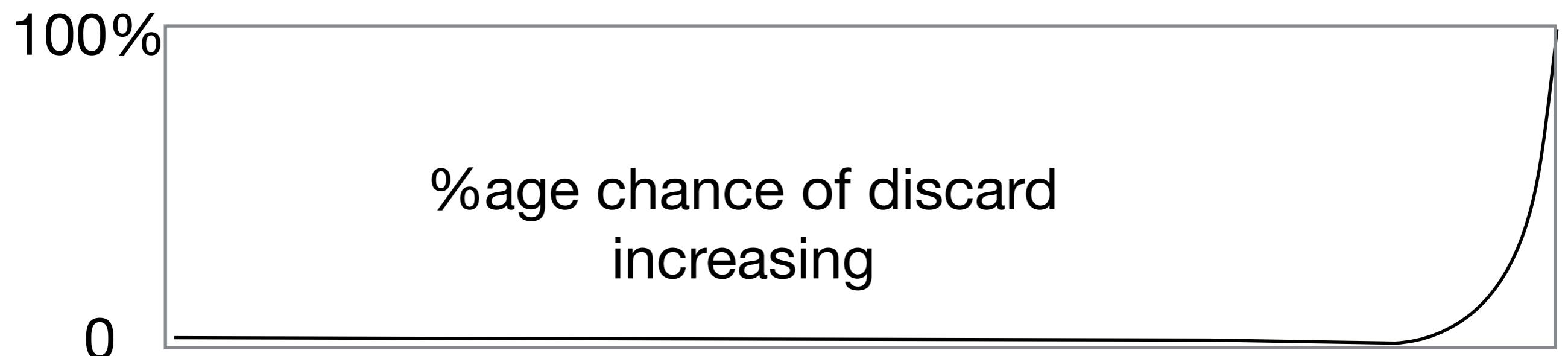
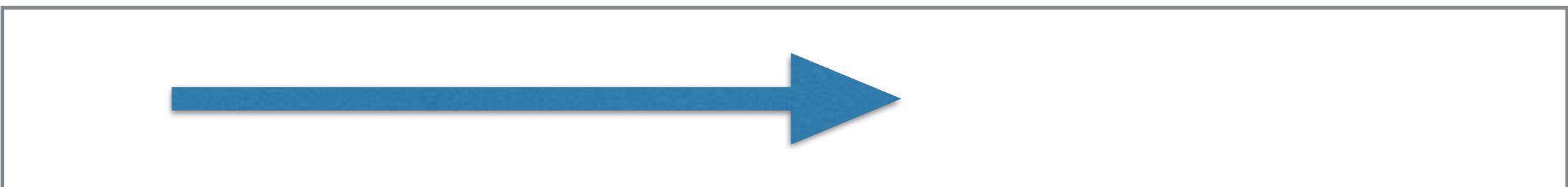
- Conservative switches accept frames in their entirety, check the checksum, then transmit them to other interfaces
  - Introduces additional latency compared to a straight piece of wire (For GigE, 1 bit period is 1ns, full packet is  $1500 \times 8\text{ns} = 12\mu\text{s}$ , equivalent to ~3.6km of copper; for 10BaseT it's 1.2ms, or 360km of copper).
- Aggressive switches look at the header, and immediately start transmitting on the correct interface (“cut through”).
  - Latency is just the 160–192 bits of the header, so <2% of a full packet: ~60m of copper for GigE, 6km for 10BaseT.
- This propagates broken frames if there are any to be propagated, as it can't check the checksum

# Random Early Drop

- Naively, when a buffer fills up, you start to drop packets as you can't put them anywhere
- We will come on to transport connections in detail, but in general, packet loss results in a timeout followed by a retransmission, which net slows things down after some interval
- A new strategy is to randomly drop packets with a probability which increases as the buffer fills, so the dropping starts earlier but more gently, hopefully reducing speed before real loss starts to happen.
- The loss of packets is seen by the sender when the acknowledgements stop, and is a signal to the sender to slow down. You hope.

# Random Early Drop

Buffer filling...



# Token Ring/Bus

- Ethernet was argued to behave badly under high load, although limited evidence was available.
- Token Rings and Token Buses pass a “token” from station to station.
- The station that holds the token can transmit, and then passes the token on when it has finished.

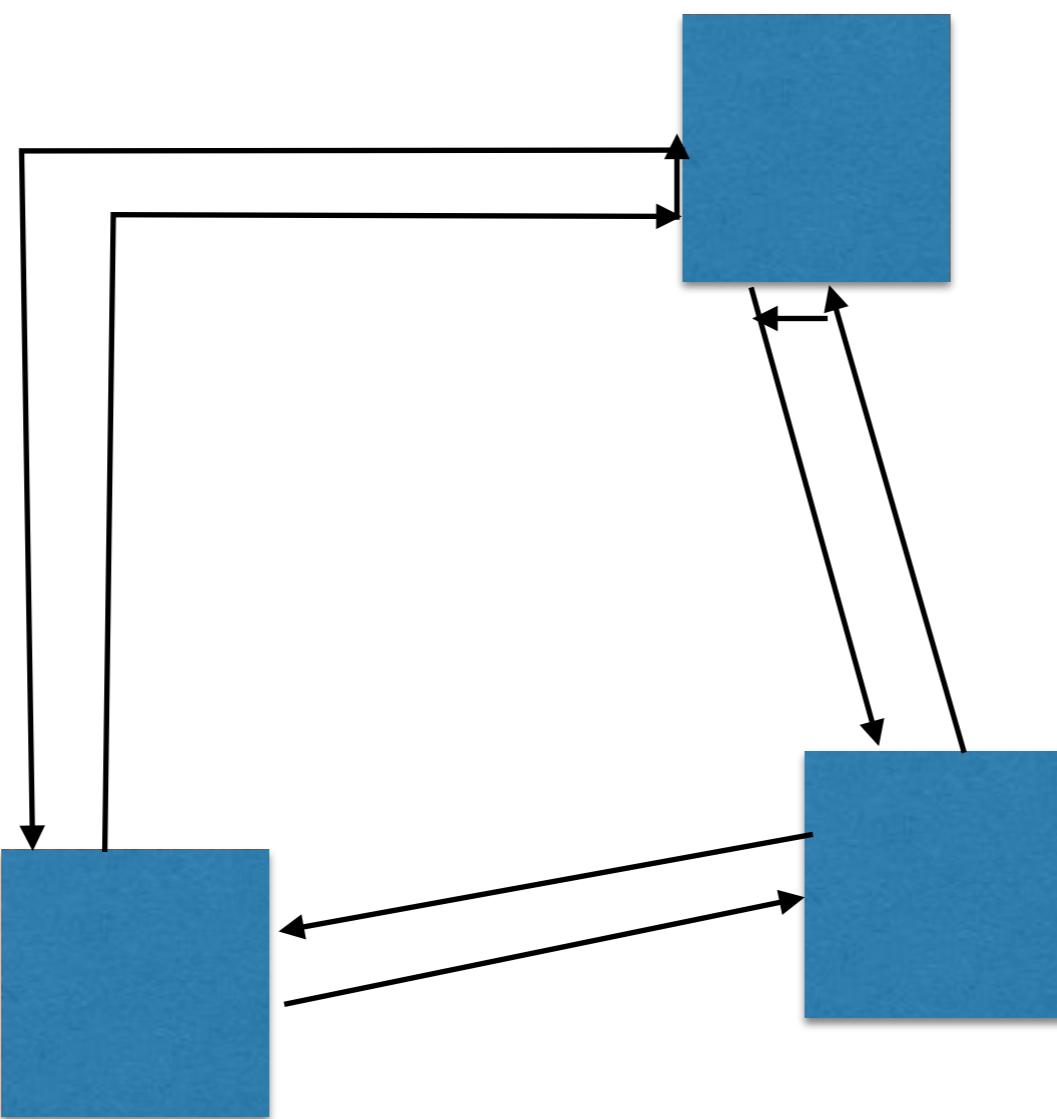
# Problems

- In theory, offers bounded latency: the token will always circulate in  $n_{\text{stations}} * \text{max\_packet\_period} * \text{fudge}$ .
- In practice, very complicated to get right
  - Token loss/creation
  - Station failure

# Examples

- IBM Token Ring (4Mbps, later 16Mbps)
  - Still occasionally encountered
  - Uses star topology for wiring
- FDDI Fibre (100Mbps, fastest game in town until switched full-duplex 100BaseT with cut-through switches).
  - Genuine dual ring, with complex passthrough and loop reversal algorithms
  - Still in use in interconnects and data centres, although not in new installations
  - Extraordinarily robust and stable in performance

# Dealing with Failure



What happens if  
two nodes fail in  
a large ring?

# CDDI

- There is also a variant called CDDI, FDDI over copper, using very specialised hubs with multiple paths.
- It works well and can survive multiple failures; it was also staggeringly expensive until supplanted by switched 100BaseT.

# Slotted Rings

- Known as “Cambridge Rings” from their place of development (Cambridge in East Anglia, not Cambridge Mass).
- Instead of circulating a token, empty data frames circulate, in the manner of the conveyor belt in a Sushi restaurant, or other alternatives



# Slotted Ring

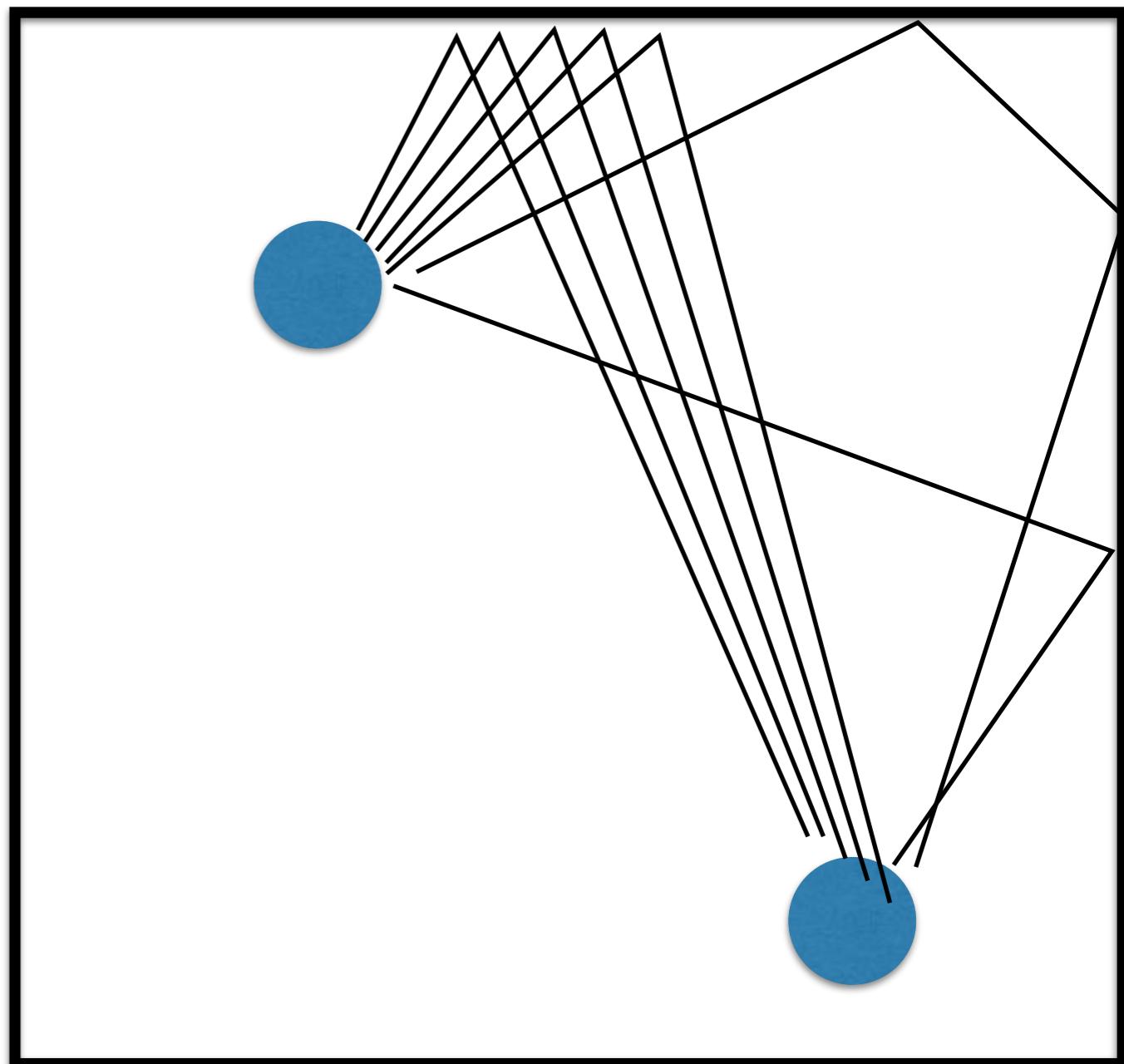
- Requires a minimum length of network, so that there are a sufficient number of empty packets circulating
  - Hence long lengths of cable coiled under the floor
- Popular in UK universities as boards were cheap and easy to build and drivers were available for common Unix variants; never achieved significant traction elsewhere.
- Probably lurking in floor voids of [cl.cam.ac.uk](http://cl.cam.ac.uk), [ukc.ac.uk](http://ukc.ac.uk) and elsewhere.

# ATM: The Telco Strikes Back!

- ATM: Asynchronous Transfer Mode
- Proposed by Telcos as part of the broadband unified services architectures of the 1990s.
- For reasons of nasty politics, breaks data into a stream of 48-byte packets.
  - Americans ~~and everyone remotely sensible~~ wanted 64, French wanted 32 because then they could run voice without needing echo cancellation, compromise of 48 suited no-one.
- Virtual circuits, so only needs a 5-byte header (but again political, as 5 is ~10% of 48 which was seen as “acceptable”)

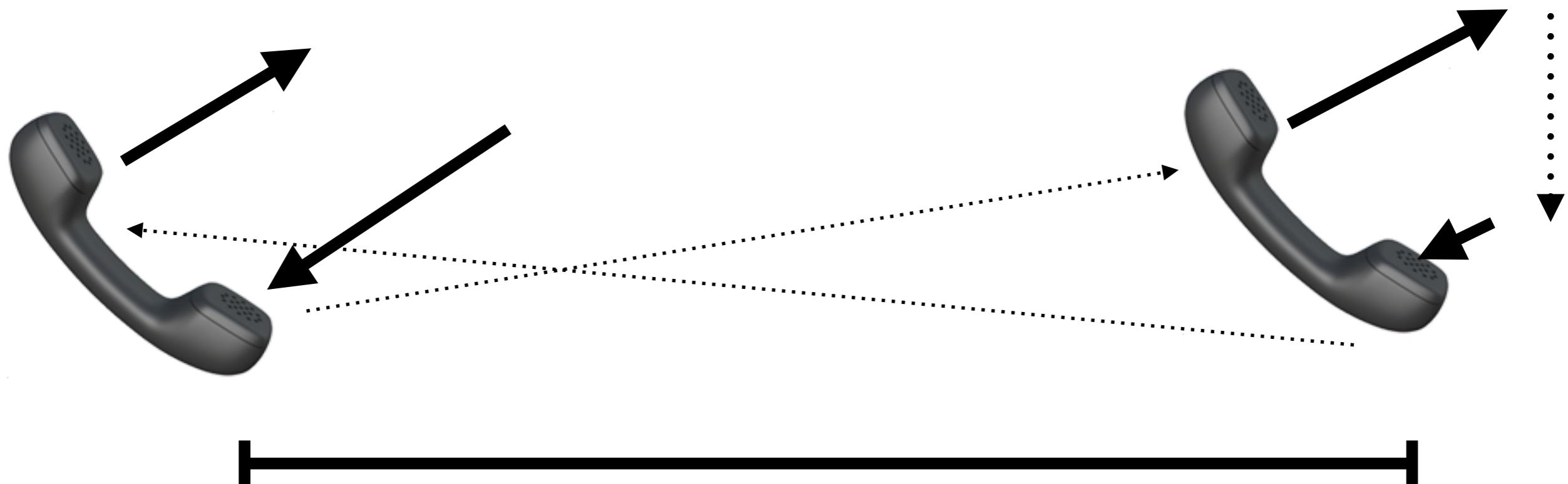
# In passing...echo cancellation

- If you are speaking in a room, the echo from your voice is a diffuse field of noise, as the many possible paths all have slightly different lengths.
- Your brain is very good at dealing with this, and you aren't normally aware of the reverberation of a small room (but wait until you get older!)
- Your brain rejects any stronger echoes arriving within ~50ms (“Haas effect”)



# Telephones aren't rooms

Any echo is a sharp, single event that your  
brain struggles to reject



Target: 35ms RTT, equal to ~10m of air

Light travels 10000km

Reality in digital systems...?

# America is big

- Speed of light means that for a phone call from New York to San Francisco you are not realistically going to be able to get it under 35ms whatever you do
- Hence you need to use complex electronics to filter out the echo (“echo cancellation”) to get decent “toll quality” audio.
- France is a lot smaller, and you can get away without the complexity

# Latency caused by filling packets

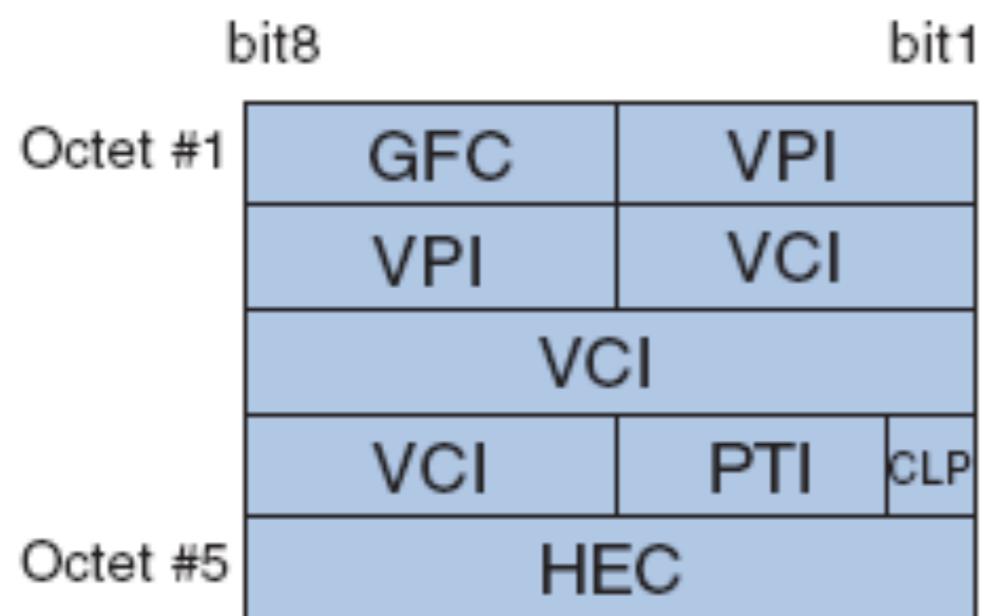
- Filling a 64 byte packet when you are sending 8KHz, 8 bit samples (ie, 64Kbps): 8ms
  - Note: filling a 1280 byte packet (20x bigger) is 160ms!
- Receiving it at the other end: 8ms
- That's 32ms round trip: almost all your budget gone
- With 32 byte packets, 16ms: you've got time to switch the packet
- $35\text{ms} - 16\text{ms} = 19\text{ms}$ , 5700km at speed of light
- Americans were running echo cancellation already so didn't care, and wanted larger packets for efficiency
- French wanted smaller packets to avoid the problem.
- Everyone lost, as 48 byte packets satisfied no-one (and made the standard look a bit mad)

# ATM Justification

- Smaller packets gives lower latency (but not low enough, as we saw)
- Switching a stream of small datagrams is allegedly very inefficient (large headers, lots of routing decisions)
- ATM is therefore virtual circuit orientated
- Also incorporates extensive traffic shaping and policing options (more later)

# ATM Headers

## User-Network (UNI)

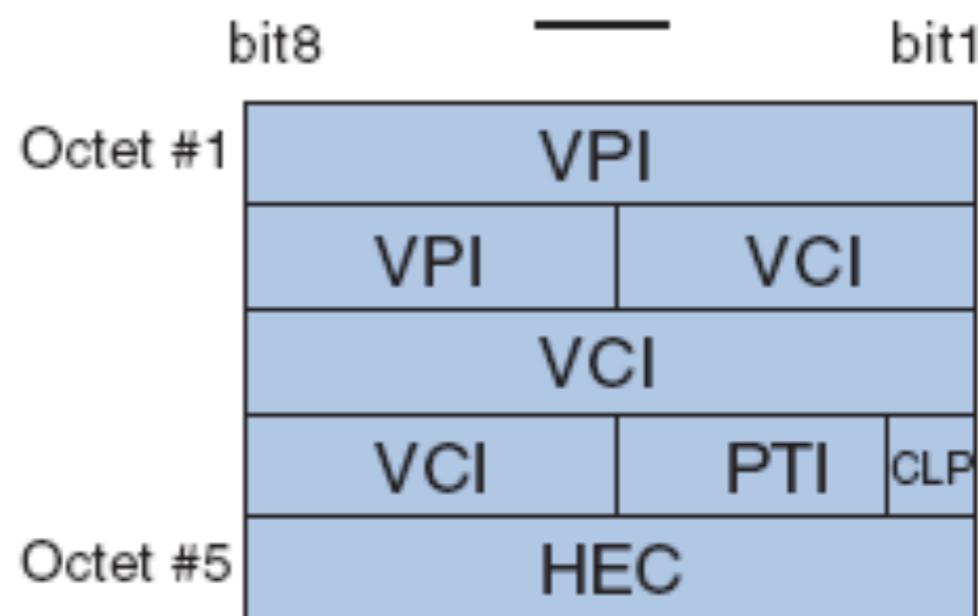


GFC: Generic Flow Control

VPI: Virtual Path Identifier

VCI:Virtual Channel Identifier

## Network-Network (NNI)



PTI: Payload Type Identifier

CLP: cell loss priority

HEC: Header Error Control

Note: for extra fun, addressing information is not byte-aligned

# ATM25

- 25Mbps
- Can be built using adaptations of IBM 16Mbps Token Ring hardware; easy to encapsulate into USB 1.1 or USB 2.0.
- Was the dominant interface for ADSL modems during the late 1990s, and is the internal switching format for ADSL exchange equipment
- Still very influential in the form of PPPoA.

# ATM155, 622...

- Faster variants used (mostly) within telco core networks, although enjoyed a brief period of use in data centres prior to being killed by cheap GigE.
- Can be used to carry IP in various forms
  - “classical” uses a virtual circuit as a two-station network,
  - “LAN Emulation”, aka “LANE”, tries to emulate a larger ethernet with lots of switching: scales very badly
- Further breaking ~1500 byte IP/Ethernet up into 48 byte cells (“AAL5”) appalling for performance and reliability
- But is a good way to mix “toll quality” voice with data for multi-service networks.
- Proved too complex, too expensive, and switch vendors were acquired and progressively run down
- Still in use in carrier networks, but being pushed out by ethernet.

# Nailed Up Circuits

- ATM is virtual circuit orientated: you ask the network to establish a circuit, and once set up the packets just have to say which circuit they are on.
- Original idea for UK ADSL broadband was switched virtual circuits (SVC): you could choose your ISP dynamically, and a visitor could plug into your line and use their ISP (think dial-up, if you are old enough).
- Unfortunately...

# Performance Hopeless

- ATM switches couldn't handle volume of circuit establishment required, even in early trials ("Project Ascot" in Ealing, a few thousand houses)
- Solution was "permanent virtual circuits" (PVCs), nailed up at the point at which the service is commissioned. Hence the "0.38" or "0.101" you may be familiar with: that's the identity of the PVC from your house to your ISP.
- Messy.

# A bit of transmission

- SDH: Synchronous Digital Hierarchy
  - aka SONET (synchronous optical networking) in US., which has detailed differences.
- Multiplexes “trails” of 2Mbps upwards into STM1 (155Mbps), STM4 (622Mbps), STM16 (2.4Gbps) and STM64 (10Gbps).
- You can extract and insert individual 2Mbps trails from a passing 10Gbps stream (“add/drop multiplexor”)
- “Packet over SONET” aka PoS still regularly used for long-haul Internet traffic. Most telco transmission equipment up until five years ago was SDH.

# Wave Division Multiplexing

- (D|C) WDM
  - Dense/Coarse Wave Division Multiplexing
  - Use different colour light to transmit multiple streams down a single fibre. For “colour” say “lambda” if you want to hang with the cool kids.
  - Coarse: 20nm difference between adjacent channels
  - Dense: originally 0.8nm difference between adjacent channels (100GHz channels based around 193.1THz reference).
    - now can be 0.4nm or 0.2nm differences in wavelength.
  - Commercial systems go up to 10Tbps and beyond~

# Using WDM

- Each channel can carry different traffic (including ATM, ethernet, SDH, whatever)
- Increasingly, ethernet straight over WDM is the way telcos are going, with the assumption that most ethernet will just be carrying IP (what else is there?)

# Summary

- Ethernet works for getting data between computers that have cables between them. It won the battle.
- Other things can be made to work, but were more expensive/more complicated/harder/more political, and lost
- In 2017:
  - Short range: ether over copper
  - Medium range and/or hostile environments: ether over multimode fibre
  - Long range: ether over WDM

# Networking 7: IP

[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# Contents

- IP as a concept
- IPv4 addressing
- IPv6 addressing
- Packets and routing

# Why IP?

- Far and away the dominant networking protocol of the past thirty years
- A single network layer, over which all transports can run, and...
- ...a single network layer which can run over all available lower layers

# What is IP?

- A unreliable, unsequenced datagram service
- You put an address on a packet and the network makes an attempt (“best efforts”) to deliver it somewhere, hopefully the right place.
- There is no checksum covering the data, so even protection from corruption is the responsibility of upper layers
- Hard to imagine a network layer which offers less

# Why did IP win?

- Easy to implement, and implemented on all “Computer Science Favourites” of the early 1980s (Multics, TOPS-10, TOPS-20, Lisp Machines, Berkeley Unix)
  - Berkeley Unix, in turn, became the operating system of choice for Unix workstations, the dominant computer science environment of the late 80s and early to mid 1990s.
- Works over everything, from long-haul radio to exotic high-speed fibre.
- Has/had the Support of US DoD, (D)ARPA and NSF
- Actually rather good as well

# History

- Proposed in 1974 paper [1]
- Experimental versions 0 to 3 described in Internet Experimental Notes (IENs 2, 26, 28, 41, 44, 54)
- IPv4 described in RFC760, January 1981.
- Updated by RFC791, September 1981, which is still current (1349, 6864 and 2472 describe and clarify some little-used extensions).

[1] Vinton G. Cerf, Robert E. Kahn, "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communications, Vol. 22, No. 5, May 1974 pp. 637–648

# Packet Format

- Each row is 32 bits, four bytes, 1 word. Options are optional, so a typical header is 20 bytes (5 words)
  - Note fields aren't byte aligned: this is a 1970s design.

0	1	2	3		
0 1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0 1		
<hr/>					
<b>Version</b>	<b>IHL</b>	<b>Type of Service</b>	<b>Total Length</b>		
<hr/>					
<b>Identification</b>		<b>Flags</b>	<b>Fragment Offset</b>		
<hr/>					
<b>Time to Live</b>	<b>Protocol</b>	<b>Header Checksum</b>			
<hr/>					
<b>Source Address</b>					
<hr/>					
<b>Destination Address</b>					
<hr/>					
<b>Options</b>		<b>Padding</b>			
<hr/>					

# 32 bit addresses

- At the time, insanely large. The Arpanet reached peak of 113 nodes by 1983, split in half by Arpanet / MILNet separation.
  - The question is not “how could they be so short-sighted as not to use 48 or 64 bits?” Rather “isn’t it amazing they didn’t use 16 or 24 bits?”
- Original concept of IP was to have 8 bits indicating the site, and 24 bits to specify a machine at that site.
  - Only proposed users were big US universities and companies, US government and a small number of defence-related operations, all in NATO.
- This was seen as wrong very quickly: 256 sites simply not enough, even in the early 1980s.

# Notation

- Conventionally written as four decimal numbers, each encoding one byte (wastefully), separated by dots. Printable form 7..15 bytes long (4 times 1..3 digits plus three dots).
- Typically not zero-padded (usually 192.168.1.1 rather than 192.168.001.001), but sometimes people use %03d.
- Hexadecimal would have been much better (you can encode, decode and mask by eye) but hexadecimal wasn't used much in the 1970s. We're lucky it wasn't octal, which was! IPv6 is hex, as we will see.

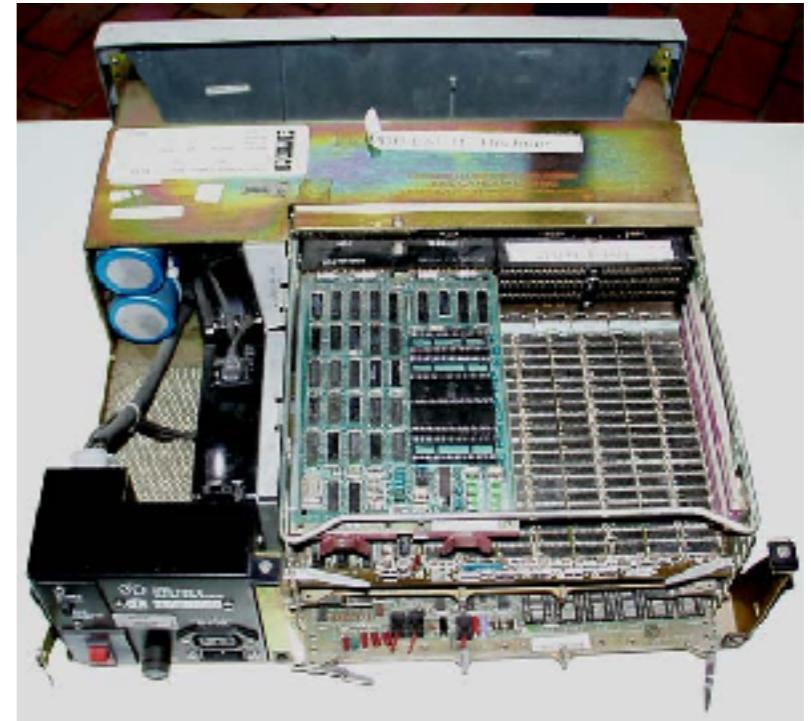
1001 0011	1011 1100	1100 0000	1111 1010
147	188	192	250
9 3	b c	c 0	f a

# Routing Decisions

- All IP addresses identify a network (the leftmost part of the address) and the host on that network (the rest of the address). We will define “leftmost part” and “rest” soon.
- A router ignores the host on network part for all non-local addresses, looks up the network in a routing table, and chooses which interface to send the packet out through.
  - “Default network” handles all other cases.
- A router close to the centre of the inter-network needs to know about most networks.
- And at the time IP was designed, 64 KILO bytes was a LOT of RAM (maximum address space on a pdp11).

# Priority: performance on limited hardware, 1980-style

- With the speeds and feeds of the era, building large distributed routing tables was hard. It was essential to keep the number of networks about which information needed to be exchanged to a minimum.
- Limiting it to 256 networks was unreasonable, but there had to be a low limit.
- 32 bits = ~ 4 billion addresses, in a era where a few thousand computers would be seen as an upper bound. Efficient allocation did not matter and was not a design goal: they anticipated utilisation of a fraction of a percent.
- The priority was being able to make routing decisions quickly on realistic hardware (roughly, a DEC LSI 11/23 “fuzzball”: 64kB per process maximum). Those decisions are with us still today.



# “Classful” Addresses

- If the address starts with a 0, the first 8 bits identify the network, the remaining 24 bits identify the host on that network. 1.x.y.z through to 127.x.y.z
- Gives 128 “Class A” addresses to large sites that get  $2^{24}$  (~16 million) hosts each. These went to MIT, BBN, Ford, DEC, Boeing, the UK government.
  - HP now has net 15 and DEC’s network 16, via its purchase of Compaq who had bought DEC.
  - Net 0 not used, net 127 reserved for loopback (almost exclusively 127.0.0.1). Instantly wastes over 16 million address ( $2^{25}-1$ ).

# “Classful” Addresses

- If the address starts with 10, the first 16 bit identify the network, the remaining 16 bits the host on that network.  
128.x.y.z through to 191.x.y.z
- Gives  $2^{14}$  (16384) “Class B” addresses with  $2^{16}$  (65536) hosts each.
- Initially easy to get for smaller universities and companies
  - Birmingham had **two**, one applied for centrally, one applied for to use in CS by Bob Hendley and me, not used after 1990, now sold (we didn’t see a penny of it).

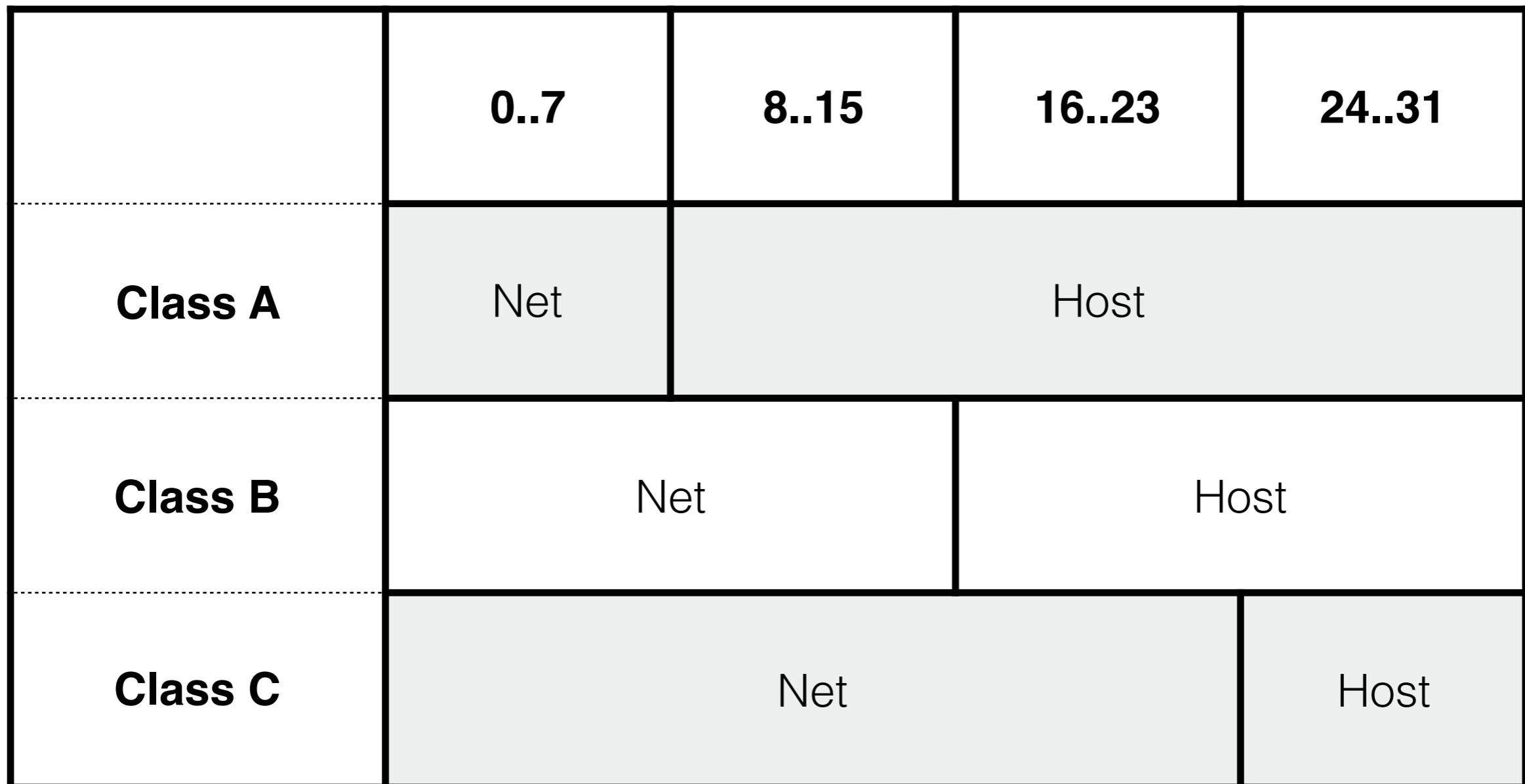
# “Classful” Addresses

- If the first three bits are 110, the first 24 bits identify the network, the remaining eight the host on the network. 192.x.y.z through 223.x.y.z
- Gives  $2^{21}$  (2097152) “Class C” addresses with  $2^8$  (256) hosts per network.

# “Classful” Addresses

- Remaining space used for multicast (“Class D”, first four bits 1110, 224.x.y.z through 239.x.y.z) and reserved for experimental use (“Class E”, 1111, 240.x.y.z through 255.x.y.z).
- Multicast never really achieved much outside private networks, so address space that was allocated is wildly excessive.

# Classes



# Wasteful Allocation

- Total reachable space is 87% of the available addresses
- But no university, not even MIT, has 16 million hosts, so Class A space very sparsely occupied
- Very few universities or companies have 65536 hosts, so Class B space very sparsely occupied
  - In both cases, even fewer hosts externally accessible
- Class C space was initially available, in bulk, to anyone with an Internet connection (Fulcrum Communications as it then was had 18 Class Cs, 4608 addresses, for <500 employees). This is not untypical.
- Estimates vary, but it's unlikely that today more than 25% of address space is usefully deployed. Huge shortages outside USA, Canada and western Europe (for practical purposes, Cold War era NATO).

# But easy for routers

- Most of the early Internet was in fact the holders of the Class A addresses.
- Routers first looked at the address to see if it was “local”: is the destination directly connected to the router via some sort of network connection? If so, send the packet direct to that destination.
- Otherwise, they looked at the first bit of an address. If it was zero, they looked up the first byte in a 128-entry table of “next hops”: the IP numbers of other routers that are directly connected (LAN, WAN) and are believed to be “closer” to the destination.
  - With 32 bit addresses, such a 128 entry table occupies 512 bytes. Easy, even in 1980.
  - The original ARPAnet backbone was “net 10”, later re-purposed as we will see.
- If it’s found, send the packet to that router.
- Otherwise look up remaining two or three bytes in a more complex table (some sort of tree or hash table).
- Otherwise use the default route, if it exists.

# Routing vs Memory

- Today you can have a fully populated routing table for every /  
24 in 128MB ( $2^{27}$ ) of RAM (ie, a Raspberry Pi or an iPhone can  
function as a core router)
- There is no current need to do this, but you could have a  
unique destination for every IP number (all 32 bits) in 32GB of  
RAM ( $2^{35}$ ) (a reasonably spec'd desktop PC, a small server)
- Routing now not a big computational problem: you simply  
index into a sparsely populated array rather than using  
complex trees and hash tables
- Hardly worth even caching the last eight (or whatever)  
destinations (a common trick of the past).

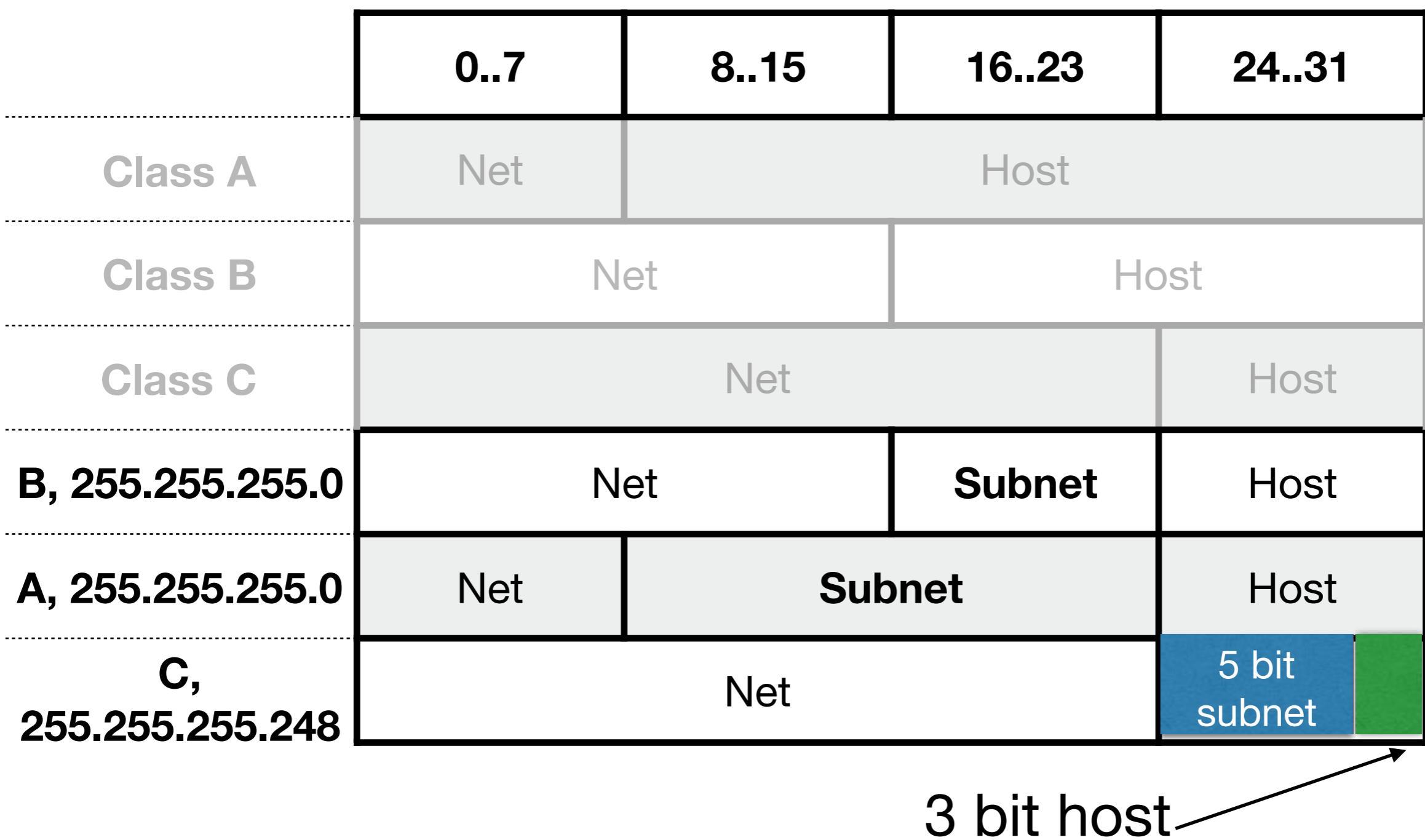
# Sub-Netting

- People used the large amount of address space to plan their internal networking by structuring the “host” part of the network.
- Users of a Class B could treat their  $2^{16}$  addresses as 256 networks each of 256 hosts. Users of a Class A could treat their  $2^{24}$  addresses as 65536 networks each of 256 hosts, or (with a lot of care and complexity) 256 groups each of 256 networks each of 256 hosts
- Practical limits of the time meant that you didn’t want more than ~100 hosts on an Ethernet anyway.
- By late 1980s, you could “subnet” on non-byte boundaries, too
  - Systems that can’t subnet, or can’t subnet on arbitrary boundaries, are now obsolete; the hacks used to work around it are of historical interest only (and vile) — look up “Proxy ARP” if you have a strong stomach.
- Outside world sees one network, internally everyone knows the extra information about the layout of addresses

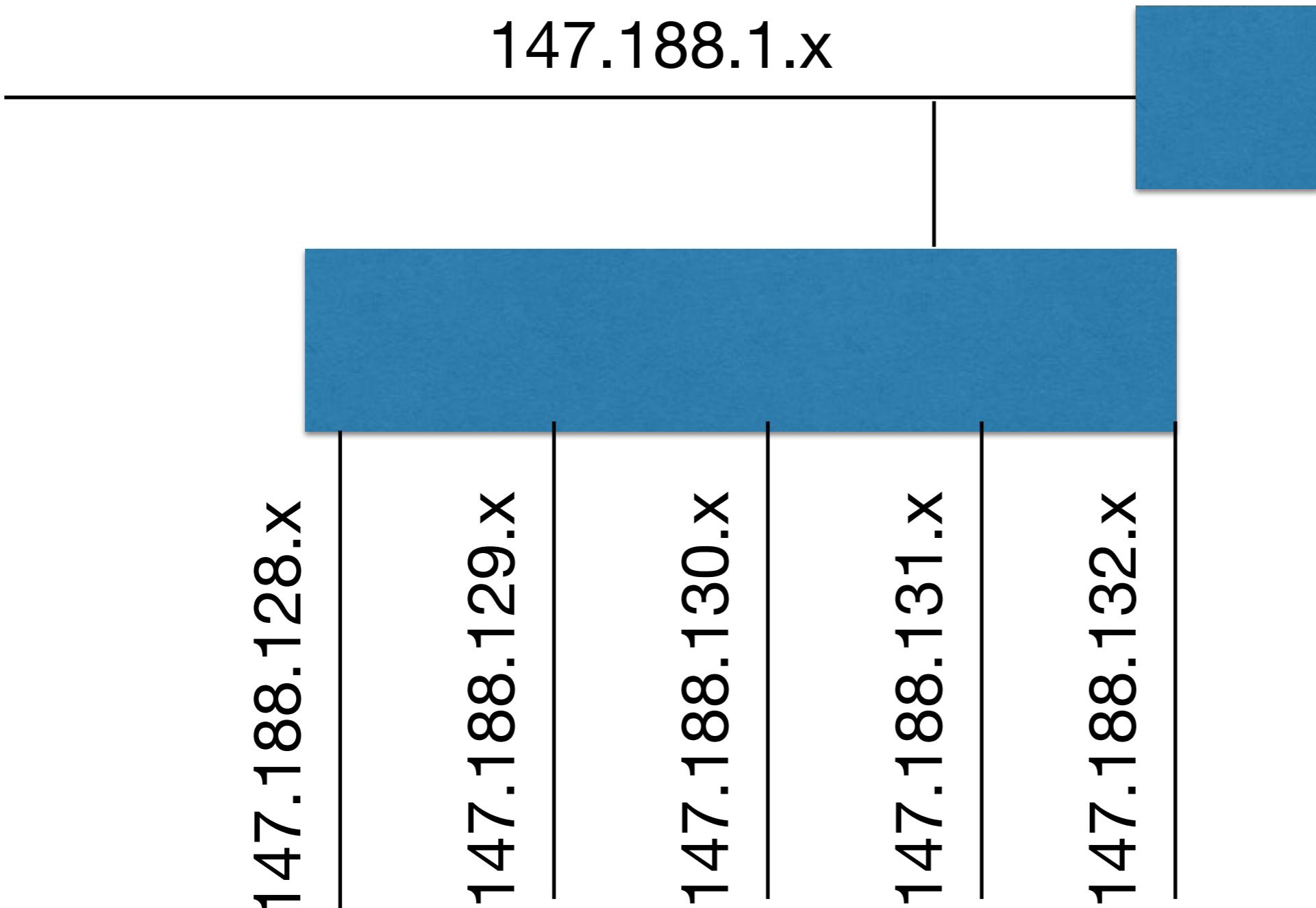
# Netmasks

- Originally notated as a netmask: the bit pattern which can be logical-and'd with an address to yield a network number.
- Class A (later /8, as we will see) is 255.0.0.0,
  - $255 = 11111111$ .  $10.1.2.3 \& 255.0.0.0 = 10.0.0.0$ .
- Class B (/16) is 255.255.0.0
- Class C (/24) is 255.255.255.0
- A Class C used as 32 networks each of 8 hosts is 255.255.255.248 (/29):  $248 = 11111000$ .

# Subnets



# Sub-netting



# CIDR and Slash Notation

- Problems of waste with Classful networking and need for more flexible sub-netting combined to produce Classless Interdomain Routing, CIDR.
- Every network address has a “netmask” which describes how much of it is network and how much of it is host.
- Class A is now a “slash eight” (MIT is 18.0.0.0/8), Class B is now a “slash sixteen” (Bham is 147.188.0.0/16) and Class C is a “slash twenty four” (FTEL is, amongst others, 192.65.220.0/24).

# And in the other direction...

- To make routing tables more compact, a group of eight contiguous Class Cs can be placed together under a /21, or sixteen contiguous Class Bs under a /12.
- This is called “super netting”, but is now less useful as routing table size is not an issue.

# Non-Byte Masks

- This extends trivially to boundaries which are not classful.
- So an ISP wanting to give prosumers some IP numbers can hand out a /28, which gives the user 16 IP numbers.
  - My home network is 81.187.150.208/28, giving me 81.187.150.208 through to 81.187.150.223, 14 hosts and two different broadcast addresses.

# Recovering the Class As

- Some of the Class As were recovered, by a variety of threats, cajoling and bankruptcy (and Stanford being cool and stand-up and giving it back voluntarily in exchange for a bunch of Class Bs)
- Widespread allocation had stopped at Net 57 anyway (Societe Internationale de Telecommunications Aeronautiques S.C.R.L.)
- Remainder and returned networks were broken up and issued in varying size allocations
- <http://www.iana.org/assignments/ipv4-address-space/ipv4-address-space.xhtml>

# RFC1918

- Class A Network 10 became free when the Arpanet backbone was closed down.
- So 10.0.0.0/8, along with the available 172.16.0.0/12 (172.16.x.y through to 172.31.x.y) and 192.168.0.0/16 (192.168.x.y) were allocated for private use. These **must not** be routed outside private domains.
- This usually requires “address translation”, which we will cover later.

# IPv6: IP with big addresses

- 32 bit addresses have run out (last /8s allocated in February 2011).
- 128 bit addresses gives  $2^{96}$  more addresses.
- Population of planet is  $<2^{34}$  (16 billion) and likely to remain so
- $2^{34}$  people,  $2^{128}$  addresses,  $2^{94}$  addresses per person.
- $2^{94} = 19807040628566084398385987584$  ( $\sim 2 \times 10^{28}$ )

# In reality, more like $2^{64}$

- Minimum allocation unit is a /64; even your mobile phone will probably get a /64. IPv6 is “really” a 64 bit protocol.
- Intention is that with 64 bits available after the routed “prefix”, allocation of addresses on local networks is much easier
- $2^{64}$  still gives  $2^{30}$  addresses each (~1bn per person).
  - At the moment, only  $2^{61}$  addresses available to allocate, so  $2^{27}$  each (~120million per person)

# IPv6

The diagram illustrates the structure of an IPv6 header. It consists of several fields arranged horizontally:

- Version**: Located at offset 0.
- Traffic Class**: Located at offset 4.
- Flow Label**: Located at offset 8.
- Payload Length**: Located at offset 12.
- Next Header**: Located at offset 16.
- Hop Limit**: Located at offset 20.
- Source Address**: Located at offset 28, spanning from index 32 to 56.
- Destination Address**: Located at offset 56, spanning from index 56 to 80.

Vertical lines with '+' signs indicate the boundaries between fields and the start/end of the address fields. The source and destination address fields are further subdivided into four 8-bit bytes, as indicated by the vertical lines within those fields.

# IPv6 Addresses

- Standard format is a hex string, broken into 16 bit chunks with colons, leading zeros suppressed
  - 2001:8b0:129f:a90f:60c:ceff:fedd:f68
- “::” means “as many zeros as fit here”
  - 2001:8b0:129f:a90f:: = 2001:8b0:129f:a90f:0:0:0:0

# IPv6 reservations

- ::/128 “uninitialised”, ::1/128 “loopback” (note only one address, not 16 million!)
- ::ffff/96 IPv4 mapping (ie ::ffff:1.2.3.4)
- fc00::/7 Private address space
- fe80::/8 link local,
- 2001:db8::/32 documentation etc (and a few others for similar purposes)
- 2000::/3 allocated as single block for normal use.
  - $2^{61}$  address blocks for end-users.

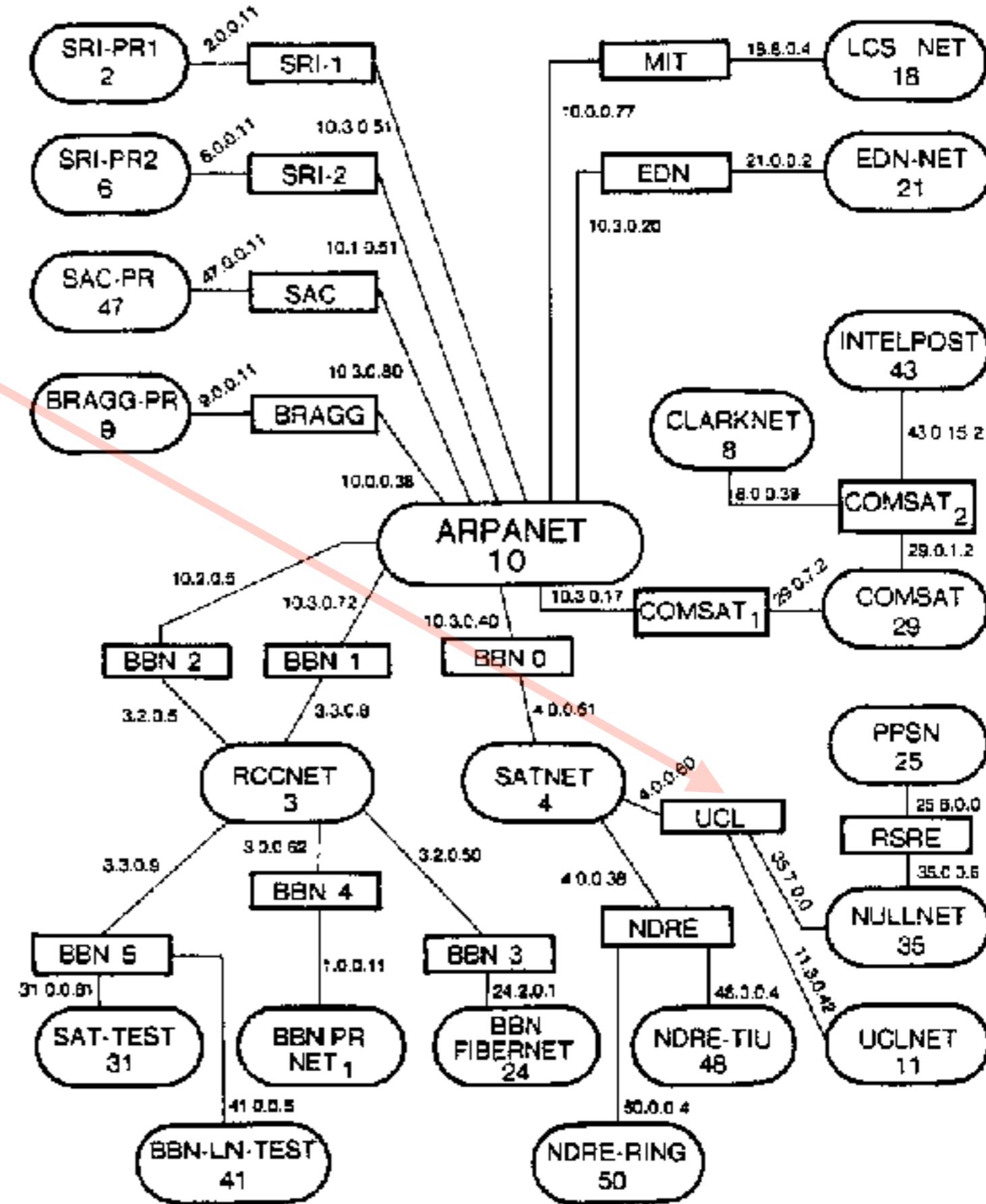
# IPv6 Routing Tables

- Will in the long-term require the complexity originally used for IPv4
- At the moment, very sparse (sadly)
- 1TB is  $2^{40}$  bytes, so  $2^{64}$  is  $2^{24}$  TB, which isn't going to happen any time soon. But nor is  $2^{64}$  allocated networks.

# IP in operation

- Sender: choose an interface we believe to be closer to the destination, and send the packet
- Recipient: if the packet is for us, process locally.
  - Otherwise, send it on.
- We will cover routing in more detail later, so we are just trying to get the general flavour here

- Consider packet at router at UCL
- Network 11? It's local, send via local ethernet (or Cambridge Ring, I seem to recall).
- Network 35 or 25? Send via Nullnet, whatever that might be.
- Otherwise, send via Satnet to main Arpanet.



# Simple Example

```
igb@ossec-sol:~$ netstat -nrv
```

IRE Table: IPv4

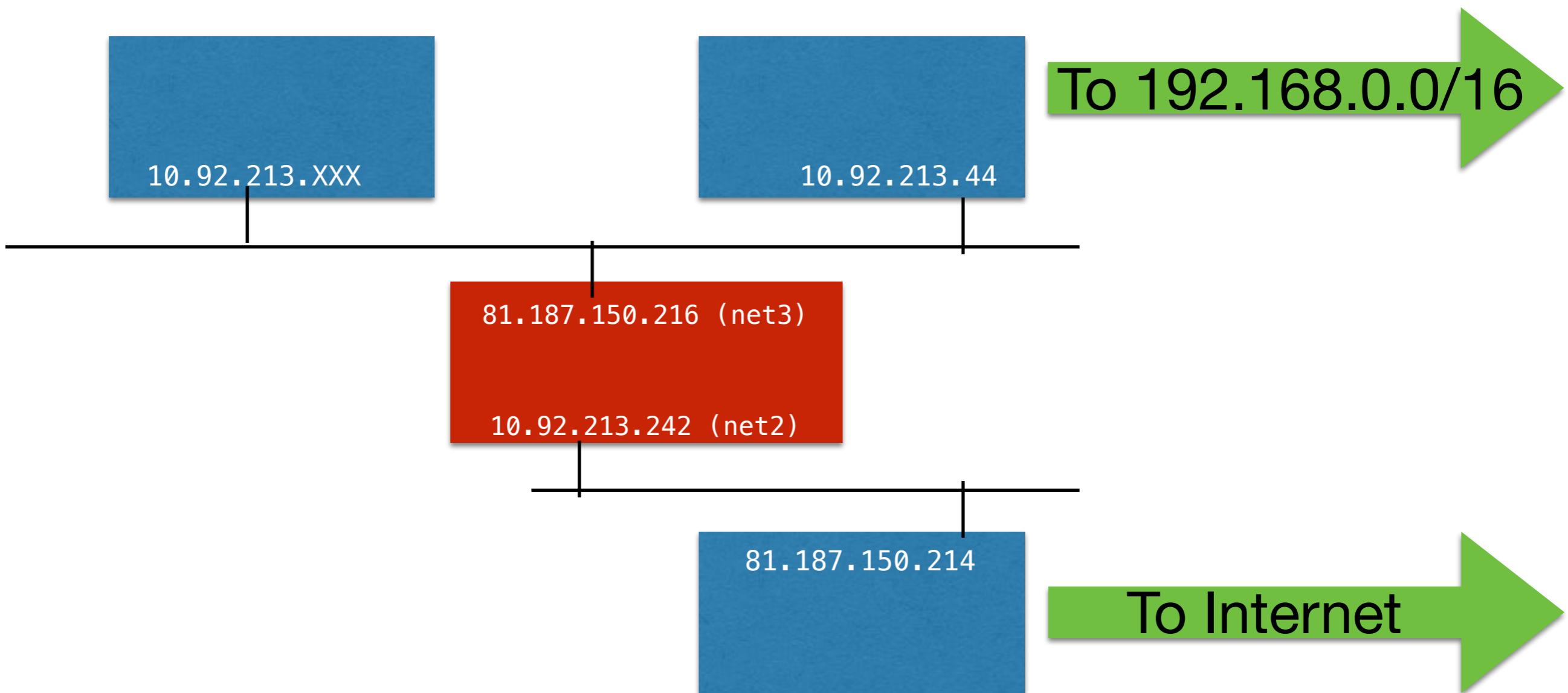
Destination	Mask	Gateway	Device	MTU	Ref	Flg	Out	In/Fwd
default	0.0.0.0	81.187.150.214		0	2	UG	8426	0
10.92.213.0	255.255.255.0	10.92.213.242	net2	1500	6	U	757664	0
81.187.150.208	255.255.255.240	81.187.150.216	net3	1500	5	U	1854	0
127.0.0.1	255.255.255.255	127.0.0.1	lo0	8232	2	UH	0	0
192.168.0.0	255.255.0.0	10.92.213.44		0	1	UG	0	0

Machine with two network interfaces

Note net2 is RFC1918, probably “internal”, and net3 is not, probably “external”

192.168/16 and default are additional routes

# In Pictures



# Routing Decisions

- Decrement the TTL (and do so each time you've held on to the packet for a second, not that that happens).
- Look at each destination we know about, starting with the longest mask and working to the shortest
- Here we have locally connected ethernets (net2 and net 3)
- Traffic to 10.92.213.0/24 and 81.187.150.208.28 is “local” and goes direct over ethernet
- Traffic to 192.168.0.0 is sent over the ethernet to 10.92.213.44 (**a gateway**)
- Traffic to anywhere else is sent over the ethernet to 81.187.150.214 (again, a gateway)
- Machine might itself be a router: whether it forwards packets that arrive on one interface with addresses on the other side is a policy decision.

# IP on Ethernet

- On an ethernet, or other “point to multi-point” network, how do we find the MAC address of the next hop?
- IPv4 uses ARP: Address Resolution Protocol
  - Simple, old and frighteningly insecure
  - Ask “WHO HAS” a particular IP number
  - Station with that IP number, or someone who claims to know about it, tells us. Ripe for exploitation, as we will learn in network security lectures
- IPv6 uses “Neighbo(u)r Discovery Protocol” to do similar job, with ICMPv6 messages 135 (solicitation) and 136 (advertisement)

# IP on point-to-point links

- If a link is point to point, you just send the packet down it.
- Might be a physical point-to-point link (a serial line of some sort, perhaps) or might be a tunnel (packets encapsulated in other packets). We will talk about tunnels later.
- “The internet is a large open field, with many tunnels running underneath it”.

# IP to Gateways

- When sending a packet to a gateway, you look up the gateway address using ARP if necessary, but the IP destination remains the ultimate destination. Gateways don't appear in the IP header.
- Gateways at other end of point to point links just involve sending the packet.

# Hop Counts

- Each packet has a Time To Live (TTL).
- Decrementated each time the packet is processed, or whenever a router holds on to it for more than one second (rare today).
- When it hits zero, packet is discarded and an error report (“ICMP Time Exceeded”) is generated.
- Typical initial value 30–60, depending on current estimates of “diameter” of internet
- Prevents packets circulating endlessly in case of routing loops or similar.
- Requires re-computation of the header checksum in IPv4
  - Fast algorithms used to recompute it based on knowing what has changed: this isn’t a secure hash
- IPv6 doesn’t have a header checksum, relying instead on lower layers being reliable and upper layers being sensible
  - Good reason for routers and switches to have ECC RAM. 1 flipped bit in  $10^{12}$  = 1 silently broken packet per second at 1Tb/sec.

# Networking 9: Address Allocation

[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# Ethernet Addresses

- Ethernet Addresses (MAC addresses) are normally inherent to the machine.
- 48 bits (6 bytes)
  - 3 bytes identify manufacturer (manufacturers can get multiple blocks)
  - 3 bytes identify device
- Can be changed for good-ish reasons (some old protocols altered the MAC address to match the higher layer address), bad reasons (“MAC spoofing” to evade access controls) and unusual reasons (duplicates are not unheard of, although you would be amazingly unlucky).
- Debate as to whether machines with two interfaces should have one or two addresses: standard says one, reality (and requirements of 2017) says two. Issue only arises now with elderly Sun hardware.

# Network Numbers

- Where do you get an IP network from?
  - Small allocations come from your ISP, and belong to your ISP. You have to renumber your network if you use them and change ISP.
  - Larger “provider independent” allocations can be obtained from **regional registries**.
  - You need to make a very good case to get these now
    - We will discuss other issues with PI towards the end of the course.

# Where do local IP numbers come from?

- Static Allocation
- bootp (obsolete)
- DHCP/DHCPv6
- SLAAC (IPv6 only)

# Static Allocation

- The end point is given an IP number in some sort of configuration file / memory / register.
- Each time the device boots, it gets exactly that IP number, even if it is wrong for the network, clashes with other devices, etc.
- The machine might notice that it is a duplicate, but otherwise has very little protection.

# Static Allocation

- Essential for routers and infrastructure devices that need to be up in the very early stages after a power failure.
- Often used for servers that need to have fixed addresses (www.my.domain, mail.my.domain).
  - But there are alternatives for this.

# bootp

- Device broadcasts its ethernet address
- “bootp server” hands back an IP number and some other stuff (DNS servers, default router).
- What is the problem with this?

# bootp limitations

- RFC951
- Ironically, written by Sun employee number one, but never actually used by Sun as bootp not powerful enough for their requirements.
- Only really works for static assignments by another route (perhaps to machines too limited to be able to store a static address), because no means to reclaim addresses that are no longer used
- Normally configured with a large table statically mapping MAC addresses to IP numbers.
- Obsolete for this reason. Some DHCP servers offer bootp for backwards compatibility: usually now switched off.

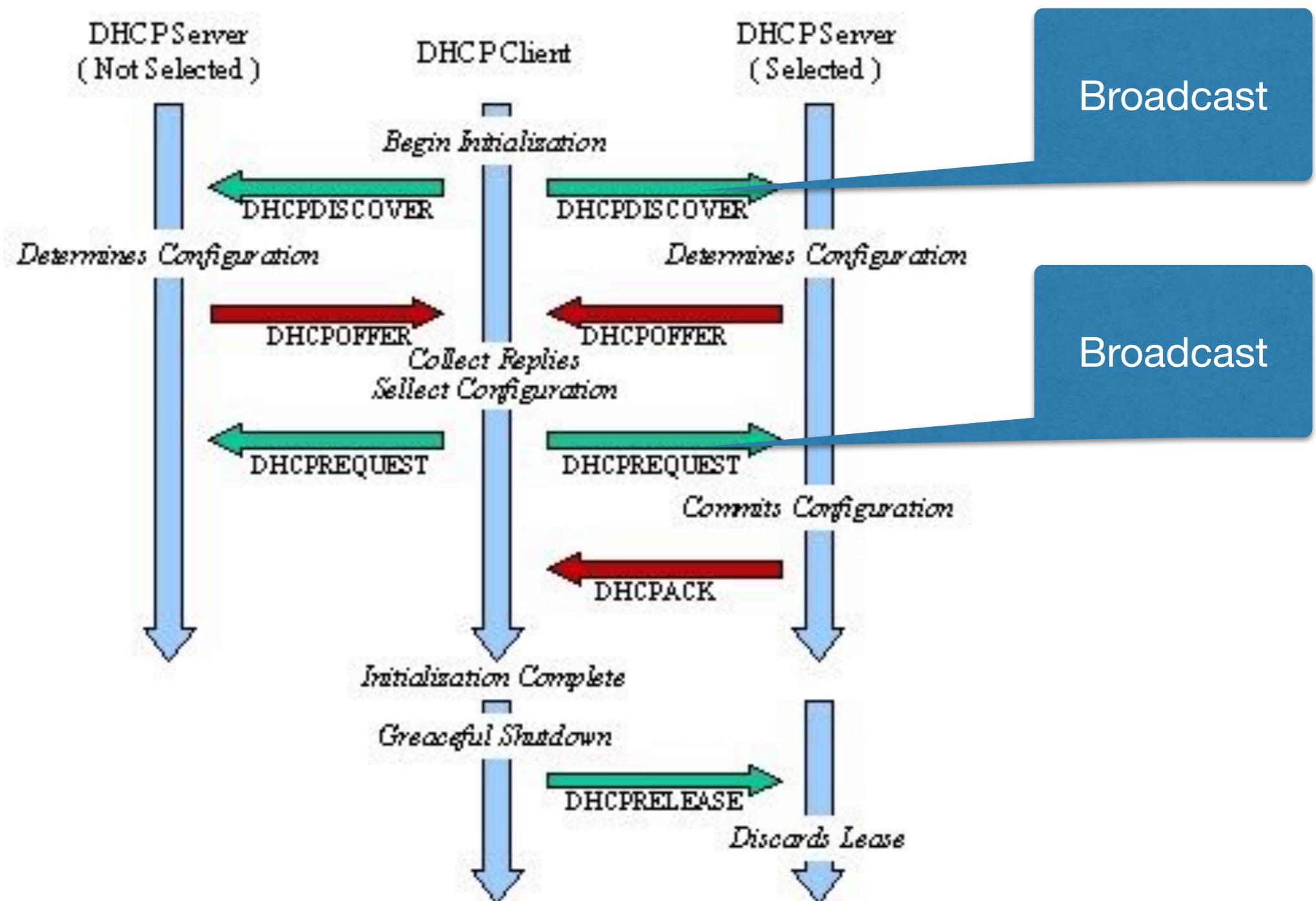
# DHCP

- Dynamic Host Configuration Protocol (RFC2131)
- Provides a means to **lease** a temporary IP number for a specified duration, as well as obtaining addresses mappings 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

# DHCP Initial Operation

- Client broadcasts a request for an IP number, including its MAC address or some other identifier (DHCPDISCOVER)
- Server(s) reserve an available IP number, and broadcast an offer of it with a lease time (how long IP number is valid for)(DHCPOFFER)
- Client chooses from amongst offers, and broadcasts a reply containing chosen IP number (DHCPREQUEST)
- Server that offered the IP number finishes reserving it and acknowledges (DHCPACK); other servers see that their offer has been declined and unreserve their offer (or wait a decent interval and free the reservation unilaterally)

# DHCP Initial Operation



# DHCPRQUEST

- Initially, a broadcast containing the IP number (and other information) that the client has decided to use.
- Received by DHCP servers:
  - If it matches what they offered, they know the client is using the offered IP number, so locks the offered IP number for the duration of the lease
  - If it does not match (eg, MAC address is no one they've made an offer to, or claimed IP number is different) then they do nothing

# DHCP Request

- But a client can also directly send a request to a known server, in order to renew a lease.
- Conventionally, renewal attempts start after half the lease has passed.

# DHCP Static v 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 that 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.

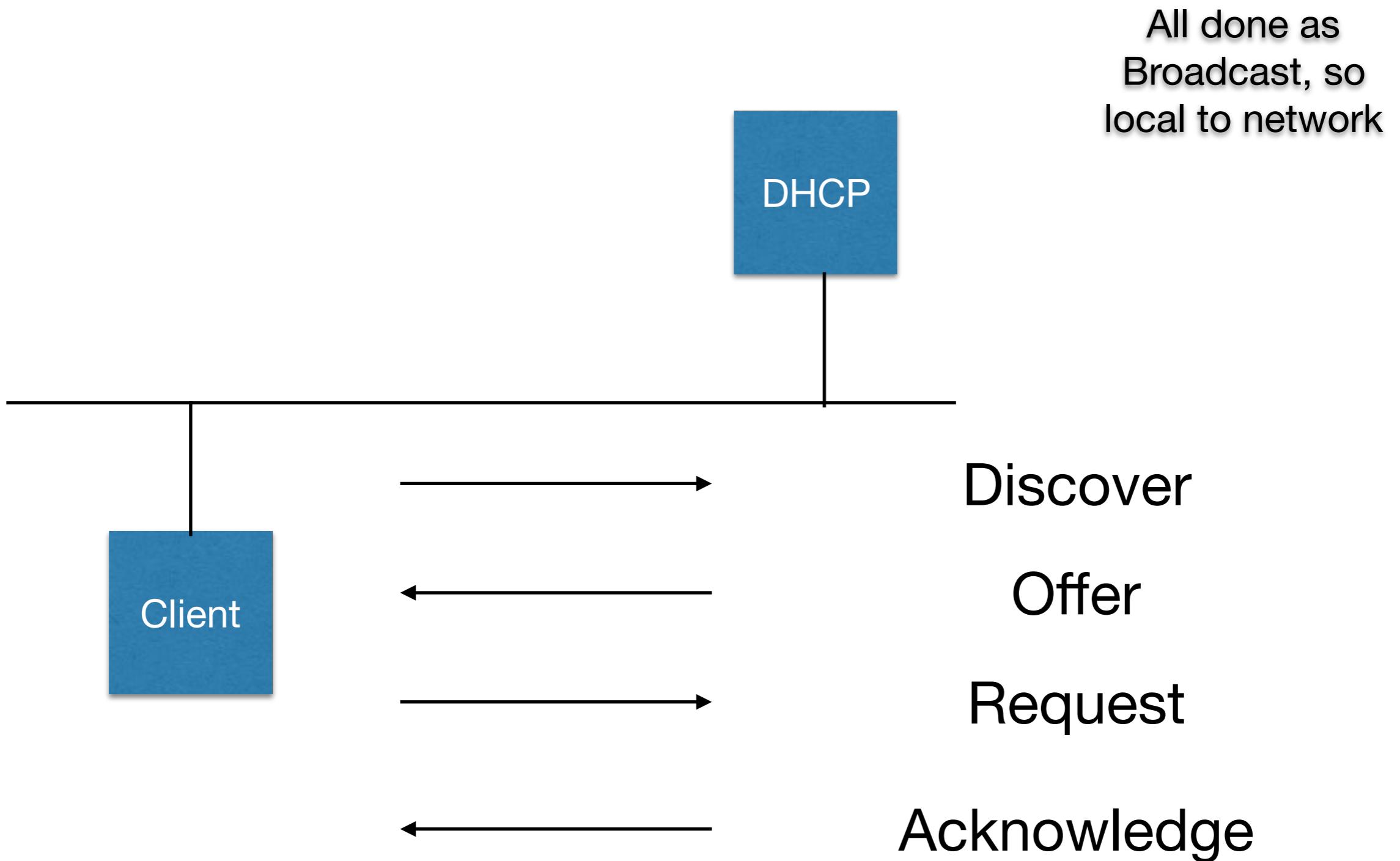
# DHCP Redundancy

- Loss of DHCP server will wipe out your network.
- But DHCP servers contain delicate state
- Simple home routers can be careless about this, and restart of router may require restart of client devices to get all state into agreement: hence typically use short lease times
- Choices include:
  - two DHCP servers managing disjoint pools and let client select (client IP numbers will change at random intervals)
    - you can have one server lag the other by a second to only be used in an emergency, which is a common trick
  - complex failover protocol so that both servers make the same offer and commit to the same lease (no standards, and implementations are rare)
  - Using higher-level redundancy protocols to have one virtual server (probably best).

# DHCP is a mess!

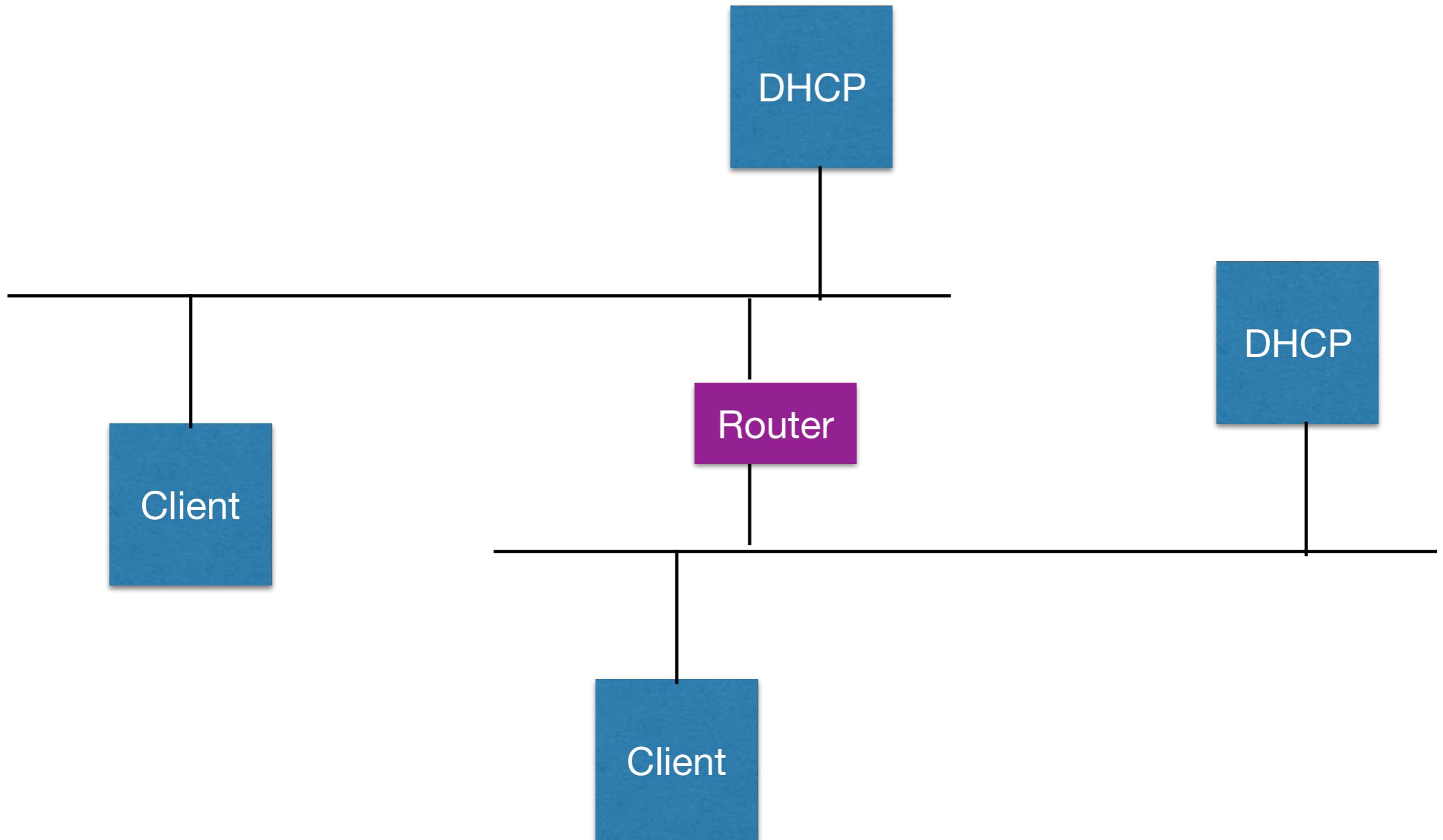
- DHCP failure is a serious problem
- DHCP is a nightmare to debug
- DHCP relays in complex networks are also very complex (look them up! take a headache tablet first!)

# DHCP Relaying



# Multiple Networks?

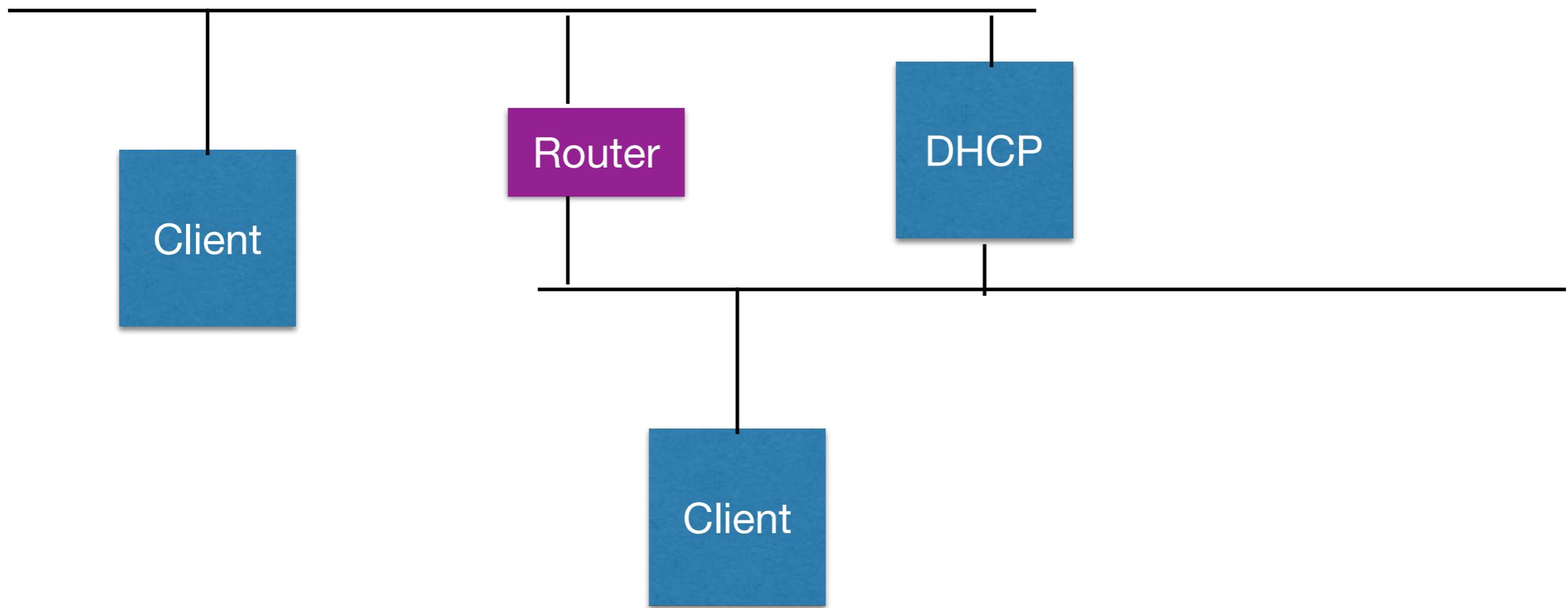
## Option1: Lots of DHCP



# Problems

- We discussed how vital DHCP servers are
- Now you have the problem of making all of them reliable, rather than just one
-

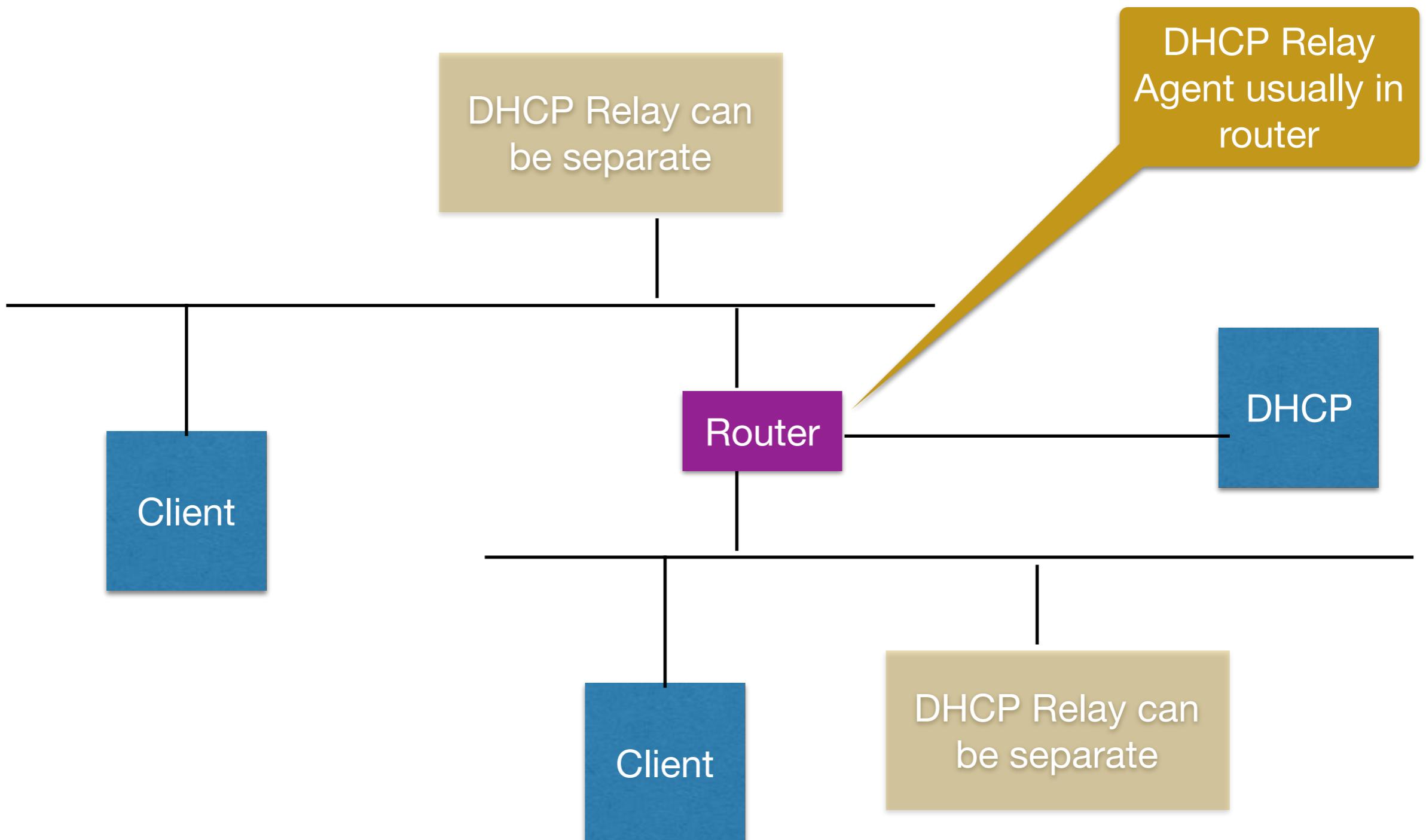
# Option 2: Multihome



# Problems

- Not scalable into very large networks
- Security nightmare (DHCP server is an implicit by-pass for any filtering that firewalls might do)

# Option 3: Relaying



# DHCP Relaying

- Relay agent on each network, usually embedded in router (but can be standalone)
  - Contains no state, can safely be replicated and have multiple running at the same time
- Operation is messy

# DHCP Relaying

- Relay agent hears a broadcast packet
- Is configured to send all requests to a known DHCP server, after filling in its own address
- DHCP server receives unicast DHCP packets, containing a relay address.
- Server sends responses back to the relay
- The relay then broadcasts them on the local network

# DHCP Relaying

- Relaying to a single server is the right solution in complex networks
  - DHCP server can be industrial-strength in a secure (power, aircon) data centre
- Makes complying with legal and regulatory requirements easier
- Nasty to debug

# DHCP Security

- Obvious DoS: rapidly request leases, and then abandon them, so that the servers have no more addresses to hand out.
  - Some servers have no, or unrealistic, release periods
  - Colleague reports a \*\*\*\*\* is doing this regularly (either deliberately or through messed-up laptop) on Chiltern Line trains south of Banbury

# DHCP Security

- And it's un-authenticated, so running a rogue DHCP server on a network which provides addresses of hacked DNS and router is powerful.
- And hard to stop: more in Network Security next semester.

# IPv6 address allocation

- DHCPv6: broadcast your MAC address, get an IPv6 configuration, as with DHCP (v4)
  - All the usual problems with DHCP
  - Much debate about logging requirements, however
- Preferred methods are static for servers and SLAAC, Stateless Address Auto Configuration, for everything else
  - Limited experience, so “preferred” may not be “right”

# IPv6 Router Advertisements

- In general, IPv4 routers (where a machine should send packets which are not destined for the local network) are configured either statically or by DHCP.
- IPv6 routers send router advertisements, which announce their existence and provide basic information about connectivity, both as periodic broadcasts and in response to solicitation messages
  - ICMP v6 messages 133 (solicitation) and 134 (advertisement)
  - At last support for DNS server information is starting to be discussed

# SLAAC for IPv6

- Routers broadcast router advertisement packets
- From these, a device can deduce the prefix, the /64 that identifies the network
- They then put their 48 bit MAC address, plus some other stuff, into the address and just use it, without any more formalities
  - Full details, including dealing with clashes, in RFC4862
- But...

# Problem #1 with SLAAC: DNS

- The device now has an address, but no details on things like DNS servers
- In dual-stack world, it can just use the IPv4 devices discovered with DHCP (dirty, but works)
- In single-stack world, intention is that there be standard multicast addresses for standard services, or that it's provided via NDP, or via DHCP “O” mode (next slide), or something.
  - Sadly none of these are universally implemented: another proof IPv6 deployment is still not really ready, making dual-stack (at least via RFC1918) almost essential

# IPv6 DHCP for configuration

- Router Advertisement contains options field
- Amongst other options are “M” and “O”.
- These control how addresses are allocated and managed

# IPv6 “Managed”

- If the router advertisement has the “M” flag set, the network is “Managed”.
- This means that clients must not use IPv6 SLAAC, and instead must use DHCPv6.
  - DHCPv6 will include DNS information

# IPv6 “Other”

- A nasty hack.
- If the “O” or “Other Information” flag is set, the IPv6 node gets its own IPv6 address (by SLAAC, or statically allocated, or some other mechanism) but still goes to the DHCPv6 server for other configuration data.
- DHCPv6 server doesn’t need to manage leases, so is essentially static.
- Best solution until DNS multicast is reliably available.
  - But apparently (hot off the press!) now broken in Android, just as it started working in iOS.

# Problem #2 with SLAAC

- In IPv4, your MAC address (which is unique to your machine) never leaves the local network
- In IPv6 with SLAAC, your MAC address is embedded in your IP number.
- This allows an observer to track you from network to network just by your temporary IP number.
- Arguably a privacy risk

# IPv6 Privacy

```
en0: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST>
mtu 1500
options=27<RXCSUM,TXCSUM,VLAN_MTU,TS04>
ether 00:26:bb:60:07:ce
inet6 fe80::226:bbff:fe60:7ce%en0 prefixlen 64 scopeid 0x4
inet 10.92.213.177 netmask 0xffffffff broadcast 10.92.213.255
inet6 2001:8b0:129f:a90f:0226:bbff:fe60:7ce prefixlen 64 autoconf
```

This page shows your IPv6 and/or IPv4 address

You are connecting with an **IPv6** Address of:

**2001:8b0:129f:a90f:226:bbff:fe60:7ce**

[IPv4 only Test](#)

[Normal Test](#)

[IPv6 only Test](#)

If the IPv6 only test shows "The page cannot be displayed" (Internet Explorer), "Server not found" (Firefox), any error or search page then you do not have working IPv6 connectivity. "Normal Test" shows which protocol your browser prefers when you have both IPv4 and IPv6 connectivity. This page should work even on computers with IPv6 only connectivity.

You can access this page with any of these easy to remember url's:

[ip4.me](#) (defaults to IPv4 only test)

[ip6.me](#)

[whatismyv6.com](#)

[whatismyipv6address.com](#)

# IPv6 Privacy

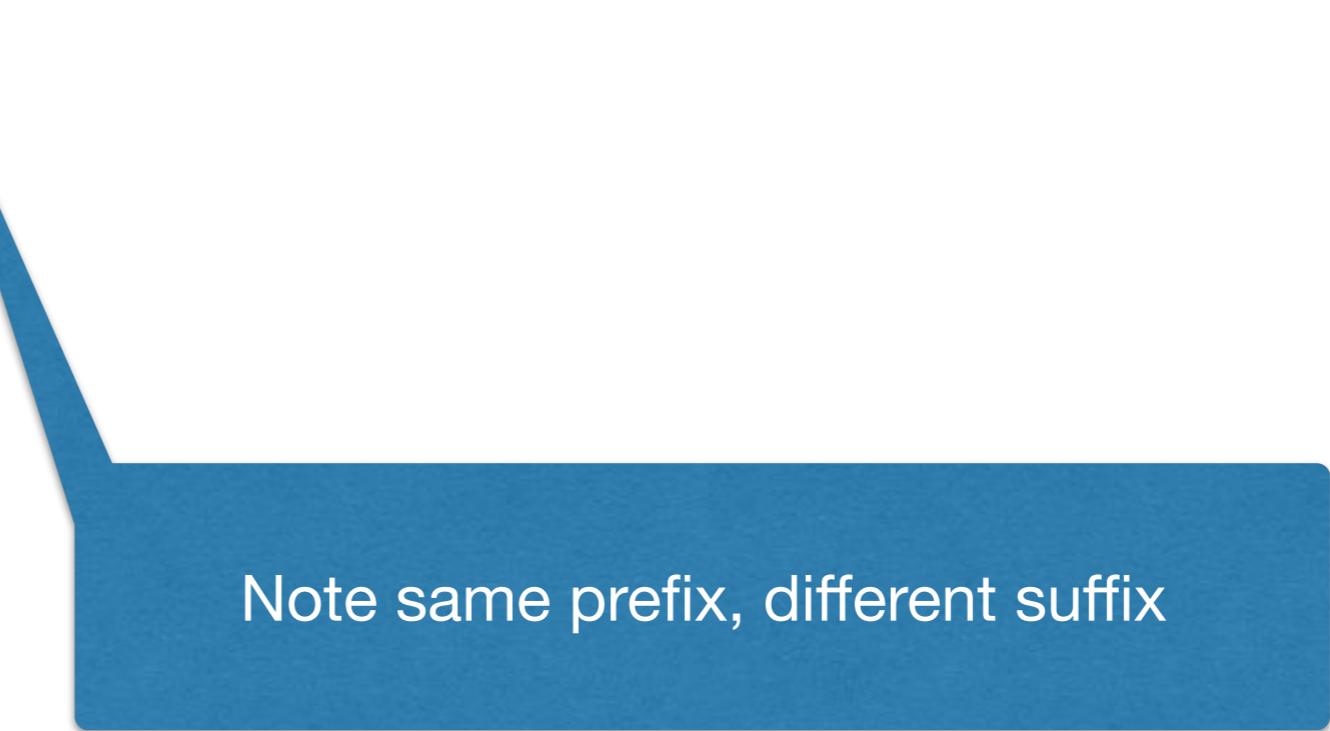
- In fact, this worry is probably over-stated
  - Implies an attacker who can observe packets on all the networks you visit, but cannot observe any authentication exchange which identify you.
  - If they know which networks to look at, they already know where you are
  - If they can observe packets on those networks, they can see your machine by MAC address
  - If they have access to authentication logs or similar, they can track you by network number (first 64 bits) plus your authentication information.

# RFC4941 Privacy

- Like SLAAC, but instead a random IPv6 address is chosen each time one is required (with mechanism for dealing with very unlikely clashes)
  - iPhones, Android and the like do this by default (which can be really annoying), getting a new address on every wake up from sleep
  - Linux, Windows, OSX have it as a choice (typically “on” for laptops, “off” for fixed machines) and acquire a new address whenever they change network.

# Privacy Addresses

```
options=27<RXCSUM, TXCSUM, VLAN_MTU, TS04>
ether 00:26:bb:60:07:ce
inet6 fe80::226:bbff:fe60:7ce%en0 prefixlen 64 scopeid 0x4
inet 10.92.213.177 netmask 0xffffffff00 broadcast 10.92.213.255
    inet6 2001:8b0:129f:a90f:3912:c2fc:cf01:6434 prefixlen 64
autoconf temporary
```



Note same prefix, different suffix

# RFC7217

- Designed to produce stable addresses on each network to which you connect.
- Hashes the network's prefix, the interface identity (name, MAC, etc), the optional Network ID (SSID, etc), a counter in case there is a clash and a secret key initialised at OS installation.
- $\text{RID} = F(\text{Prefix}, \text{Net\_Iface}, \text{Network\_ID}, \text{DAD\_Counter}, \text{secret\_key})$
- Preferable to RFC4941, because addresses remain the same on the same network (good for logging) ^

# On latest OSX bits

- Probably some other new OSes as well.
- Means that

```
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
ether 78:4f:43:90:d9:f8
inet6 fe80::145a:76e7:d93a:7f92%en0 prefixlen 64 secured scopeid 0x5
```

# Need to check iOS

- My iOS11 iPhone had 2001:8b0:129f:a90f:21f5:affa:eef:cbb1 for 24 hours, which is longer than it used to manage (a year ago, the address changed each time phone slept/woke). Similar longevity for addresses on wife's iPad (still iOS 10).
- But only 24 hours, to “secret” probably re-initialised on reboot, or something.

# Networks: TCP and UDP

Transport Layer

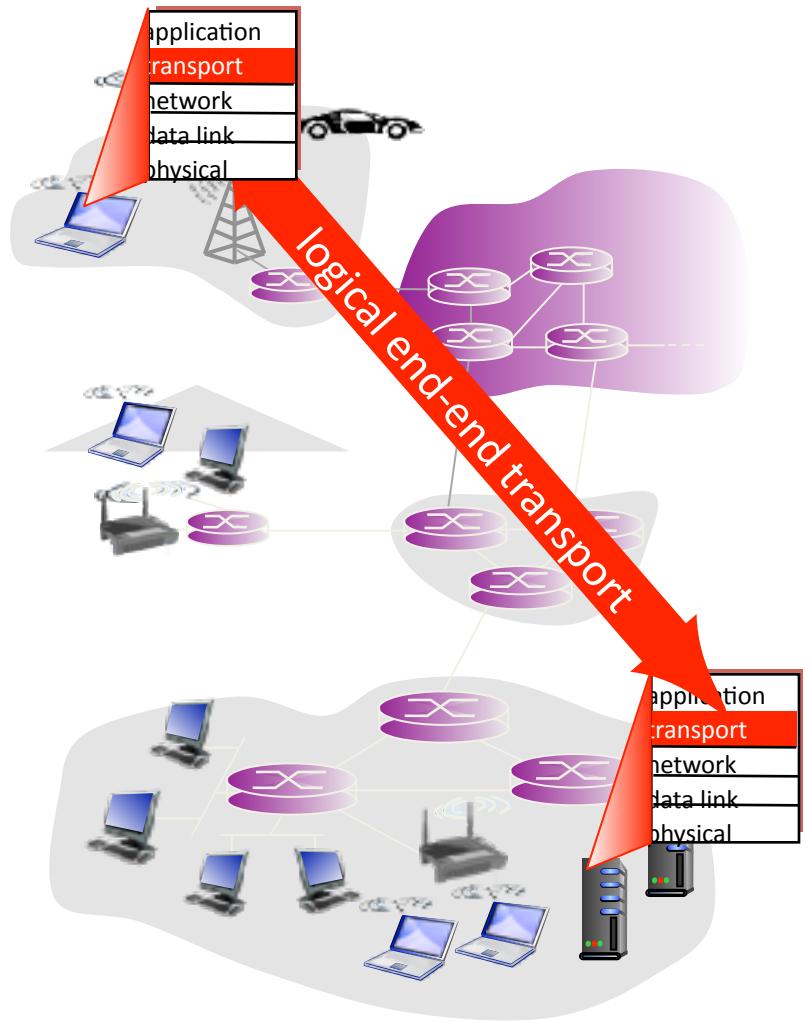
Dr Mirco Musolesi

Dr Ian Batten



# Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into *segments*, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



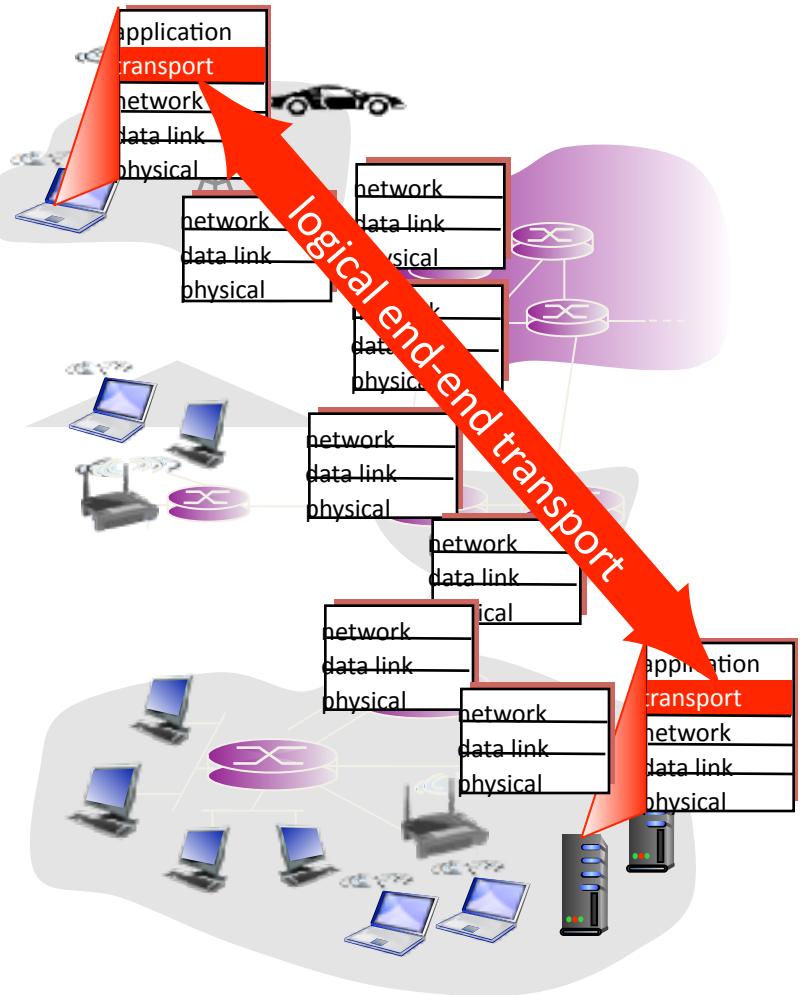
# Transport vs. network layer

- **network layer**: logical communication between **hosts**
- **transport layer**: logical communication between **processes**
  - relies on, enhances, network layer services



# Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



# Multiplexing/demultiplexing

Demultiplexing at rcv host:

delivering received segments  
to correct socket

Multiplexing at send host:

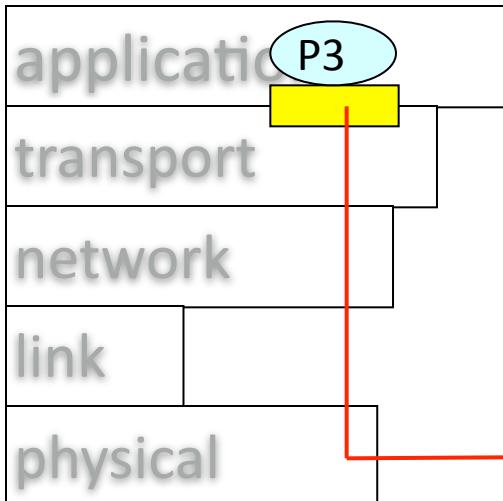
gathering data from multiple  
sockets, enveloping data with  
header (later used for  
demultiplexing)



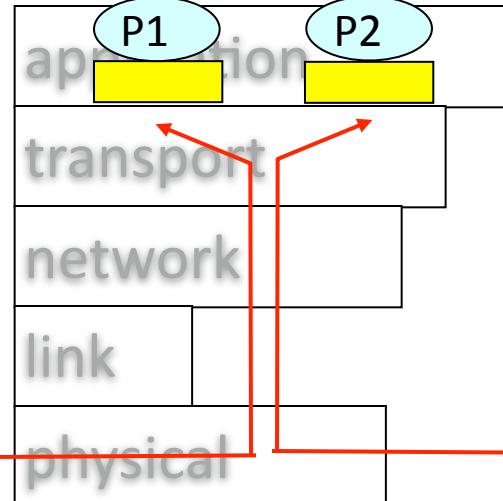
= socket



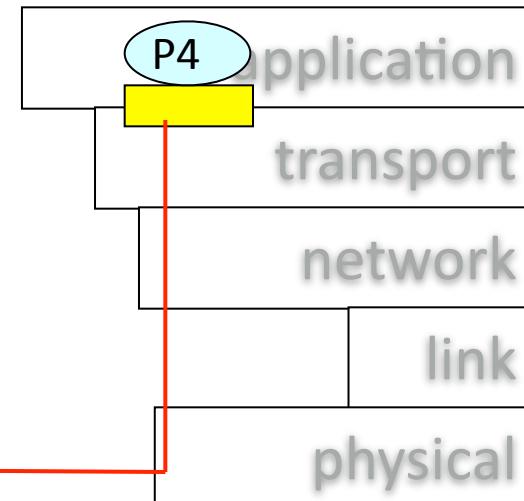
= process



host 1



host 2

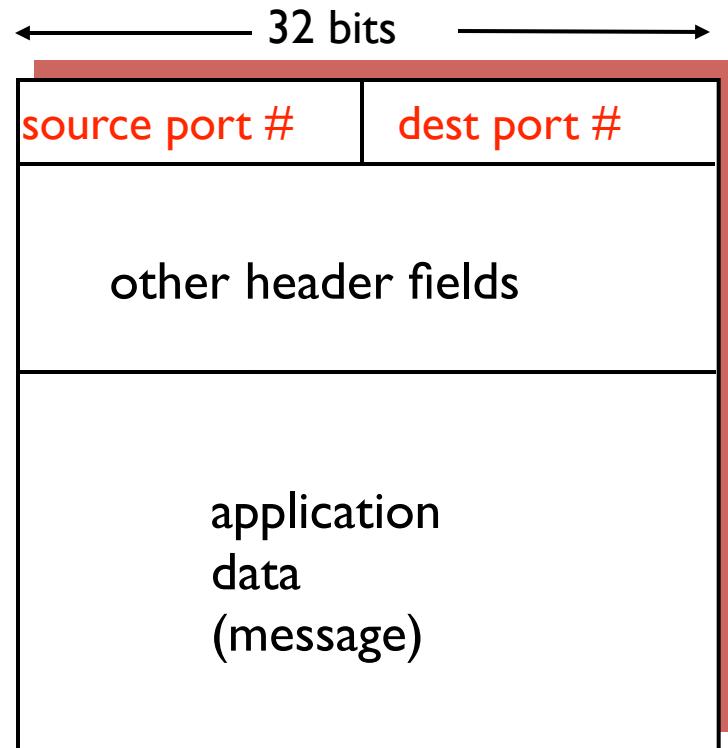


host 3



# How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

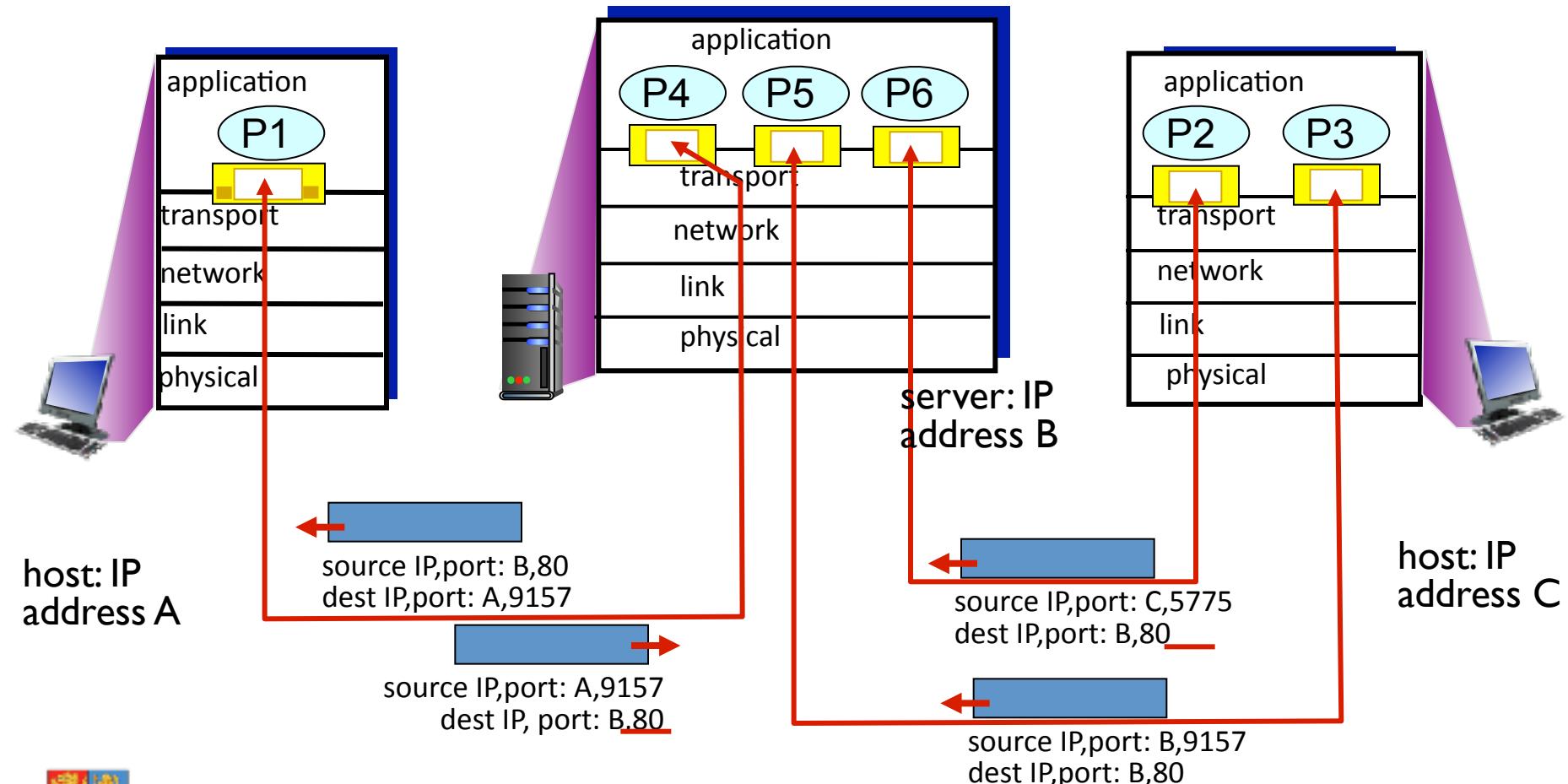


# Connection-oriented demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses all four values to direct segment to appropriate socket
- server host may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

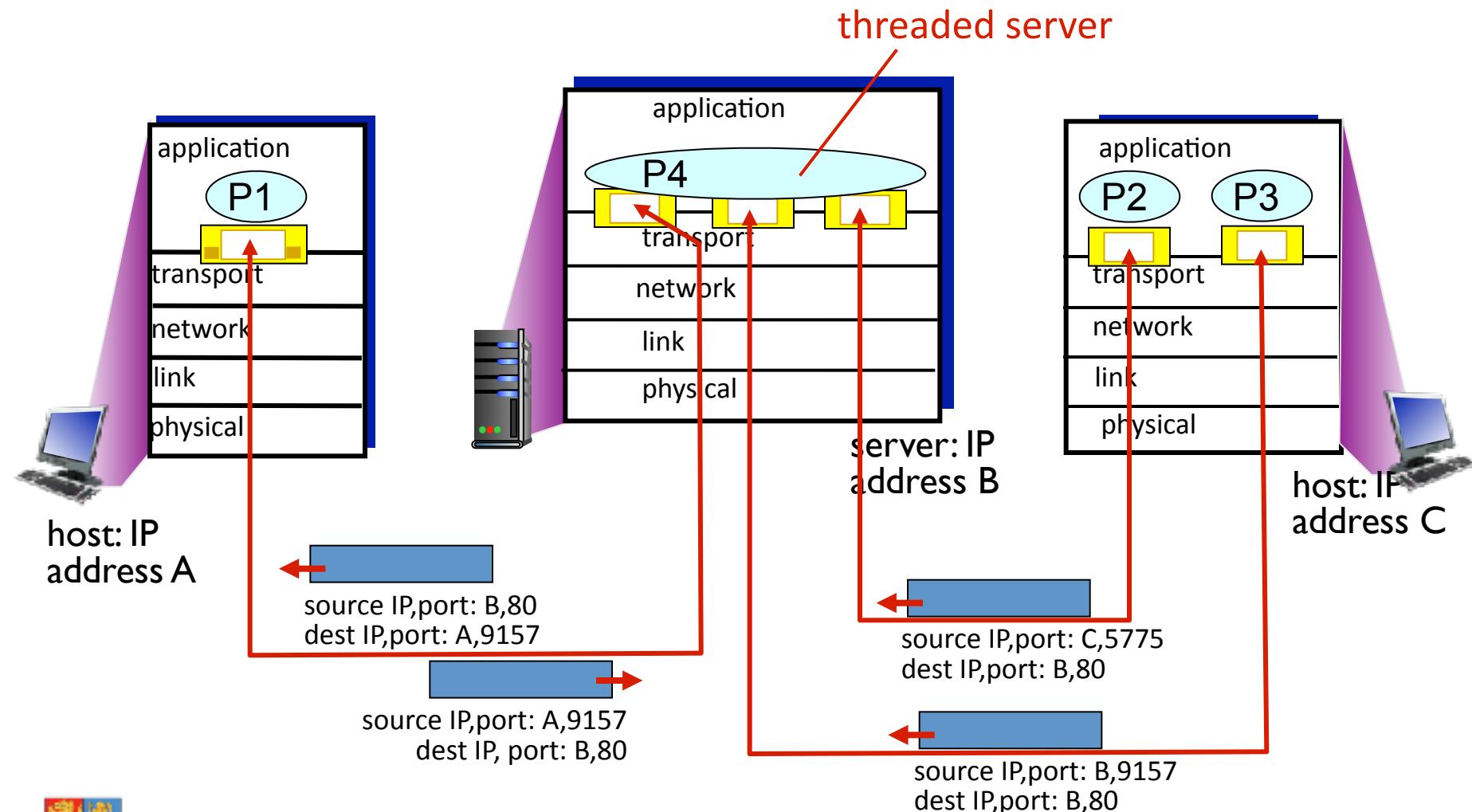


# Connection-oriented demux: example



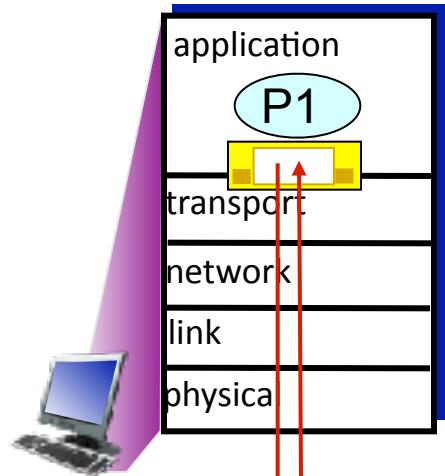
three segments, all destined to IP address: B,  
dest port: 80 are demultiplexed to *different* sockets

# Connection-oriented demux: example

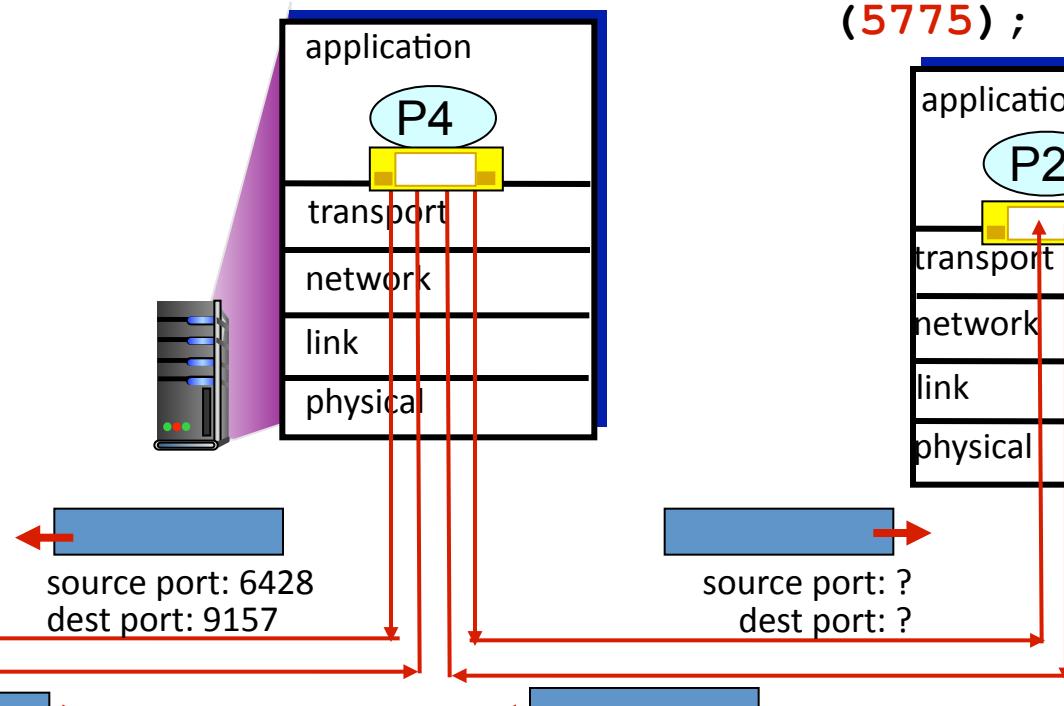


# Connectionless demux: example

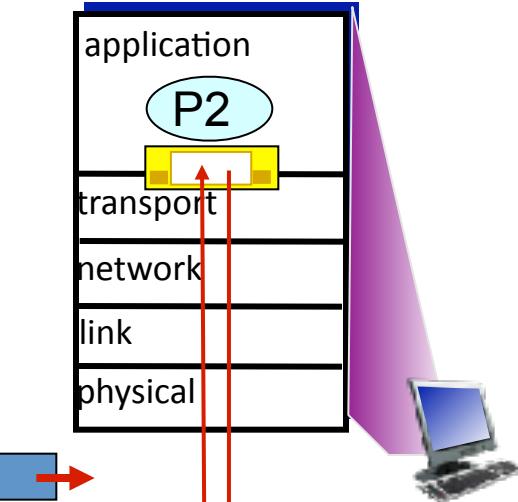
```
DatagramSocket  
mySocket2 = new  
DatagramSocket  
(9157);
```



```
DatagramSocket  
serverSocket = new  
DatagramSocket  
(6428);
```



```
DatagramSocket  
mySocket1 = new  
DatagramSocket  
(5775);
```



# Connectionless demultiplexing

- Create sockets with port numbers:

```
DatagramSocket mySocket1 = new  
DatagramSocket(12534);
```

- UDP socket identified by two-tuple:

(dest IP address, dest port number)

- When host receives UDP segment:
  - checks destination port number in segment
  - directs UDP segment to socket with that port number
- IP datagrams with different source IP addresses and/or source port numbers can be directed to same socket



# Connectionless demultiplexing

- *recall:* created socket has host-local port #:

```
DatagramSocket mySocket1  
= new DatagramSocket(12534);
```

- when host receives UDP segment:

- checks destination port # in segment
- directs UDP segment to socket with that port #

when creating datagram to send into UDP socket, you must specify

- destination IP address
- destination port #

IP datagrams with *same dest. port #*, but different source IP addresses and/or source port numbers will be directed to *same socket* at dest

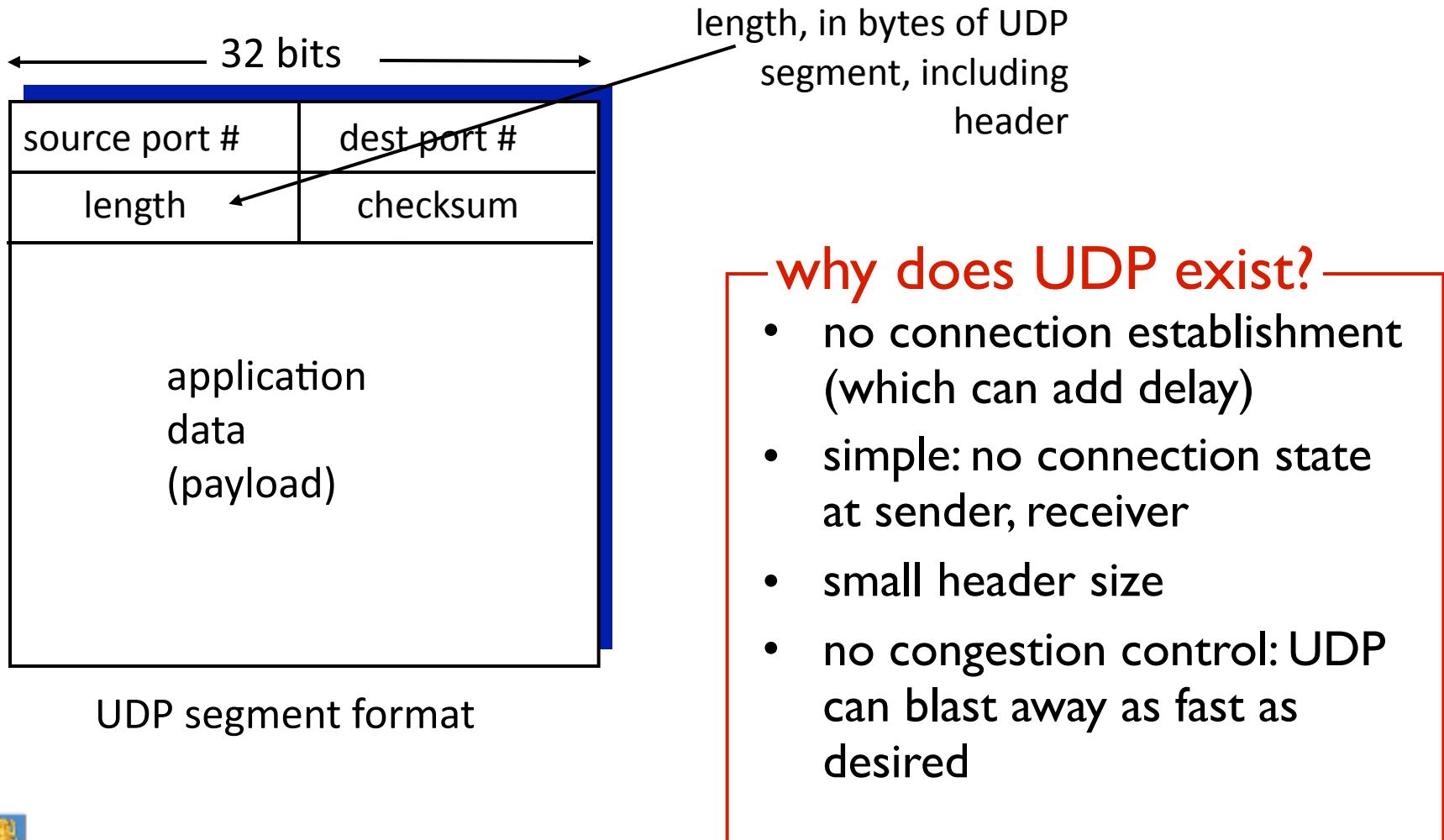


# UDP: User Datagram Protocol [RFC 768]

- “no frills,” “bare bones” Internet transport protocol
  - “best effort” service, UDP segments may be:
    - lost
    - delivered out-of-order to app
  - *connectionless*:
    - no handshaking between UDP sender, receiver
    - each UDP segment handled independently of others
- ❖ UDP use:
- streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
- ❖ reliable transfer over UDP:
- add reliability at application layer
  - application-specific error recovery!



# UDP: segment header



# UDP checksum

**Goal:** detect “errors” (e.g., flipped bits) in transmitted segment

sender:

- treat segment contents, including header fields, as sequence of 16-bit integers, with checksum field set to zero
- **“Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.”** (RFC 768 – that's all it says!)
- sender puts checksum value into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:



# Internet checksum: example

example: add two 16-bit integers

$$\begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \\ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 1 \\ \hline \end{array}$$

wraparound

$$\begin{array}{r} 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \\ \hline \end{array}$$

sum       $1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 1 \ 0 \ 0$   
checksum       $0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1$

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

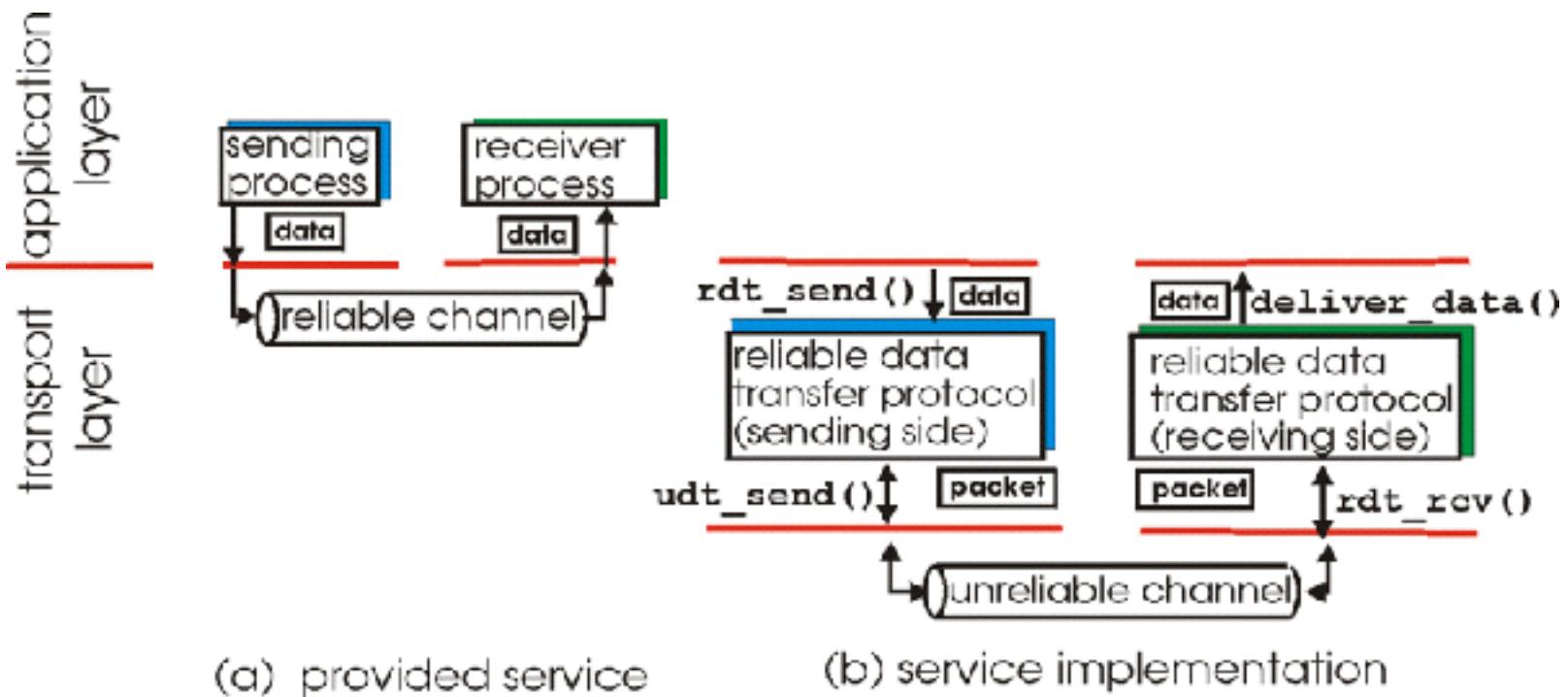
# Implementation Issues

- The design is very 16-bit: IEN 45 in 1978 is all about pdp11s, and the non-16-bit machines are actually 36 bit (pdp10s).
  - 32 bit add (or 64 bit add) is expensive on such boxes
- You can get the effect of the end wraparound by summing 16 bit quantities in a 32 bit word and then adding the top and bottom halves together repeatedly until the top half is zero.
- RFC1071 has the gory details: it took until 1988 to publish
  - And is itself full of ancient history, like Cray assembler
- Allegedly, at the first interworking “bake off” **no** implementations actually interworked...



# Principles of reliable data transfer

- important in application, transport, link layers

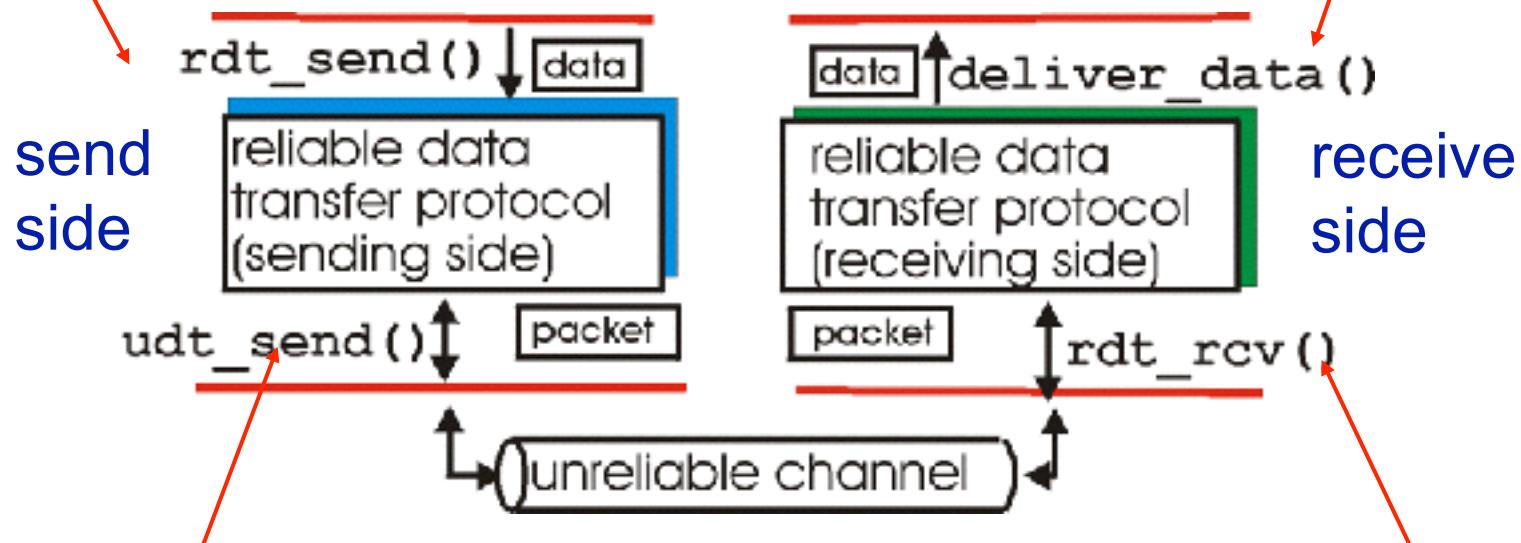


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)



# Reliable data transfer: getting started

**rdt\_send()** : called from above, (e.g., by application). Passed data to deliver to receiver upper layer



**udt\_send()** : called by rdt, to transfer packet over unreliable channel to receiver

**deliver\_data()** : called by rdt to deliver data to upper

# In a perfect world...

- Channels are perfectly reliable:
  - No errors (i.e., flipped bits)
  - No packet loss
  - Packets received by the receiver in order
- If a channel is perfectly reliable it is sufficient to send and receive data: you don't need any additional mechanisms such as acknowledgements
- But the world is not perfect and errors happen...
- ...and you need to recover from errors!



# Acknowledgement mechanisms

- Let us assume first that the channel might introduce errors (i.e., corrupted packets) but packets are not lost
- Underlying channel may flip bits in packet
  - Checksum is used to detect bit in errors
- The key problem is how to recover from errors
- Two mechanisms
  - *acknowledgements (ACKs)*: receiver explicitly tells sender that the packet has received correctly

OR

- *negative acknowledgements (NAKs)*: receiver explicitly tells sender that packet had errors
  - Sender retransmits packet on receipt of NAK



# What happens if ACKs/NACKs get corrupted?

- You can have errors in packet (data) transmission but also in ACK/NACK transmission
- And so what happens if ACKs/NACKs get corrupted?
  - Essentially, the sender does not know what happens at the receiver
- One solution is to simply retransmit, **but** you can't simply do that, since you might have **duplicates** at the receiver
- Problem: how to handle duplicates



# Note the checksums are weak

- The IP checksum is weak, and won't reliably detect multiple bit flips, depending on where they happen.
- Usually a channel that unreliable won't pass traffic successfully, and the TCP protocol itself will collapse
- But If you are moving data whose contents matter bit-for-bit, you need to use a stronger checksum over the whole thing (SHA256 is good)



# Handling Duplicates

- In order to handle duplicates, every packet should have a **sequence number** that identifies that packet uniquely
- The receiver discards (i.e., does not deliver the packet to the layer above) the duplicate packets

And now let's consider the general case:  
channels **do** lose packets...

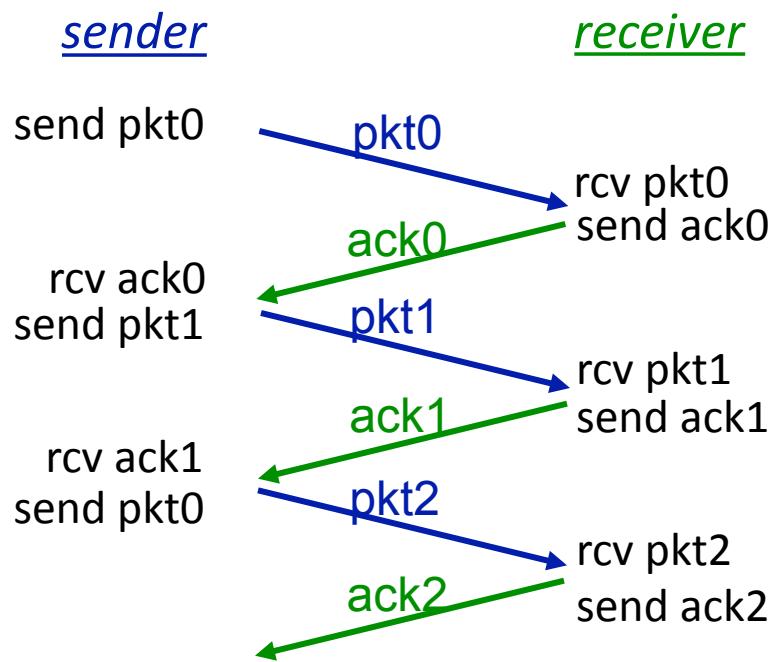


# And a channel can also lose packets...

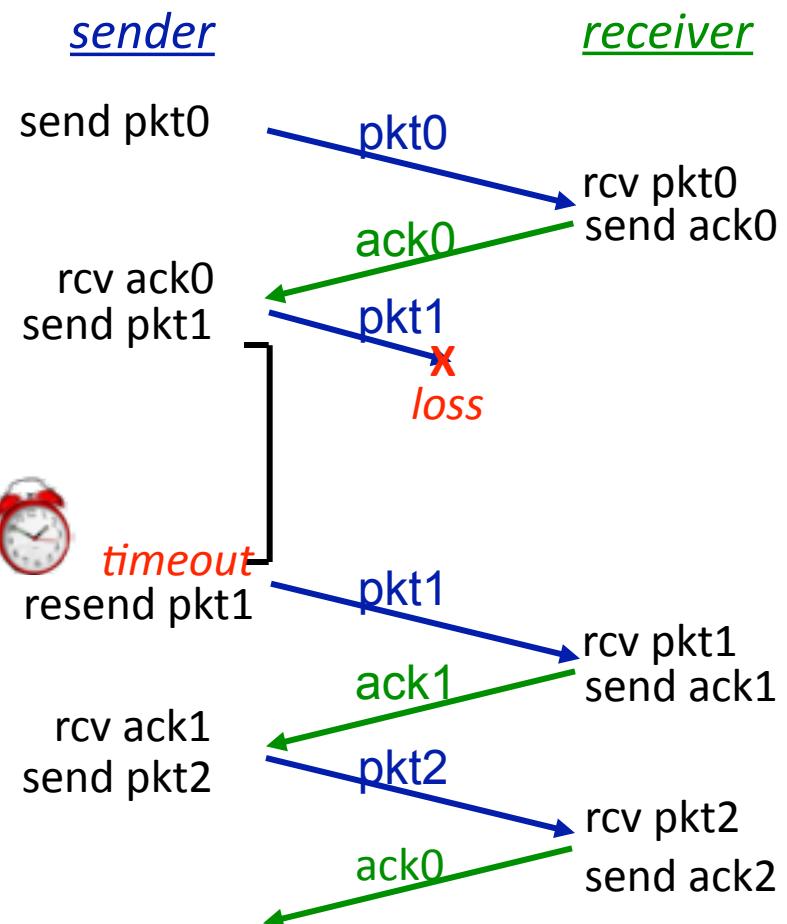
- Underlying channel can also lose packets (data,ACKs)
  - checksum, sequence numbers ,ACKs, retransmissions are of help, but not enough...
- Sender waits “reasonable” amount of time for ACK
  - retransmits if no ACK received in this time
  - if pkt (or ACK) just delayed (not lost):
    - retransmission will be duplicate, but seq. #'s already handles this
    - receiver must specify seq # of packets being ACKed
- This mechanism requires countdown timer



# ACKs and time-outs in action



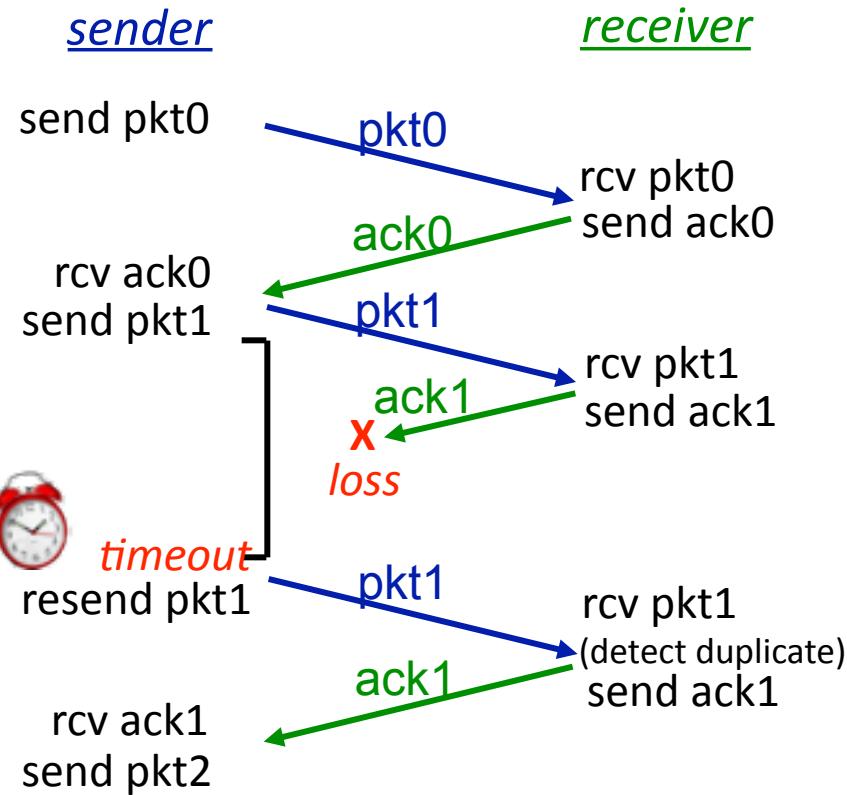
(a) no loss



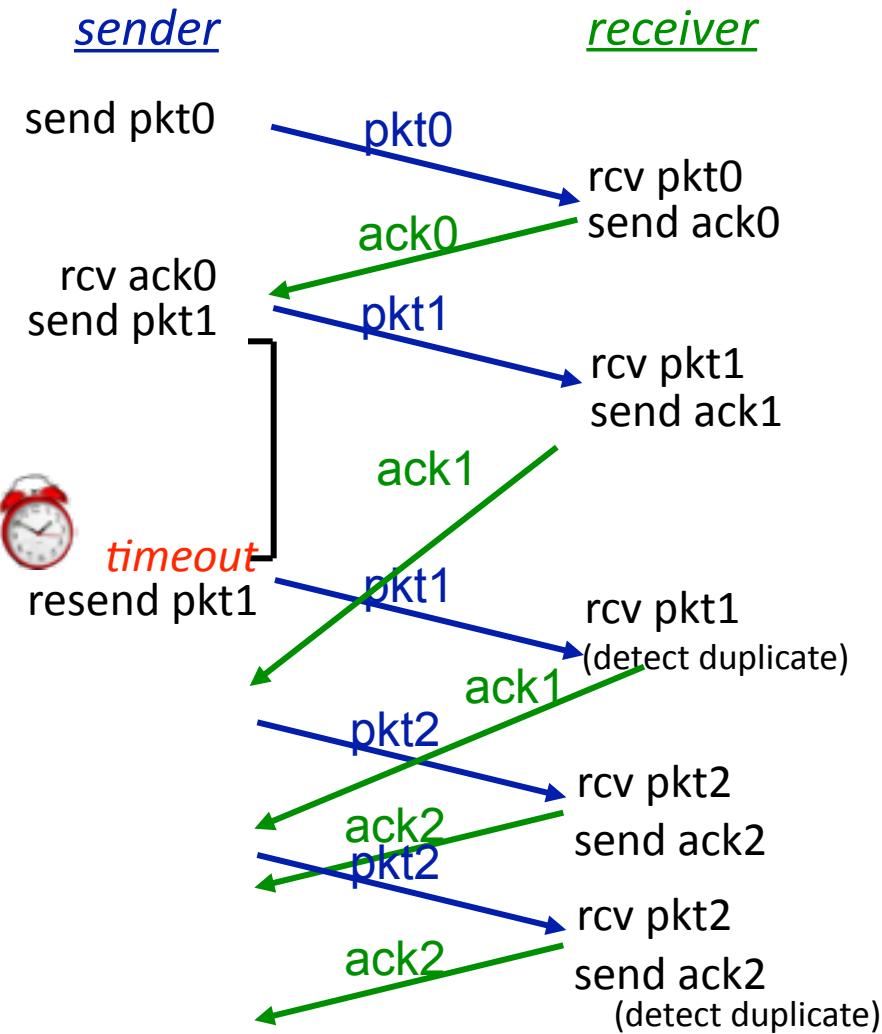
(b) packet loss



# ACKs and time-outs in action



(c) ACK loss



(d) premature timeout/ delayed ACK

# Latency kills this method

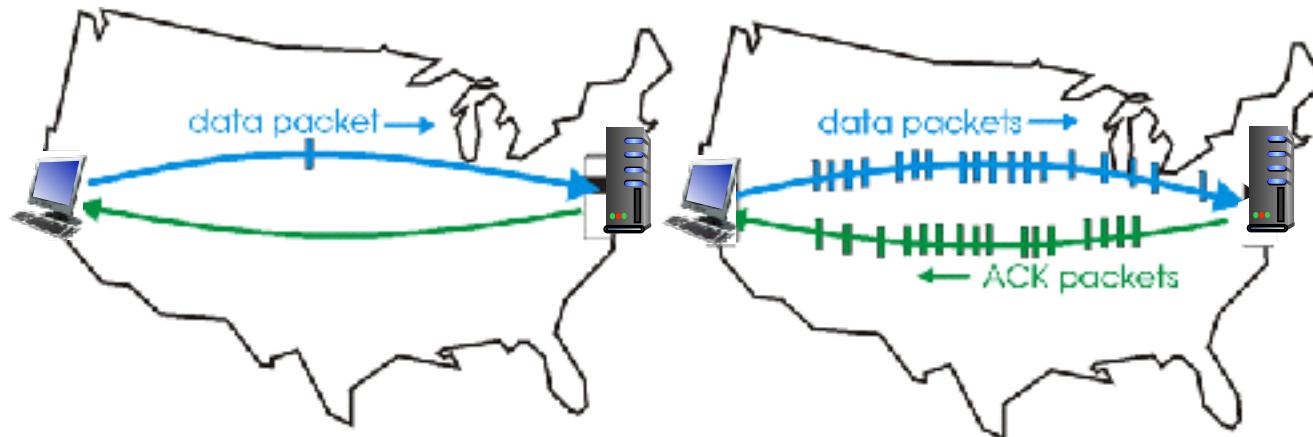
- Suppose we only allow one packet outstanding at a time
- Speed of light is  $3 \times 10^8 \text{ms}^{-1}$ .
- Plenty of paths are  $3 \times 10^6 \text{m}$  (short transatlantic path to east coast).
- So 20ms round trip time, assuming infinitely fast routers and computers.
- 50 packets per second @1500bytes/packet = 75KB/sec ~ **600Kb/sec** maximum.



# Pipelined protocols

**pipelining:** sender allows multiple, “in-flight”, yet-to-be-acknowledged packets

- range of sequence numbers must be increased
- buffering at sender and/or receiver



- two generic forms of pipelined protocols: *go-Back-N*, *selective repeat*



# Pipelined protocols: overview

## Go-back-N:

- sender can have up to N unacked packets in pipeline
- receiver only sends **cumulative ack**
  - doesn't ack packet if there's a gap
- sender has timer for oldest unacked packet
  - when timer expires, retransmit *all* unacked packets

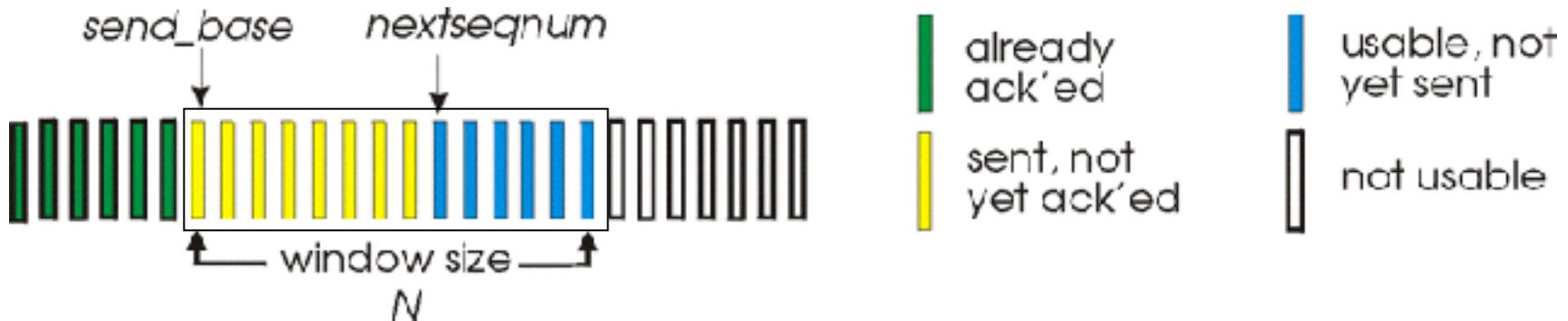
## Selective Repeat:

- sender can have up to N unack'd packets in pipeline
- rcvr sends ***individual ack*** for each packet
- sender maintains timer for each unacked packet
  - when timer expires, retransmit only that unacked packet



# Go-Back-N: sender

- k-bit sequence number in packet header
- “window” of up to N, consecutive unacked packets allowed



- ❖ ACK(n):ACKs all pkts up to, including sequence number n - **“cumulative ACK”**
  - may receive duplicate ACKs (see receiver)
- ❖ timer for oldest in-flight pkt
- ❖  $\text{timeout}(n)$ : retransmit packet n and all higher sequence number packets in window



# Go-Back-N: receiver

ACK-only: always send ACK for correctly-received packet with highest *in-order* sequence number

- may generate duplicate ACKs
- need only remember **expectedseqnum**
- out-of-order packets:
  - discard (don't buffer): *no receiver buffering!*
  - re-ACK packets with highest in-order sequence number



# Go-Back-N in action

sender window (N=4)

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

sender

receiver

send pkt0

send pkt1

send pkt2

send pkt3

(wait)

rcv ack0, send pkt4

rcv ack1, send pkt5

ignore duplicate ACK

*pkt 2 timeout*

send pkt2

send pkt3

send pkt4

send pkt5

receive pkt0, send ack0

receive pkt1, send ack1

receive pkt3, discard,  
(re)send ack1

receive pkt4, discard,  
(re)send ack1

receive pkt5, discard,  
(re)send ack1

rcv pkt2, deliver, send ack2

rcv pkt3, deliver, send ack3

rcv pkt4, deliver, send ack4

rcv pkt5, deliver, send ack5

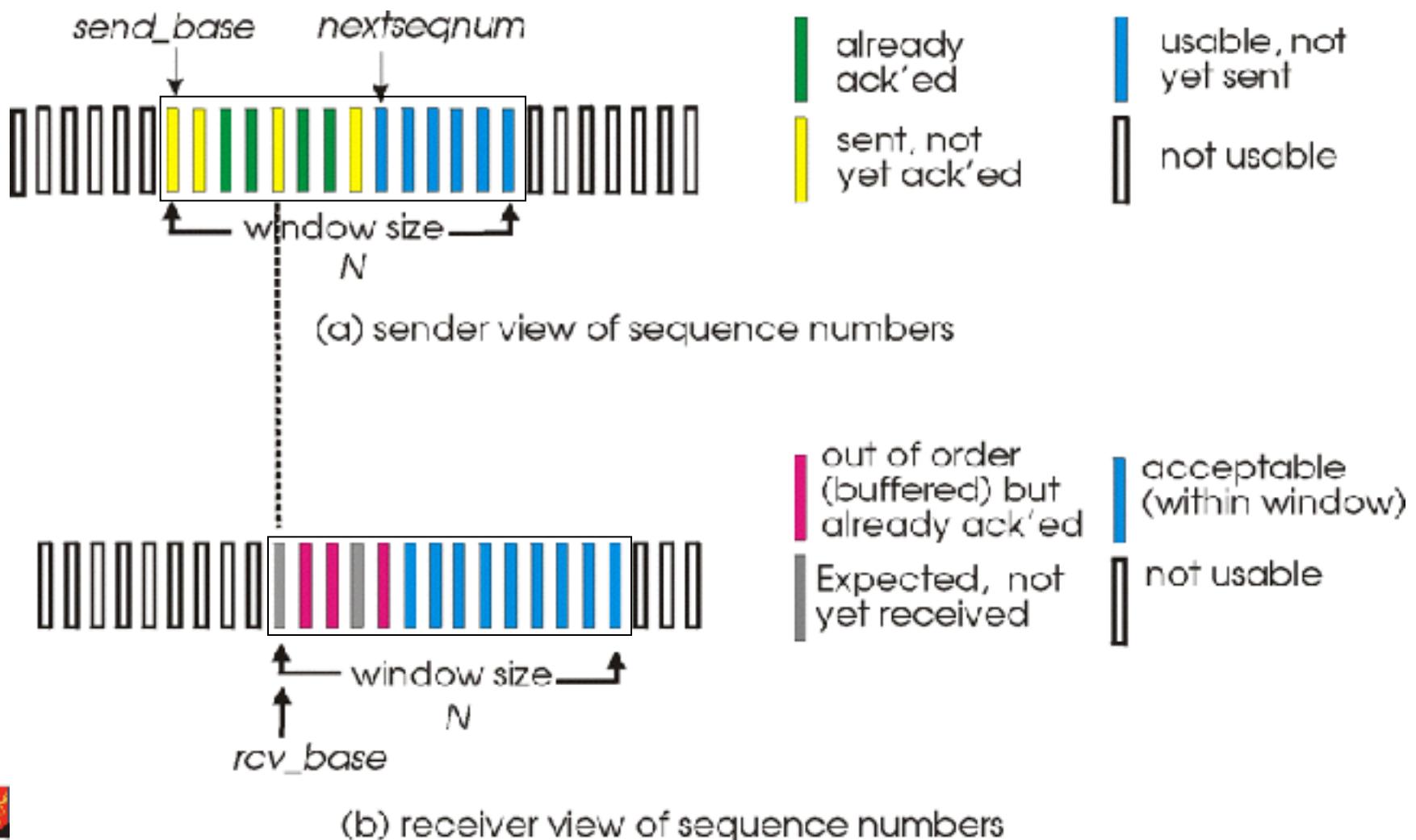


# Selective repeat

- Receiver *individually* acknowledges all correctly received packets
  - Buffers packets, as needed, for eventual *in-order delivery* to upper layer
- Sender only resends packets for which ACK not received
  - sender timer for each unACKed packet
- **sender window**
  - $N$  consecutive sequence numbers
  - limits sequence numbers of sent, unACKed pkts



# Selective repeat: sender, receiver windows



# Selective repeat

## sender

### data from above:

- if next available sequence number in window, send packet

### timeout(n):

- resend pkt n, restart timer

### ACK(n) in $[sendbase, sendbase+N]$ :

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq number

## receiver

### pkt n in $[rcvbase, rcvbase+N-1]$

- ❖ send ACK(n)
- ❖ out-of-order: buffer
- ❖ in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

### pkt n in $[rcvbase-N, rcvbase-1]$

- ❖ ACK(n)

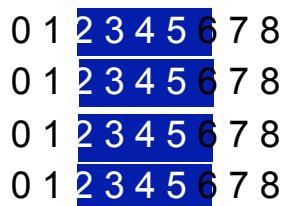
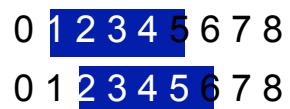
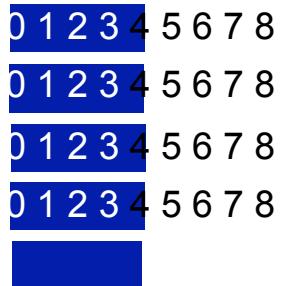
### otherwise:

- ❖ ignore



# Selective repeat in action

sender window (N=4)



sender

send pkt0  
send pkt1  
send pkt2  
send pkt3  
(wait)  
rcv ack0, send pkt4  
rcv ack1, send pkt5  
record ack3 arrived  
pkt 2 timeout  
send pkt2  
record ack4 arrived  
record ack4 arrived

receiver

receive pkt0, send ack0  
receive pkt1, send ack1  
receive pkt3, buffer,  
send ack3  
receive pkt4, buffer,  
send ack4  
receive pkt5, buffer,  
send ack5  
rcv pkt2; deliver pkt2,  
pkt3, pkt4, pkt5; send ack2

*Q: what happens when ack2 arrives?*



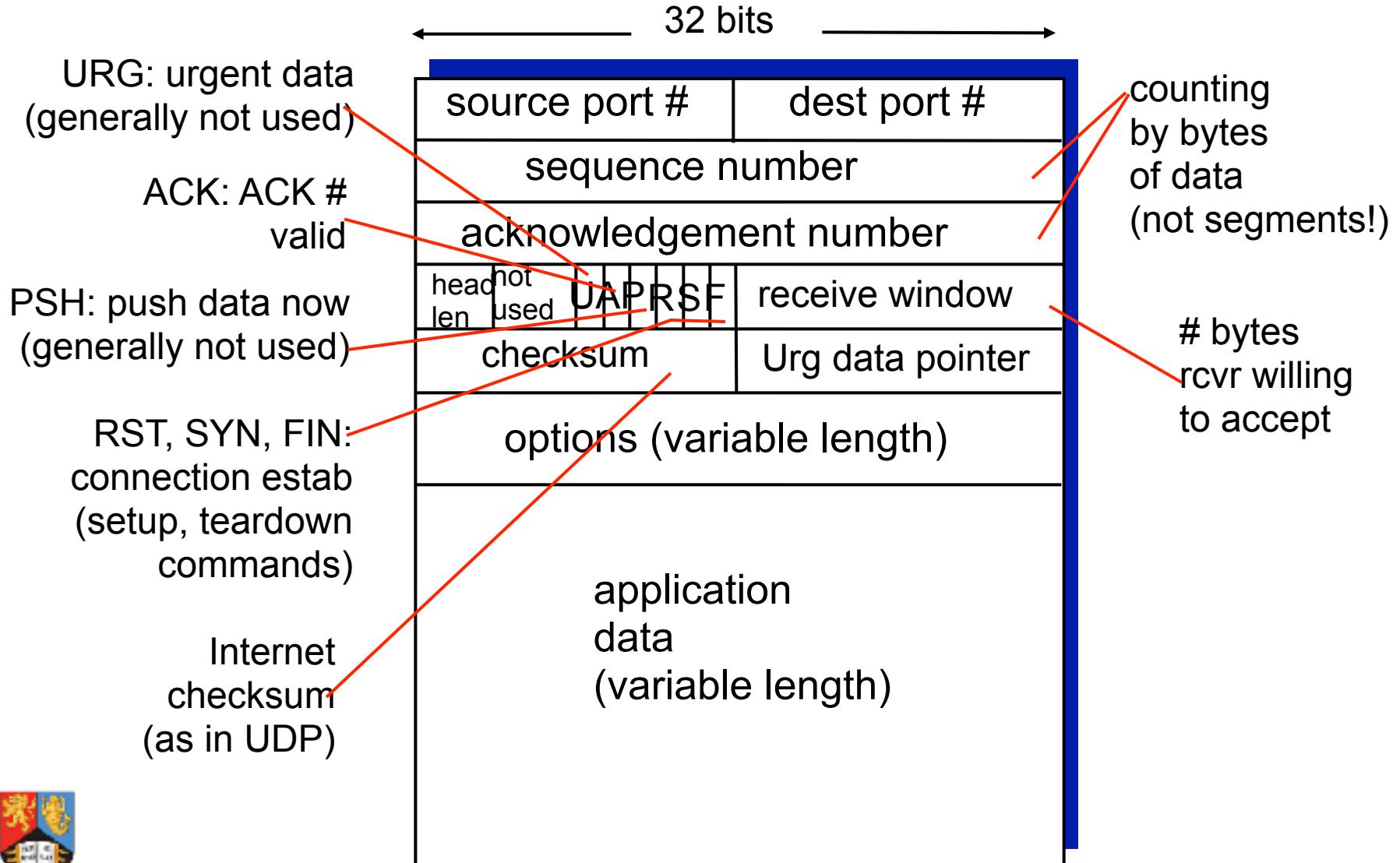
# TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- **point-to-point:**
  - one sender, one receiver
- **reliable, in-order byte stream:**
  - no “message boundaries”
- **pipelined:**
  - TCP congestion and flow control set window size
- **full duplex data:**
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- **connection-oriented:**
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- **flow controlled:**
  - sender will not overwhelm receiver



# TCP segment structure



# TCP seq. numbers, ACKs

## sequence numbers:

- byte stream “number” of first byte in segment’s data

## acknowledgements:

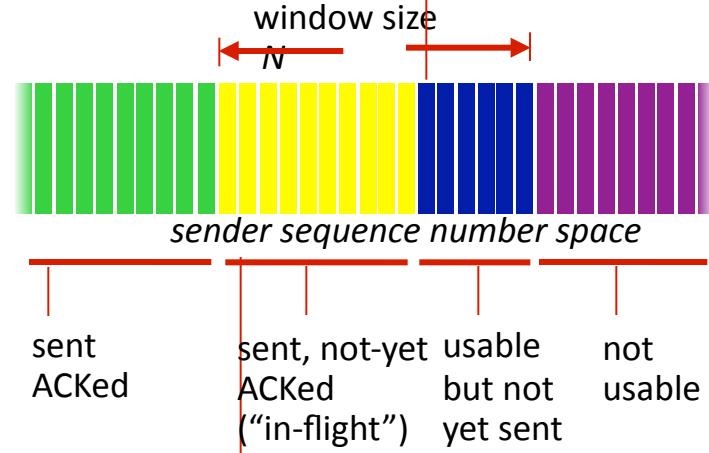
- seq # of next byte expected from other side
- cumulative ACK

**Q:** how receiver handles out-of-order segments

- A: TCP spec doesn’t say, – up to implementor

outgoing segment from sender

source port #	dest port #
sequence number	
acknowledgement number	
	rwnd
checksum	urg pointer

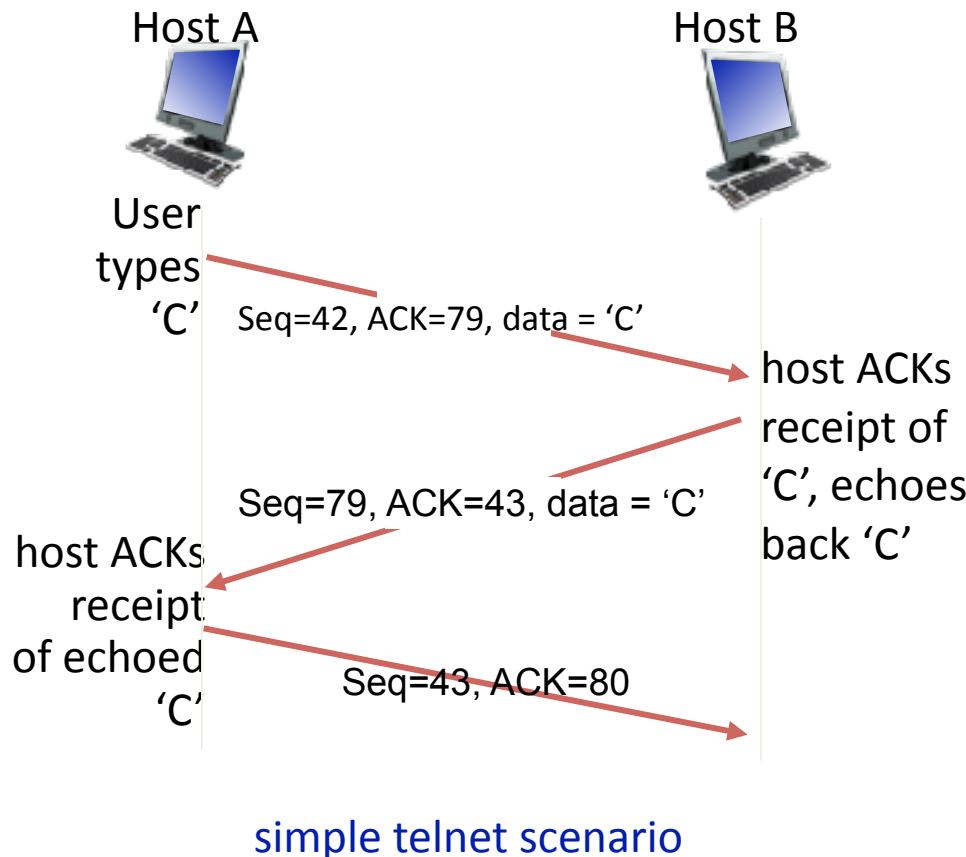


incoming segment to sender

source port #	dest port #
sequence number	
acknowledgement number	
A	rwnd
checksum	urg pointer



# TCP seq. numbers, ACKs



# TCP round trip time, timeout

**Q:** how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- *too short*: premature timeout, unnecessary retransmissions
- *too long*: slow reaction to segment loss

**Q:** how to estimate RTT?

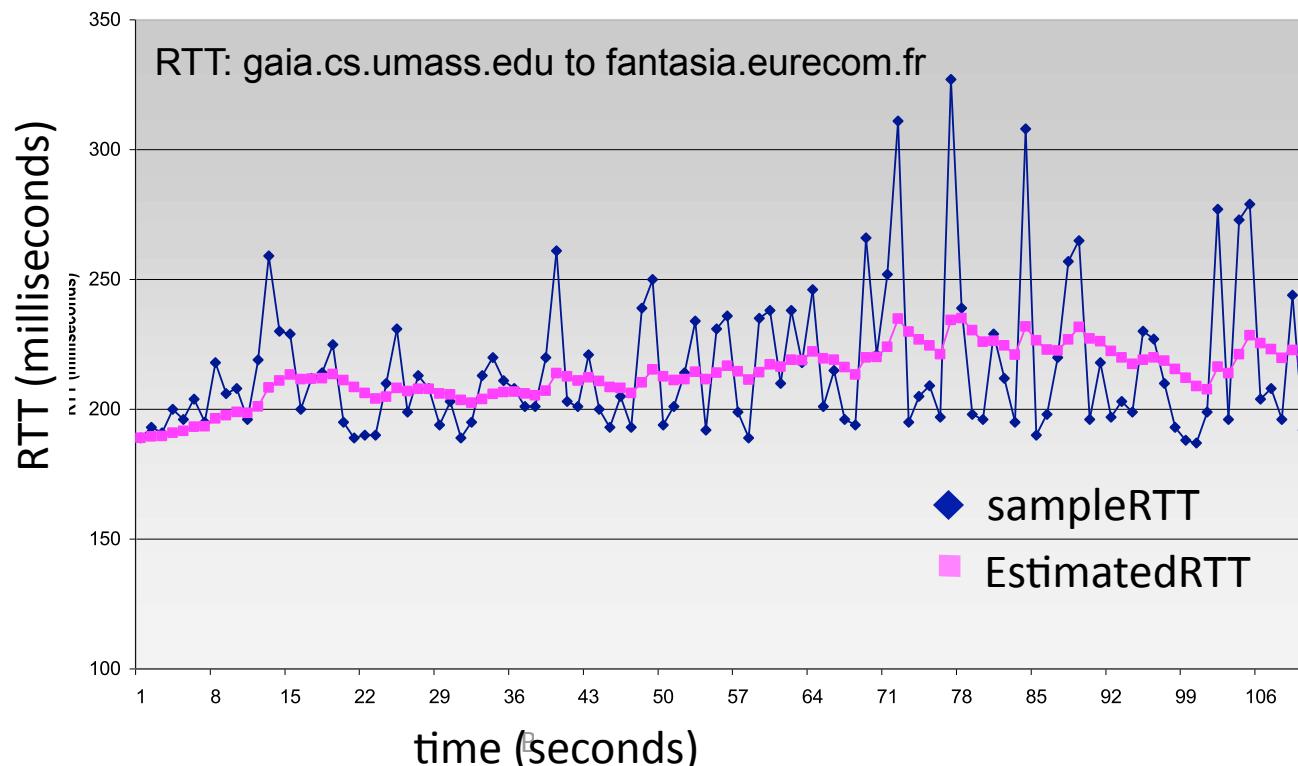
- **SampleRTT**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **SampleRTT** will vary, want estimated RTT “smoother”
  - average several *recent* measurements, not just current **SampleRTT**



# TCP round trip time, timeout

$$\text{EstimatedRTT} = (1 - \alpha) * \text{EstimatedRTT} + \alpha * \text{SampleRTT}$$

- ❖ exponential weighted moving average
- ❖ influence of past sample decreases exponentially fast
- ❖ typical value:  $\alpha = 0.125$



# TCP round trip time, timeout

- **timeout interval:** EstimatedRTT plus “safety margin”
  - large variation in EstimatedRTT → larger safety margin
- estimate SampleRTT deviation from EstimatedRTT:
$$\text{DevRTT} = (1-\beta) * \text{DevRTT} + \beta * |\text{SampleRTT} - \text{EstimatedRTT}|$$
(typically,  $\beta = 0.25$ )

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 * \text{DevRTT}$$



estimated RTT

“safety margin”

# TCP reliable data transfer

- TCP creates **reliable data service** on top of IP's unreliable service
  - pipelined segments
  - cumulative acks
  - single retransmission timer
- retransmissions triggered by:
  - timeout events
  - duplicate acks

let's consider simplified TCP sender:

- ignore duplicate acks
- ignore flow control, congestion control



# TCP sender events:

## *Data received from app:*

- create segment with seq number
- seq number is byte-stream number of first data byte in segment
- start timer if not already running
  - think of timer as for oldest unacked segment
  - expiration interval: `TimeOutInterval`

## *timeout:*

- retransmit segment that caused timeout

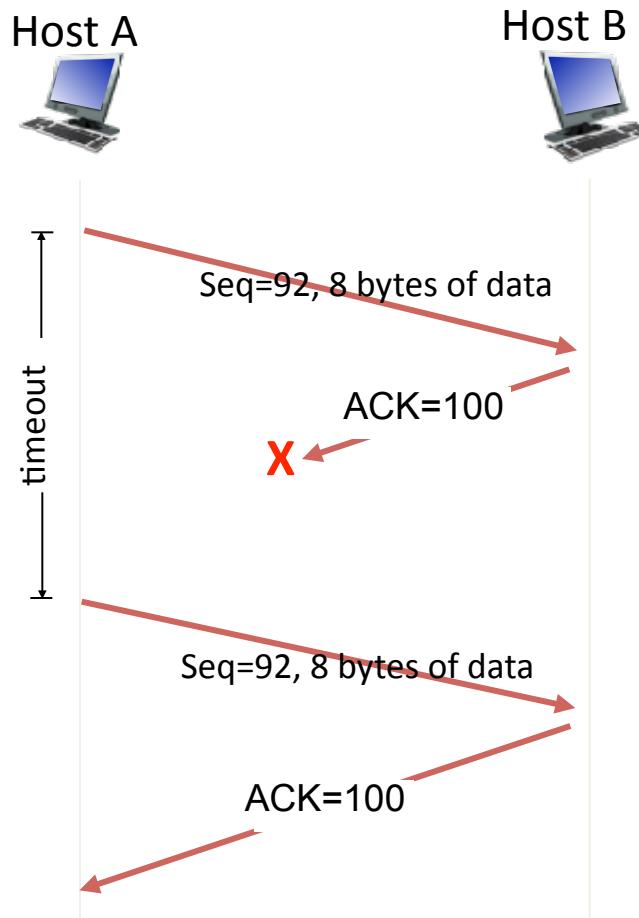
- restart timer

## *ack received:*

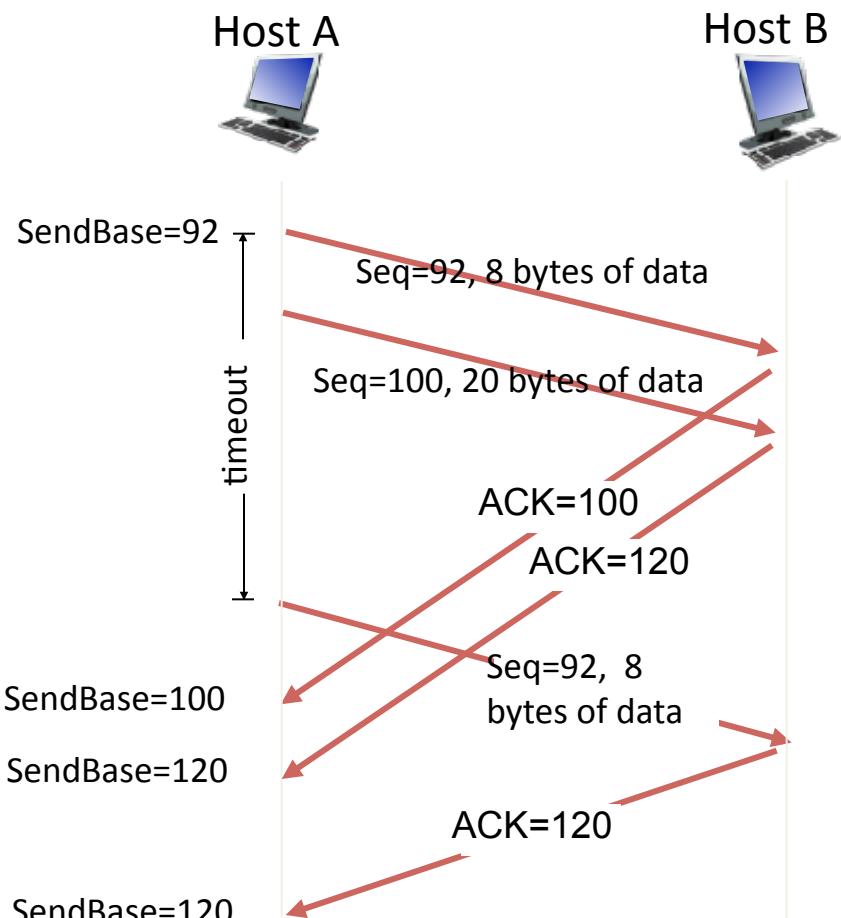
- if ack acknowledges previously unacked segments
  - update what is known to be ACKed
  - start timer if there are still unacked segments



# TCP: retransmission scenarios



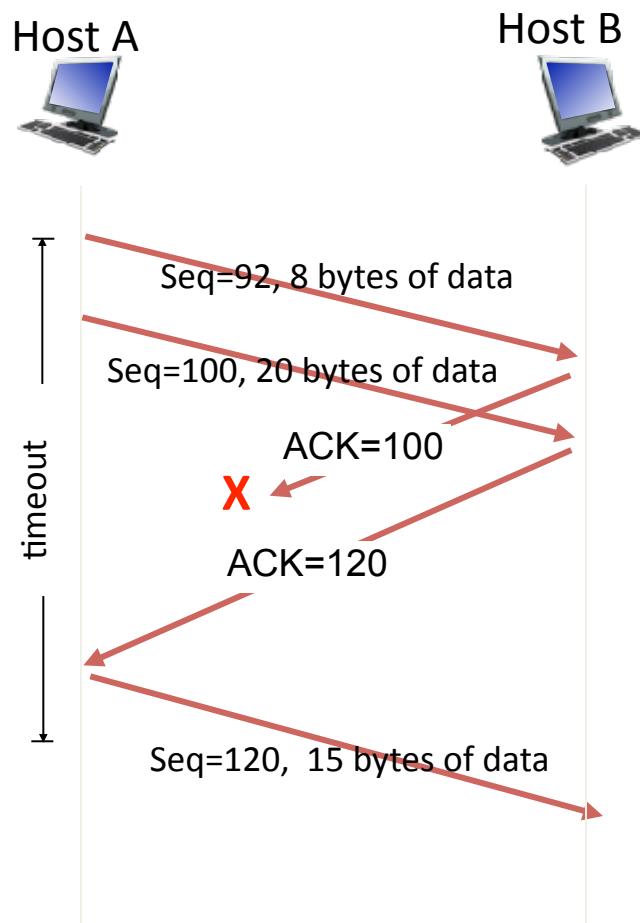
lost ACK scenario



premature timeout



# TCP: retransmission scenarios



cumulative ACK

# TCP ACK generation

<i>event at receiver</i>	<i>TCP receiver action</i>
arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send <i>duplicate ACK</i> , indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap



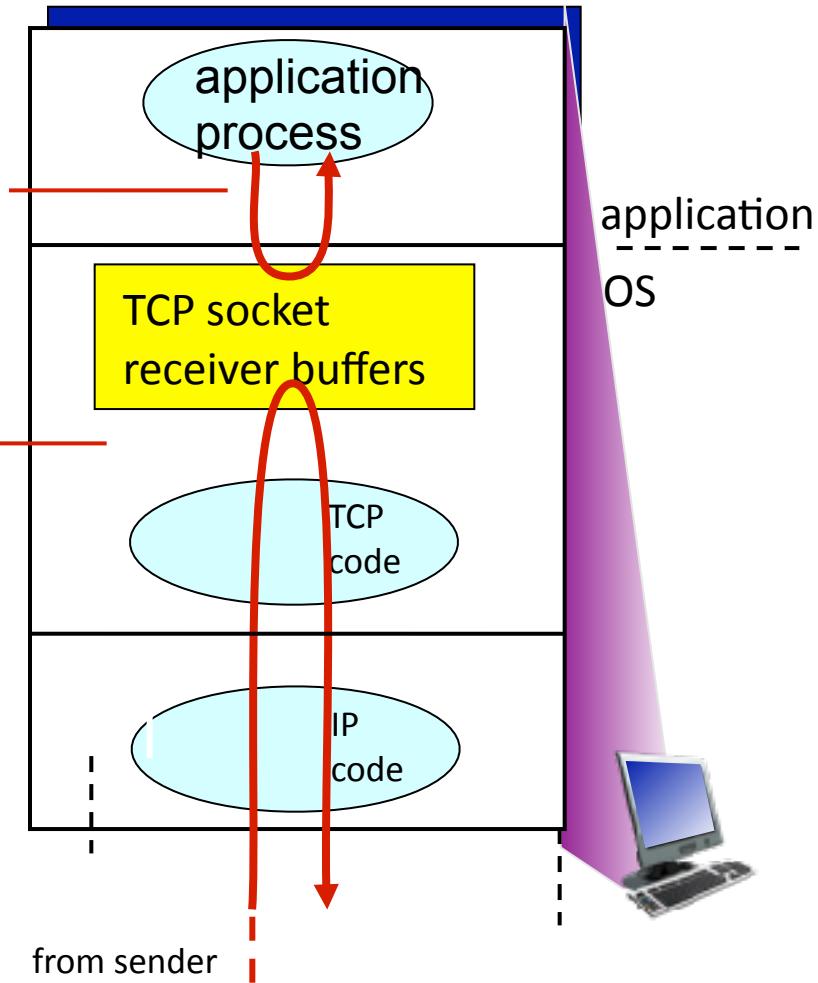
# TCP flow control

## *flow control*

receiver controls sender, so  
sender won't overflow receiver's  
buffer by transmitting too much,  
too fast

application may  
remove data from  
TCP socket buffers ....

... slower than TCP  
receiver is delivering  
(sender is sending)

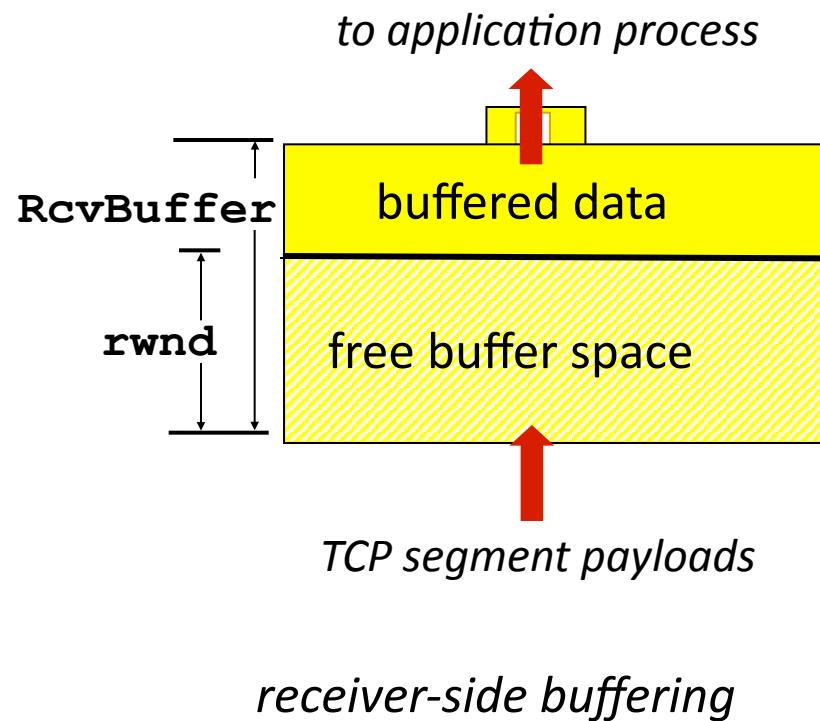


receiver protocol stack



# TCP flow control

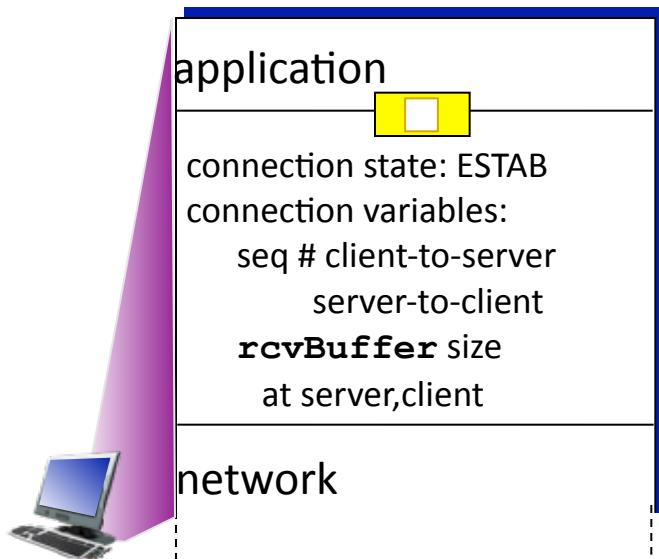
- receiver “advertises” free buffer space by including **rwnd** value in TCP header of receiver-to-sender segments
  - **RcvBuffer** size set via socket options (typical default is 128k bytes)
  - many operating systems autoadjust **RcvBuffer**
- sender limits amount of unacked (“in-flight”) data to receiver’s **rwnd** value
- guarantees receive buffer will not overflow



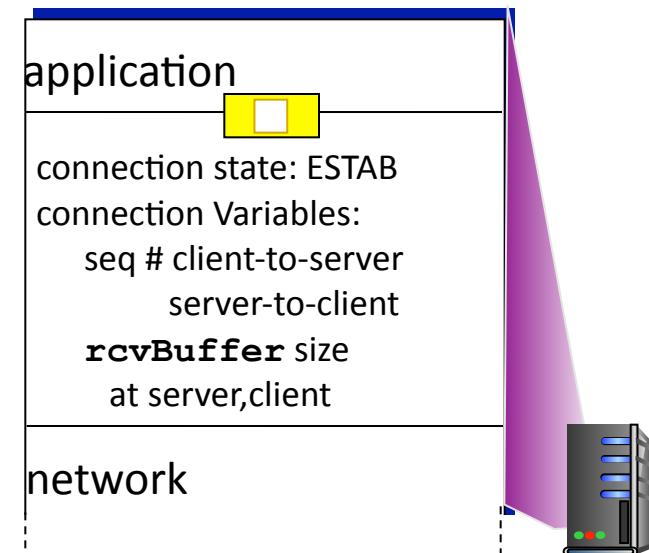
# Connection Management

before exchanging data, sender/receiver “handshake”:

- agree to establish connection (each knowing the other willing to establish connection)
- agree on connection parameters

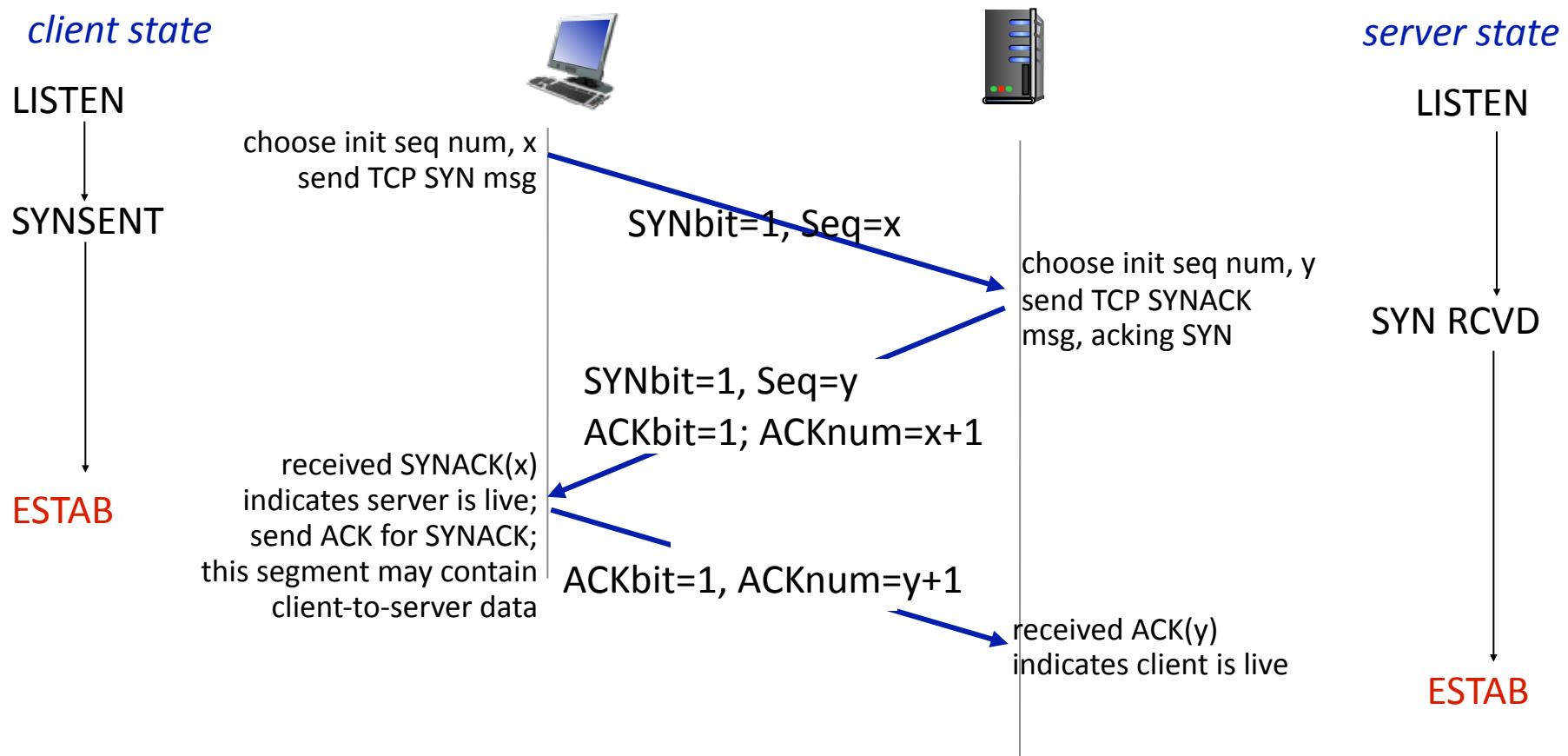


```
Socket clientSocket =  
    newSocket("hostname", "port  
    number");
```



```
Socket connectionSocket =  
    welcomeSocket.accept();
```

# TCP 3-way handshake



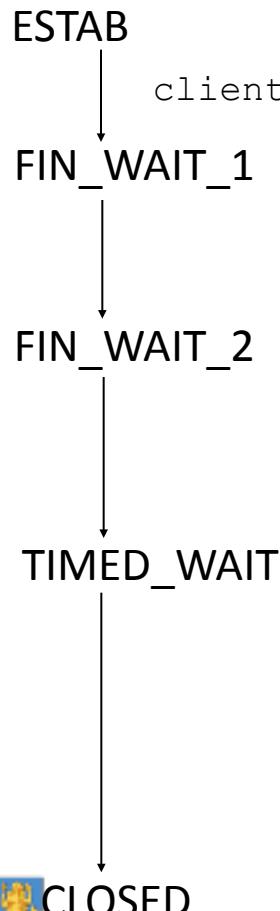
# TCP: closing a connection

- client, server each close their side of connection
  - send TCP segment with FIN bit = 1
- respond to received FIN with ACK
  - on receiving FIN,ACK can be combined with own FIN
- simultaneous FIN exchanges can be handled

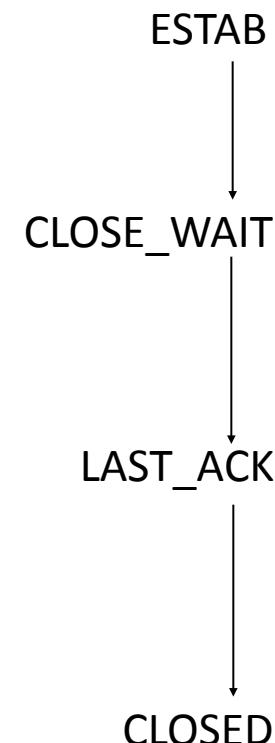


# TCP: closing a connection

*client state*



*server state*



can no longer  
send but can  
receive data

wait for server  
close

timed wait  
for  $2 * \text{max}$   
segment lifetime

FINbit=1, seq=x

ACKbit=1; ACKnum=x+1

FINbit=1, seq=y

ACKbit=1; ACKnum=y+1

can still  
send data

can no longer  
send data

# Summary

- Creation
  - SYN/SYN-ACK/ACK
  - Both sides now have a sequence number.
- Shut down:
  - FIN/ACK/FIN/ACK
  - Might be condensed to FIN/FIN-ACK/ACK if acks delayed and/or fast machines
- Bulk transfer
  - Receiver advertises receive window (rwnd)
  - Sender transmits until rwnd bytes unack'd
  - Receiver acks contiguous segments, plus duplicate ack for out-of-order segments
  - Sender resends all unack'd data on timeout, or receipt of duplicate ack



# More TCP Issues

- Options
  - SACK enables selective acknowledgements
  - Window Scaling deals with  $\text{bandwidth} \times \text{delay} > 64\text{K}$
  - Timestamping for PAWS (protection against wrapped sequence numbers) and RTTM (round trip time measurement)
- Algorithms
  - Slow Start to fill the pipeline smoothly
  - Fast Retransmission of dropped packets
  - Rapid Recovery to avoid Slow Start when it is not necessary.



# Selective Acknowledgement

- Instead of sending a cumulative ACK, instead send a set of “upto...starting from...” pairs.
- Allows the acknowledgement to indicate that parts of the sequence space have been received.
- Other algorithms (especially fast retransmission) have made it less useful, but it is widely supported anyway.



# Window Scaling

- TCP “Receive Window” is advertised in every packet, and indicates how much data the receiver is willing to accept right now.
- It is a 16 bit quantity, so limited to 64k.
- At 100MBps (GigE), this means any network with  $\text{RTT} > 655 \mu\text{s}$ , which is <100km point to point, cannot fill the pipeline (ie, will have to stop and wait for ack)
- 40GigE: 2.5km
- In reality, RTT includes time taken to generate an ack, so limit is far lower.



# Window Scaling

- Solution is to negotiate a scaling factor for the receive window
- Increases effective window size to  $2^{30}$  (shift window left by up to 14 bits)
- Allows bandwidth\*delay up to 1GB.
- Hence GigE up to 10s RTT, or 40GigE up to 250ms RTT.



# PAWS

- If you have a receive window that is approaching 1GB then with small amounts of reordering and delay, you can have duplicate sequence numbers on packets which are 4GB apart in the sequence space.
- “Time”stamp in header is monotonically increasing, every 1ms or less.
- This allows you to distinguish between “new” and “old” packets with same sequence no.
- 32 bit counter running at 1000 Hz takes 50 days to wrap around!



# RTTM

- Measuring RTT can be difficult in the face of retransmissions
- Instead place timestamp in header of data segments (also used for PAWS) and reflect that timestamp back in an ACK.
- Subtract the ACK timestamp from current value of counter to get RTT. Simples!



# Slow Start

- In many environments, the local network is faster than the wide area network.
- So a large advertised receive window (possibly scaled) will result in the sender potentially sending large amounts of data to the local router, which then has to clock it out at a slower speed.
- Potential packet loss (limited router memory), does not make effective use of the whole path.



# Slow Start

- Once connection is established, do not initially send complete receive window.
- Instead, only send one packet, and increase the number of packets sent by one for every acknowledgement received, until the window is full (or double the number of packets, or some other algorithm).
- Conceptually, this spreads out the packets in the network, and reduces congestion at intervening routers.



# Rapid Recovery / Fast Retransmit

- Various algorithms are implemented in various versions of various operating systems, mostly with names like “Tahoe” and “Reno”.
- Uses receipt of duplicate acks to infer that subsequent packets have been lost, and retransmits them right now rather than waiting for timeout.
- Avoids going into slow start, so that window is kept full.



# Silly Window Syndrome

- Window is zero (receiver's buffers are full)
- Receiving application is reading one byte at a time
- Receive window is now 1 byte, so ACK sent opening window by 1
- Sender sends one more byte
- We are now using two complete TCP frames every time one byte is read



# Two solutions

- Receiver: do not advertise small increases in window, but only advertise when either we can accept one more segment, or half our buffer.
- Sender (“Nagle’s algorithm”): do not send small packets when there is outstanding unack’d data (more complex in reality: RFC896).



# Concrete TCP

- I find it easier to understand things from real examples.
- I hope this helps.



# Simple Daemon

```
my $listen = new IO::Socket::INET (LocalPort => $port,
    Proto => "tcp", Listen => 5,
    ReuseAddr => 1) or die "cannot listen $!";
warn "listening on $port";

while (my $s = $listen->accept ()) {
    my $pid = fork ();
    die "fork $" unless defined ($pid);
    if ($pid == 0) {
        warn "connection accepted";
        $listen->close ();
        sendfiles ($s, @ARGV);
        exit (0);
    } else {
        $s->close ();
    }
}
```



# sendfiles

This number matters later

```
sub sendfiles {  
    my $s = shift;  
    while (my $f = shift) {  
        my $fh = new IO::File "<$f" or next;  
        my $buffer;  
        while (my $bytes = $fh->sysread ($buffer, 8192)) {  
            $s->syswrite ($buffer, $bytes) or die "cannot write $!";  
        }  
    }  
}
```



# Connection Refused

```
igb@pi-two:~$ telnet pi-one 5758
Trying 2001:8b0:129f:a90f:ba27:ebff:fe00:efe7...
Trying 10.92.213.231...
telnet: Unable to connect to remote host: Connection refused
igb@pi-two:~$
```



# Connection Rejected

00:00:00.000000 IP6 2001:8b0:129f:a90f:ba27:ebff:fec9:acae.41529 > 2001:8b0:129f:a90f:ba27:ebff:fe00:efe7.5758: Flags [S], seq **1698998887**, win 28000, options [mss **1400**, sack0K, TS val 19928636 ecr 0,nop,wscale 6], length 0

00:00:00.000358 IP6  
2001:8b0:129f:a90f:ba27:ebff:fe00:efe7.5758 >  
2001:8b0:129f:a90f:ba27:ebff:fec9:acae.41529: Flags [R.], seq 0, ack **1698998888**, win 0, length 0

00:00:00.002250 IP 10.92.213.238.44781 > 10.92.213.231.5758:  
Flags [S], seq **1260504038**, win 29200, options [mss **1460**, sack0K, TS val 19928636 ecr 0,nop,wscale 6], length 0

00:00:00.002565 IP 10.92.213.231.5758 > 10.92.213.238.44781:  
Flags [R.], seq 0, ack **1260504039**, win 0, length 0



# Connection Immediately Closed

```
igb@pi-two:~$ telnet pi-one 5757
Trying 2001:8b0:129f:a90f:ba27:ebff:fe00:efe7...
Trying 10.92.213.231...
Connected to pi-one.
Escape character is '^]'.
Connection closed by foreign host.
igb@pi-two:~$
```



# Immediate Close

```
00:00:00.002258 IP 10.92.213.238.49560 > 10.92.213.231.5757: Flags [S],  
seq 2044006020, win 29200, options [mss 1460,sackOK,TS val 19986406 ecr  
0,nop,wscale 6], length 0  
00:00:00.002595 IP 10.92.213.231.5757 > 10.92.213.238.49560: Flags [S.],  
seq 894665383, ack 2044006021, win 28960, options [mss 1460,sackOK,TS val  
19986382 ecr 19986406,nop,wscale 6], length 0  
00:00:00.003596 IP 10.92.213.238.49560 > 10.92.213.231.5757: Flags [.],  
ack 1, win 457, options [nop,nop,TS val 19986406 ecr 19986382], length 0  
00:00:00.018316 IP 10.92.213.231.5757 > 10.92.213.238.49560: Flags [F.],  
seq 1, ack 1, win 453, options [nop,nop,TS val 19986384 ecr 19986406],  
length 0  
00:00:00.019698 IP 10.92.213.238.49560 > 10.92.213.231.5757: Flags [F.],  
seq 1, ack 2, win 457, options [nop,nop,TS val 19986408 ecr 19986384],  
length 0  
00:00:00.021259 IP 10.92.213.231.5757 > 10.92.213.238.49560: Flags [.],  
ack 2, win 453, options [nop,nop,TS val 19986384 ecr 19986408], length 0
```



# Question

- I said in previous lecture minimum TCP session **SEVEN** packets (SYN, SYN/ACK, ACK, FIN, ACK, FIN, ACK).
- Here there are only **SIX**
- Why?



# Delayed Acks at Work

- After receipt of the first FIN, the caller has nothing to say, and no outstanding un-ack'd data
- So ACK is sent only when there is traffic to pass, in this case the answering FIN.
- Hence the close is FIN, FIN/ACK,ACK.
- Requires client to call close() within 500ms of receipt of FIN, with no other traffic outstanding.



# Slightly Delayed Close

```
#!/usr/bin/perl -w

use warnings;
use strict;
use IO::Socket::INET;

my $s = new IO::Socket::INET (PeerAddr => "10.92.213.231:5757", Proto => "tcp") or die "$!";
sleep (2);
exit (0);
```

# Typical Seven Packets

```
00:00:00.000000 IP 10.92.213.238.49583 > 10.92.213.231.5757: Flags [S], seq  
1156157459, win 29200, options [mss 1460,sackOK,TS val 20078528 ecr 0,nop,wscale 6], length 0  
00:00:00.000413 IP 10.92.213.231.5757 > 10.92.213.238.49583: Flags [S.], seq  
1726094991, ack 1156157460, win 28960, options [mss 1460,sackOK,TS val 20078505 ecr  
20078528,nop,wscale 6], length 0  
00:00:00.001393 IP 10.92.213.238.49583 > 10.92.213.231.5757: Flags [.], ack 1, win  
451, options [nop,nop,TS val 20078528 ecr 20078505], length 0  
// 457*2^6 = 29248  
00:00:00.014287 IP 10.92.213.231.5757 > 10.92.213.238.49583: Flags [F.], seq 1, ack  
1, win 453, options [nop,nop,TS val 20078506 ecr 20078528], length 0  
00:00:00.023675 IP 10.92.213.238.49583 > 10.92.213.231.5757: Flags [.], ack 2, win  
457, options [nop,nop,TS val 20078531 ecr 20078506], length 0  
// TWO SECOND DELAY  
00:00:02.002245 IP 10.92.213.238.49583 > 10.92.213.231.5757: Flags [F.], seq 1, ack  
2, win 457, options [nop,nop,TS val 20078728 ecr 20078506], length 0  
00:00:02.002556 IP 10.92.213.231.5757 > 10.92.213.238.49583: Flags [.], ack 2, win  
453, options [nop,nop,TS val 20078705 ecr 20078728], length 0
```



# Window Scaling

- Note Window Scale has been set to 6  
options [mss 1460,sackOK,TS val 19986406 ecr 0,nop,wscale 6]
- Both implementations are the same (Linux, pretty much this week's kernel)
- Win is being sent as  $\sim 500, 500 * 2^6 = \sim 32000$ , which is a typical value for a Receive Window



# Other Systems Are Different

- Solaris requires more provocation to select Window Scaling, especially on local area network
- Can be Asymmetric

```
00:07:51.826744 IP 10.92.213.241.44205 > 10.92.213.231.5757: Flags [S], seq 87151905, win 64240,  
options [mss 1460,sackOK,TS val 19039311 ecr 0,nop,wscale 1], length 0  
00:07:51.827087 IP 10.92.213.231.5757 > 10.92.213.241.44205: Flags [S.], seq 3263477770, ack  
87151906, win 28960, options [mss 1460,sackOK,TS val 20196188 ecr 19039311,nop,wscale 6],  
length 0
```



# Bulk Data

```
igb@pi-two:~$ telnet pi-one 5757
Trying 2001:8b0:129f:a90f:ba27:ebff:fe00:efe7...
Trying 10.92.213.231...
Connected to pi-one.
Escape character is '^]'.
pi-one.home.batten.eu.org, DNS, DHCP, NTP (GPS), heyu, odds and ends.

*** This is a private machine. If you are not personally authorised by
*** igb@batten.eu.org then your access is illegal.

Connection closed by foreign host.
igb@pi-two:~$
```



# Sending small amounts of data

SYN SYN/ACK ACK as usual, options deleted for clarity

Data Packet:

```
00:03:11.671436 IP 10.92.213.231.5757 > 10.92.213.238.49591:  
Flags [P.], seq 1:196, ack 1, win 453, length 195
```

Ack:

```
00:03:11.672488 IP 10.92.213.238.49591 > 10.92.213.231.5757:  
Flags [.], ack 196, win 473, length 0
```

Start of shut down:

```
00:03:11.675722 IP 10.92.213.231.5757 > 10.92.213.238.49591:  
Flags [F.], seq 196, ack 1, win 453, length 0
```



# Slightly more data

- Notice mss is 1460, so anything more than 1400 bytes will require more than one packet.
- Let's try 2KB

```
igb@pi-one:~$ i=1;while [[ $i -lt 2048 ]]; do echo -n 0; let i=i+1; done > 2kfile  
igb@pi-one:~$
```

```
igb@pi-two:~$ telnet pi-one 5757  
Trying 2001:8b0:129f:a90f:ba27:ebff:fe00:efe7...  
Trying 10.92.213.231...  
Connected to pi-one.  
Escape character is '^]'.  
00000000...0Connection closed by foreign host.  
igb@pi-two:~$
```



```
// Note caller is now Solaris, wscale 1, options deleted
00:00:00.002792 IP 10.92.213.241.63887 > 10.92.213.231.5757:
Flags [.], ack 1, win 64436, length 0
00:00:00.021531 IP 10.92.213.231.5757 > 10.92.213.241.63887:
Flags [.], seq 1:1449, ack 1, win 453, length 1448
00:00:00.022776 IP 10.92.213.241.63887 > 10.92.213.231.5757:
Flags [.], ack 1449, win 64436, length 0
00:00:00.024380 IP 10.92.213.231.5757 > 10.92.213.241.63887:
Flags [P.], seq 1449:2048, ack 1, win 453, length 599
00:00:00.025245 IP 10.92.213.241.63887 > 10.92.213.231.5757:
Flags [.], ack 2048, win 64436, length 0
00:00:00.027568 IP 10.92.213.231.5757 > 10.92.213.241.63887:
Flags [F.], seq 2048, ack 1, win 453, length 0
00:00:00.028577 IP 10.92.213.241.63887 > 10.92.213.231.5757:
Flags [.], ack 2049, win 64436, length 0
00:00:00.028884 IP 10.92.213.241.63887 > 10.92.213.231.5757:
Flags [F.], seq 1, ack 2049, win 64436, length 0
```



# Slightly more data

```
igb@pi-one:~$ i=0;while [[ $i -lt 2048 ]]; do echo 0123456789ABCDE; let i=i+1; done > 32kfile
igb@pi-one:~$ ls -l 32kfile
-rw-r--r-- 1 igb igb 32768 Feb 12 06:54 32kfile
igb@pi-one:~$  
  
igb@zone-host:~$ more /tmp/qqq
Trying 2001:8b0:129f:a90f:ba27:ebff:fe00:efe7...
telnet: connect to address 2001:8b0:129f:a90f:ba27:ebff:fe00:efe7: Connection refused
Trying 10.92.213.231...
Connected to pi-one.home.batten.eu.org.
Escape character is '^]'.
0123456789ABCDE
0123456789ABCDE
0123456789ABCDE
0123456789ABCDE
0123456789ABCDE  
  
...
0123456789ABCDE
0123456789ABCDE
Connection to pi-one.home.batten.eu.org closed by foreign host.
igb@zone-host:~$
```



00:00:00.001631 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.001990 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 0  
00:00:00.002933 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.020559 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.021842 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.023312 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.024515 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.026147 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.027936 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.029063 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.030215 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.030595 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 952  
00:00:00.031462 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.032635 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.034243 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.036050 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.036384 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.036678 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.037042 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 952  
00:00:00.037689 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.038255 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.038591 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.038871 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.039169 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.039455 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.039982 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.040343 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.040604 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.040922 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.045585 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.046607 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.047684 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0  
00:00:00.049154 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 1448  
00:00:00.050696 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 456  
00:00:00.054361 IP 10.92.213.231.5757 > 10.92.213.241.54141: tcp 0  
00:00:00.055249 IP 10.92.213.241.54141 > 10.92.213.231.5757: tcp 0

# Slow Start

- Note behaviour of slow start
- Sender initially only has small amounts of unack'd data, but progressively speeds up by sending more and more packets “back to back”



# What's going on?

- What are those 952 byte packets? 952?  
Where does that come from?
- $1448 * 5 + 952 = 8192$ , the blocks I'm calling the API with.
- When in slow start, there's enough delay on proceedings that the blockings matter.
- Once the transfer really gets going, they're aggregated.
- $1448 * 11 + 456 = 8192 * 2$ : two blocks



# Edit daemon...

- Make the block size I'm sending from userland be 64k instead (ie, all of 32k file sent in one `write()`)



00:00:00.021483 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.022790 IP 10.92.213.241.57935 > 10.92.213.231.5757: tcp 0  
00:00:00.024287 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.025551 IP 10.92.213.241.57935 > 10.92.213.231.5757: tcp 0  
00:00:00.027213 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.028780 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.030097 IP 10.92.213.241.57935 > 10.92.213.231.5757: tcp 0  
00:00:00.031315 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.031664 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.031967 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.032257 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.032536 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.032837 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.033189 IP 10.92.213.241.57935 > 10.92.213.231.5757: tcp 0  
00:00:00.033575 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.033799 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.034122 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.034340 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.034686 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.034897 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.035255 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.035473 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.035805 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.036099 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.036569 IP 10.92.213.241.57935 > 10.92.213.231.5757: tcp 0  
00:00:00.036579 IP 10.92.213.241.57935 > 10.92.213.231.5757: tcp 0  
00:00:00.041608 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.042676 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 1448  
00:00:00.043686 IP 10.92.213.241.57935 > 10.92.213.231.5757: tcp 0  
00:00:00.044757 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 912  
00:00:00.047246 IP 10.92.213.231.5757 > 10.92.213.241.57935: tcp 0  
00:00:00.048122 IP 10.92.213.241.57935 > 10.92.213.231.5757: tcp 0



# Programming Tip

- For maximum performance, send the largest possible blocks into the kernel
- With 8192 byte writes, 30ms to send 32768 bytes
- With 65536 byte writes, ie “file at a file”, 23ms
- 23% performance improvement by changing one constant!
- If available, use “sendfile()” system call



# IM file, steady state

```
00:00:00.299366 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.299706 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.299997 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.300314 IP 10.92.213.241.53681 > 10.92.213.231.5757: tcp 0
00:00:00.300575 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.300854 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.301345 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.301584 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.301843 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.302088 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.302330 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.302560 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.302797 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.303115 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.303477 IP 10.92.213.241.53681 > 10.92.213.231.5757: tcp 0
00:00:00.303758 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.304058 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.304314 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.304555 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.305517 IP 10.92.213.241.53681 > 10.92.213.231.5757: tcp 0
00:00:00.306582 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.307198 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
00:00:00.307785 IP 10.92.213.231.5757 > 10.92.213.241.53681: tcp 1448
```



# Acks are trailing data

```
00:00:00.624882 IP 10.92.213.231.5757 > 10.92.213.241.34058: Flags [.], seq 988985:990433, ack 1,  
win 453, length 1448  
00:00:00.625125 IP 10.92.213.231.5757 > 10.92.213.241.34058: Flags [P.], seq 990433:991881, ack  
1, win 453, length 1448  
00:00:00.625469 IP 10.92.213.231.5757 > 10.92.213.241.34058: Flags [.], seq 991881:993329, ack 1,  
win 453, length 1448  
00:00:00.625747 IP 10.92.213.231.5757 > 10.92.213.241.34058: Flags [.], seq 993329:994777, ack 1,  
win 453, length 1448  
00:00:00.626078 IP 10.92.213.241.34058 > 10.92.213.231.5757: Flags [.], ack 991881, win 64436,  
length 0  
00:00:00.626310 IP 10.92.213.231.5757 > 10.92.213.241.34058: Flags [.], seq 994777:996225, ack 1,  
win 453, length 1448  
00:00:00.626575 IP 10.92.213.231.5757 > 10.92.213.241.34058: Flags [.], seq 996225:997673, ack 1,  
win 453, length 1448  
00:00:00.627065 IP 10.92.213.231.5757 > 10.92.213.241.34058: Flags [.], seq 997673:999121, ack 1,  
win 453, length 1448
```



# What about broken networks?

```
# use a very, very rubbish transmission protocol

sub sendfiles {
    my $s = shift;
    while (my $f = shift) {
        my $fh = new IO::File "<$f" or next;
        my $buffer;
        while (my $bytes = $fh->sysread ($buffer, 8)) {
            $s->syswrite ($buffer, $bytes) or die "cannot write $!";
            sleep (1);
        }
    }
}
```



00:00:00.020080 IP 10.92.213.231.5757 > 10.92.213.238.49778: Flags [P.], seq 1:9, ack 1, win 453, options [nop,nop,TS val 21020214 ecr 21020236], length 8  
00:00:00.021062 IP 10.92.213.238.49778 > 10.92.213.231.5757: Flags [.], ack 9, win 457, options [nop,nop,TS val 21020238 ecr 21020214], length 0  
00:00:01.023658 IP 10.92.213.231.5757 > 10.92.213.238.49778: Flags [P.], seq 9:17, ack 1, win 453, options [nop,nop,TS val 21020314 ecr 21020238], length 8  
00:00:01.024786 IP 10.92.213.238.49778 > 10.92.213.231.5757: Flags [.], ack 17, win 457, options [nop,nop,TS val 21020338 ecr 21020314], length 0  
00:00:02.027197 IP 10.92.213.231.5757 > 10.92.213.238.49778: Flags [P.], seq 17:25, ack 1, win 453, options [nop,nop,TS val 21020415 ecr 21020338], length 8  
00:00:02.028213 IP 10.92.213.238.49778 > 10.92.213.231.5757: Flags [.], ack 25, win 457, options [nop,nop,TS val 21020439 ecr 21020415], length 0  
00:00:03.030555 IP 10.92.213.231.5757 > 10.92.213.238.49778: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21020515 ecr 21020439], length 8  
00:00:03.031564 IP 10.92.213.238.49778 > 10.92.213.231.5757: Flags [.], ack 33, win 457, options [nop,nop,TS val 21020539 ecr 21020515], length 0



# We can cause Linux to lose packets

```
igb@pi-one:~$ sudo iptables -I OUTPUT -p tcp -m tcp --sport 5757 -j DROP
```

```
igb@pi-one:~$ sudo iptables --delete OUTPUT 1
```

Note that the dropped packets don't appear in tcpdump output when tcpdump is run on the local machine; if you need that, you need to monitor at the other end.

# What happened?

```
00:00:02.025408 IP 10.92.213.231.5757 > 10.92.213.238.49784: Flags [P.], seq 17:25, ack 1, win 453, options [nop,nop,TS val 21041889 ecr 21041813], length 8
00:00:02.026411 IP 10.92.213.238.49784 > 10.92.213.231.5757: Flags [.], ack 25, win 457, options [nop,nop,TS val 21041913 ecr 21041889], length 0
00:00:03.028760 IP 10.92.213.231.5757 > 10.92.213.238.49784: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21041989 ecr 21041913], length 8
00:00:03.029824 IP 10.92.213.238.49784 > 10.92.213.231.5757: Flags [.], ack 33, win 457, options [nop,nop,TS val 21042013 ecr 21041989], length 0
00:00:04.032088 IP 10.92.213.231.5757 > 10.92.213.238.49784: Flags [P.], seq 33:41, ack 1, win 453, options [nop,nop,TS val 21042090 ecr 21042013], length 8
00:00:04.033088 IP 10.92.213.238.49784 > 10.92.213.231.5757: Flags [.], ack 41, win 457, options [nop,nop,TS val 21042114 ecr 21042090], length 0
00:00:22.039894 IP 10.92.213.231.5757 > 10.92.213.238.49784: Flags [P.], seq 41:185, ack 1, win 453, options [nop,nop,TS val 21043891 ecr 21042114], length 144
00:00:22.040918 IP 10.92.213.238.49784 > 10.92.213.231.5757: Flags [.], ack 185, win 473, options [nop,nop,TS val 21043914 ecr 21043891], length 0
00:00:23.040658 IP 10.92.213.231.5757 > 10.92.213.238.49784: Flags [P.], seq 185:193, ack 1, win 453, options [nop,nop,TS val 21043991 ecr 21043914], length 8
```



# Note

- Packets aren't retransmitted, data is retransmitted
- So the dropped packets' data is all re-sent in the minimum number of packets it will fit in



# Packets Dropped

```
igb@pi-one:~$ sudo iptables -I OUTPUT -p tcp -m tcp --sport 5757 -j DROP; sleep 30; sudo iptables
--list -v; sudo iptables --delete OUTPUT 1
Chain INPUT (policy ACCEPT 53 packets, 3108 bytes)
pkts bytes target     prot opt in     out     source          destination
Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source          destination
Chain OUTPUT (policy ACCEPT 54 packets, 10767 bytes)
pkts bytes target     prot opt in     out     source          destination
  30   4153 DROP       tcp   --  any    any    anywhere      anywhere          tcp spt:
5757
igb@pi-one:~$
```



# Drop Packets on Other Side

- If we drop packets at the INPUT of the receiver, we can see the retransmissions

```
sudo iptables -I INPUT -p tcp -m tcp --sport 5757 -j DROP; \
sleep 10; sudo iptables -D INPUT 1
```



00:02:55.213001 IP 10.92.213.238.49817 > 10.92.213.231.5757: Flags [.], ack 17, win 457, options [nop,nop,TS val 21158147 ecr 21158123], length 0  
00:02:56.215151 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 17:25, ack 1, win 453, options [nop,nop,TS val 21158223 ecr 21158147], length 8  
00:02:56.216163 IP 10.92.213.238.49817 > 10.92.213.231.5757: Flags [.], ack 25, win 457, options [nop,nop,TS val 21158247 ecr 21158223], length 0  
00:02:57.215914 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21158323 ecr 21158247], length 8  
00:02:57.418894 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21158344 ecr 21158247], length 8  
00:02:57.628899 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21158365 ecr 21158247], length 8  
00:02:58.048900 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21158407 ecr 21158247], length 8  
00:02:58.888921 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21158491 ecr 21158247], length 8  
00:03:00.568905 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21158659 ecr 21158247], length 8  
00:03:03.938934 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21158996 ecr 21158247], length 8  
00:03:10.678934 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 25:33, ack 1, win 453, options [nop,nop,TS val 21159670 ecr 21158247], length 8  
00:03:10.679897 IP 10.92.213.238.49817 > 10.92.213.231.5757: Flags [.], ack 33, win 457, options [nop,nop,TS val 21159693 ecr 21159670], length 0  
00:03:10.680249 IP 10.92.213.231.5757 > 10.92.213.238.49817: Flags [P.], seq 33:137, ack 1, win 453, options [nop,nop,TS val 21159670 ecr 21159693], length 104  
00:03:10.681190 IP 10.92.213.238.49817 > 10.92.213.231.5757: Flags [.], ack 137, win 457, options [nop,nop,TS val 21159693 ecr 21159670], length 0



# Why don't the packets get bigger?

- Implementation decision: could be different on other operating systems (although appears same on Solaris, as it happens)
- Makes sense on slow, lossy network: once you suspect packets are going missing, more sensible to send small packets to find out if it is working than large ones.



# Networking API

I.G.Batten@bham.ac.uk

# Socket History

- Originally there were a variety of ad-hoc mechanisms which provided access directly to TCP (or whatever) from the OS trap/system-call layer
- Subsequently there have been better (for some value of “better”) interfaces, notably in Plan 9 and similar
  - `open (“/net/tcp/192.168.1.1/10”)` to get a stream
- But Sam Leffler’s “socket” interface from 4.1c bsd, finalised in roughly its current form in 4.2bsd, is almost universal
  - Alternatives are usually different names and calling conventions for same concepts

# Buffered I/O

- Userspace programs abstract I/O behind language-specific, hopefully portable, interfaces
  - Buffered Streams of various types in Java
  - FILE \* (“stdio”, “standard I/O”) in C
  - IO::Stream in Perl
  - Similar facilities in C++, C#, Python, etc

# Abstract I/O

- These facilities try to prevent kernel (or equivalent) routines being called every time you read or write a single character, handling buffering for you
  - stdio block buffers disk, line buffers stdout, character buffers stderr, plus “fflush”
- They also abstract things like permissions
- Common Lisp abstracts pathnames, but that facility is less common (OSX’s case insensitivity and handling of colons can be fun for Unix programs, ditto “what happens if I put a slash in a filename” on \*nix)

# Underlying Routines

- `fd = open(path, mode);` // return an int (**an fd, or file descriptor**) that indexes into a table of open files
- `bytes = read(fd, buffer, size);`
- `bytes = write(fd, buffer, length);`
- `err = close(fd);` // don't cast this to void, because NFS
- `err = ioctl(fd, operation, argptr);` // do stuff to underlying physical device
- `err = fcntl(fd, operation, argptr);` // do stuff to file descriptor generically

# Works for devices

- `ioctl()` is used to do things like setting whether `read()` returns on `\r` or on each character as it is typed
- `fcntl()` is used to do things like set non-blocking I/O
  - Distinction less clear than it was, because hardware multiplexors for RS232 aren't a thing any more

# “Everything is a file”

- New thing in Unix, compared to influences like Multics and TOPS-20, was that there was a file, with a name, you could `open()` for everything (“`/dev/console`”, “`/dev/kmem`”, “`/dev/drum`”, “`/dev/rsd3c`”)
- Files just a stream of bytes: anything else is up to user-space (records, for example). Multics did this for files, but not for devices
  - Not true for “block devices”, but not relevant here
  - `/dev` contains “special files” that have a major device number (“what is this?”) and a minor device (“which one of them is this”) which can be `open()`d and then `read()` and `write()` used for I/O, with `ioctl()` doing any required magic

# Problem for /net

- Creating a fully populated /net impossible even for IPv4 ( $2^{32}$  addresses \*  $2^{16}$  ports is 256TB even at one byte per entry, several Petabytes in real filesystems)
- Tricks by which files spring into existence when `open()`d and disappear when `close()`d are possible, but violate established Unix practice (“`ls`” doesn’t work).
  - Plan 9 “from outer space” was explicitly Unix-influenced, but not afraid to break Unix where necessary.
  - Linux /proc offers facility for files you cannot `ls`, but rarely used

# “clone devices”

- There was some precedent with Pseudo-TTYs
- These are the things that are created by ssh to act as a fake teletype (yes!) so that programs that need to mess around with line disciplines, such as emacs and vi, see what looks like a tty rather than a raw network stream
- Large numbers (100s) were needed on time-share machines of the 80s and 90s, so a mechanism arose to allow you to open a generic device and have a specific device created on your behalf (“/dev/ptmx” - look it up on a Solaris machine with “man pts”, if you can find one)
- Quite slow, doesn’t scale to massive numbers (lots of problems on Solaris as Internet grew became a thing) - created new real devices on the fly

# Sockets

- Compromise: we accept that we cannot make connections with `open()`, so don't need an object in the filesystem (or, more strictly, the filesystem namespace) corresponding to a network endpoint
- New system calls that create connections, but in a form that can be used (with care) with `read()` and `write()` (so that most things that work over a tty will work over a network, cf. uucp)
- “**sockets**”
- Sidenote: Sam Leffler went on to be a co-founder of Pixar, showing there is money in OS development

# Socket

- `socket()` system call creates a single end of a network connection
  - Again breaks with Unix tradition, as it creates fds which are not ready for `read()` or `write()`
- `bind()` associates the local end, the socket itself, with some addressing information
- `connect()` actively links a socket to another endpoint
- `accept()` is a passive version of `connect()`, following `listen()`
- `send()`, `sendto()`, `recv()`, `recvfrom()` are `write()` and `read()` analogues, although you can use `read()` and `write()` too if you don't need the extra functionality
- `shutdown()` is a richer version of `close()` (allows half-close)

# socket()

- `int socket(int domain, int type, int protocol);`
  - `domain = { PF_UNIX, PF_INET, PF_INET6, ... }`
    - Plus other stuff as appropriate for X25, ISDN...
  - `type = { SOCK_STREAM, SOCK_DGRAM, SOCK_RAW, ... }`
    - TCP, UDP, ability to make raw packets, other stuff on a per-OS basis
  - `protocol` is usually zero, but allows you to force use of a particular non-standard protocol (PF\_INET + SOCK\_DGRAM otherwise always UDP, etc)

# bind

- `int bind(int s, const struct sockaddr *name, socklen_t namelen);`
  - `struct sockaddr` is a generic for `struct sockaddr_in`, `sockaddr_in6`, etc.
  - You put protocol-specific information in here

```
int
get_listen_socket (const uint16_t port) {

    struct sockaddr_in6 my_address;

    memset (&my_address, '\0', sizeof (my_address));
    my_address.sin6_family = AF_INET6; /* this is an ipv6 address */
    my_address.sin6_addr = in6addr_any; /* bind to all interfaces */
    my_address.sin6_port = htons (port); /* network order */

    int s = socket (PF_INET6, SOCK_STREAM, 0);

    if (s < 0) {
        /* error handling */
    }

    if (bind (s, (struct sockaddr *) &my_address,
              sizeof (my_address)) != 0) {
        /* error handling */
    }
}
```

# Byte Ordering

- IBM heritage, 68k, SPARC: “big endian” - **most significant byte** of a multi-byte quantity has **lowest** address
- Intel heritage, VAX heritage, ARM (usually) heritage: “little endian” - **most significant byte** of a multi-byte quantity has **highest** address
  - Means you can do multibyte addition, including the carry, while moving up the address space
  - “On the VAX bytes are handled backwards from most everyone else in the world. This is not expected to be fixed in the near future.”

# Concretely

Represent the  
number  
123456, 1E240  
hex, in  
memory

Address	Big Endian	Little Endian
1000	0	40
1001	1	E2
1002	E2	1
1003	40	0

# There are others!

- Less commonly encountered, but as well as 1234 (big endian) and 4321 (little endian) there is/was also, for example, 2143 (Pyramid and some pdp11s).
- For extra entertainment, ARM processors and modern SPARCs can operate with either ordering, although individual OSes tend to force it one way or the other
- Vital to use the OS-supplied macros and never try to roll your own

# But on the wire?

- Network ordering is the natural big-endian order, so the most significant bit is transmitted first, the least significant bit is transmitted last
- Great for m68k and SPARC, not so good for everyone else
- Requires care for code-portability (and in any situation where you write ints to disk directly, in passing)
- More generally, of course, serialising more complex data types requires great care

# Use the macros!

- `htonl()`, `ntohs()`, etc
- Convert between **host** order and **network** order, for **shorts** and **longs**
- Anything coming from the network must be processed with `ntohl()` (usually), anything going to the network must be processed with `htonl()`.
- If you get it wrong on x86 your code almost certainly won't work (this is why it is important to test against independent implementations)
- More dangerously, if you get these wrong on SPARC your code will work (all the macros are no-ops), but it will not port to x86. Caused much fun over the years
  - allegedly this is why SPARC and PowerPC are big-endian, as both were m68k replacements and their sponsors were worried about their code-base not being fully compliant
  - See also debate about whether `*((char*)0) == '\0'`

# Voodoo Programming

- Many books, and some of my code, show use of bzero() or memset() to zero the contents of a sockaddr\_in before use
- Shouldn't be necessary and correctly assigning values more portable
  - Are int a = 0; and int a; bzero (&a, sizeof (a)); actually the same? Some standards say no (although 99.999% likely to work)
  - You should explicitly set all the elements you are using, at least, and not rely on things you need to be zero being zero'd by the above.

# Connecting

Create address of other end, and then...

```
if( connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
{
    printf("\n Error : Connect Failed \n");
    return 1;
}
```

If you succeed, the socket is connected

# Where do addresses come from?

- Historically, `gethostbyname()`
- Today all new code should use `getaddrinfo()`
- Consult documentation for the gory details
  - Abstract naming services, so look in a variety of places
  - `/etc/hosts`, DNS, OS-specific directories, AD...

# What about servers?

- Create a socket, as before (bound to INADDR\_ANY or in6addr\_any, note the difference in case)
- Call `listen(s, backlog)`
  - No-one really knows what backlog means, but 5 is the traditional number - it might mean a backlog of 5, it might be non-linear, it might be ignored: only way to know is to read the kernel source
  - Then call `accept()`

```

while (1) {
    struct sockaddr_in6 their_addr;
    socklen_t size = sizeof(their_addr);

    int newsock = accept(sock, (struct sockaddr *)&their_addr, &size);

    if (newsock == -1) {
        perror("accept");
    }
    else {
        printf("Got a connection from %s on port %d\n",
               /* decode sockaddr_in6 here */
               handle(newsock));
    }
}

```

- NB1: you get back a new socket, with the existing socket left to listen for more
- NB2: if you pass in non-zero arguments, the address of the caller is filled in for you
- NB3: note that **&** on the size: you pass in a pointer to how much space there is, so you can be told how big the result is

# Old-School

- `fork()` is a routine which makes a copy of a running process and leaves both of them running
- Only difference is that `fork()` returns a process id into the parent process and 0 into the child process
- Traditional servers fork on incoming connections and have a process per connection

# fork() server

```
pid = fork();
if (pid == 0) {
    /* In child process */
    close(sock); /* child will not accept() again */
    handle(newsock); /* your code goes here */
    return 0;
}
else {
    /* Parent process */
    if (pid == -1) {
        perror("fork");
        return -1;
    }
    else {
        close(newsock); /* this is important */
    }
}
```

# New-School

- Create a new thread on each successful call to `accept()`
- No need to close the `listen()`ing socket, as everything in one process
- *pthreads* makes this portable-ish, although high-load performance is something of a lottery

# Use thread control blocks

```
typedef struct thread_control_block {
    int client;
    struct sockaddr_in6 their_address;
    socklen_t their_address_size;
} thread_control_block_t;

thread_control_block_t *tcb_p = malloc (sizeof (*tcb_p));

if (tcb_p == 0) {
    perror ("malloc");
    exit (1);
}

tcb_p->their_address_size = sizeof (tcb_p->their_address);

/* we call accept as before, except we now put the data into the
   thread control block, rather than onto our own stack,
   because...[1] */
if ((tcb_p->client = accept (s, (struct sockaddr *) &(tcb_p->their_address),
    &(tcb_p->their_address_size))) < 0) {
```

# And create a thread

```
pthread_t thread;

if (pthread_create (&thread, 0, &client_thread,
                    (void *) tcb_p) != 0) {
    /* error handling */
}
```

# Key points with threads

- **Can access:**
  - global variables (might need a mutex)
  - variables on their **own** stack (automatics in the thread start routine and anything it calls)
  - malloc'd space which is created by another thread (again, might need a mutex)
- **Must not access:**
  - variables on other threads' stacks, including the “main” thread.

# Pros and Cons

- `fork()` is expensive (but not as bad as it used to be)
  - **NEVER USE `vfork()` EVEN IF OLD BOOKS TELL YOU TO**
  - 4.2BSD, 1984, last updated for OSX 1993: “Users should not depend on the memory sharing semantics of `vfork` as it will, in that case, be made synonymous to `fork`. ”
  - Linux, 2012: “Some consider the semantics of `vfork()` to be an architectural blemish, and the 4.2BSD man page stated: “This system call will be eliminated when proper system sharing mechanisms are implemented. Users should not depend on the memory sharing semantics of `vfork()` as it will, in that case, be made synonymous to `fork(2)`. ””
- child processes are isolated, which is good for security but bad for co-operation
- child processes can drop privilege (ie, can determine correct user and then become that user irrevocably) while all threads run with same privilege: this is important for security.
- Unlikely you could write a production-quality server for a publicly available service with threads alone

# I/O

- You can then `read()` and `write()` the new socket
  - or use `recv()` and `send()` for special purposes
  - `readv()` and `writev()` for the hardcore
- Then either `close()` it
  - or use `shutdown()` if you need to just shut down in one direction

# Who called me?

- You are told on the accept()
- And can call getpeername() at any time (just returns what accept() would have told you had you asked)
- You can call getsockname() to find out your own name if you have forgotten

# How do I write the server logic with fork?

- Often uses `exec()` to replace the child with a new process, and `dup()` or `dup2()` to put the socket onto file descriptors 0 (`stdin`), 1 (`stdout`) and 2 (`stderr`).

```
pid = fork();
if (pid == 0) {
    /* In child process */
    close(sock);
    dup2(newsock, 0);
    dup2(newsock, 1);
    dup2(newsock, 2);
    execl ("/some/binary", "binary", "argument", 0);
    /* NOTREACHED */
    exit(1);
}
else {
    /* Parent process */
    if (pid == -1) {
        perror("fork");
        return 1;
    }
    else {
        close(newsock);
    }
}
```

# You need fork() earlier

- Network services usually are “daemons”: processes which run unattended without user interaction, started when the machine boots but also available to start/restart from the command-line.
  - Alternative is “inetd”, but this has fallen from favour.
- Therefore, a process needs to start, put itself into the background and detach from its immediate environment so it will run indefinitely.

# Daemons

- Huge amounts of voodoo, but in essence you need to:
  - Detach from a “controlling tty” and make sure all output goes to files/syslog/null.
  - When ^C is hit, everything for which the tty is controlling gets SIGINT
  - Become a session group leader
  - Make sure there is no-one wait()ing for you

# Method 1: call daemon()

- Unfortunately, not portable: not in POSIX, but is in most Linuxes and (surprisingly) recent Solaris and OSX.
  - I will be OK with people using this in submissions
  - Various greybeard people will raise various objections, not all of them totally invalid, about what it leaves the child able to do.

# Method 2: fork, fork

- `fork()` once, parent exits.
- `setsid()`
- [[ Drop privilege with `setuid()` ]]
- `fork()` again, parent exits
- `close()` 0, 1, 2, re-open as logfiles or `/dev/null`
-

# Being root

- Traditionally, you needed to be root to `bind()` to ports below 1024. The reasons are irrelevant in 2017 but the problem persists.
  - Risk created by non-root `listen()`ing on <1024 far less than risk created by daemons running as root
- Possible to give that power to specified users or non-root processes with very modern Unixes, or indeed turn the restriction off, but sadly rarely used.
- Therefore, best practice is to `bind()` to <1024 while root, then `setuid()` to a less privileged user (`httpd`, `webservd`, etc).
- More complex if you then need to log real users in: architecture of `sshd` very tricky to minimise section run as root.

# Fun with setuid()

- Root can become anyone
- No-one else should be able to do this without root's help
- Utterly renouncing your power to be root ever again  
sadly error-prone and non-portable, because of  
real/effective uid split introduced in some Unixes.
- Probably strongest to setuid after first, before  
second `fork()`. Speak to an expert in your  
precise operating system.

# Best Practice

- Run one daemon, isolated from everything else, in a Docker instance, Solaris Zone, \*BSD jail, etc.
- If you can't do that, investigate chroot().
- Daemons that run long-term as root are wrong, wrong, wrong and almost always represent a major security threat.
  - Any buffer overrun = r/w compromise of every file on machine.

# More Voodoo

- Older texts may show complex code to avoid every ending up in the position where file descriptors 0, 1 and 2 in a process are all closed or, more generally, where a process has no open file-descriptors, even momentarily.
- Including opening /dev/null onto fd 0 “just in case” and avoiding `close(0); close(1); close(2); dup2(s, 0)...`
- There was a bug in 4.2bsd in the early 1980s, fixed within about six months. And yet still the superstitions remain.
  - There’s a good mini-project in trying to trace the origins of all the myths and legends around how you daemonise a process

# `select()` or `poll()`

- What if you want to read and process two sockets at the same time?
  - `select()` or `poll()`, depending on your heritage
    - These days, `select()` is usually a wrapper around `poll()`, or sometimes vice versa, with same kernel code used. On older systems one was “better” than the other, but not today.
    - Allow you to specify one or more file descriptors, and returns telling you which are safe to read without their blocking
    - Almost always preferable to non-blocking I/O (tendency to burn 100% CPU) and signal-driven I/O (difficult to get right)
    - Again, today worth considering threads

# Buffered I/O

- Using `read()` and `write()` and their networking analogues gets a system call (ie, a context switch) on every use
- This is exactly what stdio seeks to avoid
- You can if you want use `fdopen()` on Unix to get a buffered view of a socket. This is generally a good idea (scatter/gather I/O with `readv()` and `writev()` is only for the keen).
  - Similar facilities in other languages/OSES

# Summary: TCP client

- `s = socket (PF_INET6, SOCK_STREAM, ...)`
- `bind (s)`
- `connect (s)`
- `while (1) { read(s) or write(s) }`
- `close (s) or shutdown (s)`

# Summary: TCP Server

- `s = socket(PF_INET6, SOCK_STREAM, ...)`
- `bind(s)`
- `listen(s)`
- `while (1) { ns = accept (s); fork();`
  - `/* child */ close(s); while (1) { doio  
(ns) } close (ns);`
  - `/* parent */ close (ns);`
- Failure to close ns in parent is classic long-term leak-and-fall-over

# HTTP and Friends

I.G.Batten@bham.ac.uk

# FTP(s)

- Wide range of protocols available to “send a filename, receive a file”.
  - **FTP**, which we will discuss later, probably oldest and certainly ugliest
  - **kermit** and **uucp** different ways to use serial lines (complete with own “transport” layers)
    - uucp mostly Unix-only, still shipped on OSX
    - kermit multi-platform
    - both can be coerced into running over a TCP connection

# pre-HTTP

- Range of protocols for finding resources to then fetch (probably with FTP)
  - gopher
  - archie
  - There were others
- All completely killed by rise of HTTP and (later) Search Engines

# Why HTTP?

- Hypertext Transport Protocol
- Originally designed to deal with downloading HTML with support for hyperlinks (“HTTP GET”)
- Extensible to a wide range of other tasks with other commands (“HTTP POST”, “HTTP PUT”)
- Flexible concept of URL means that it can do many other tasks apart from shipping files

# Why HTTP?

- Protocol is relatively easy to implement, but very flexible
- Protocol is not a screaming nightmare for networking engineers, unlike FTP
- Protocol decouples names from things (URLs look like Unix paths, but don't have to be)

# Basic structure

- Client sends some (optional) options
- Client sends command
- Client sends a blank line
- Server sends a status
- Server sends some commentary and information
- Server sends a blank line
- Server sends some data

# Lines

- This being the Internet, lines are terminated with \r\n (carriage-return line feed, aka control-M control-J, aka 0x0d, 0x0a)
- “man ascii”
  - Just to be annoying, Unix convention is \n, DOS convention is \r.

# Example

- We are examining the behaviour of the command  
“curl -v -6 \  
https://www.batten.eu.org/index.html”

# HTTPS vs HTTP

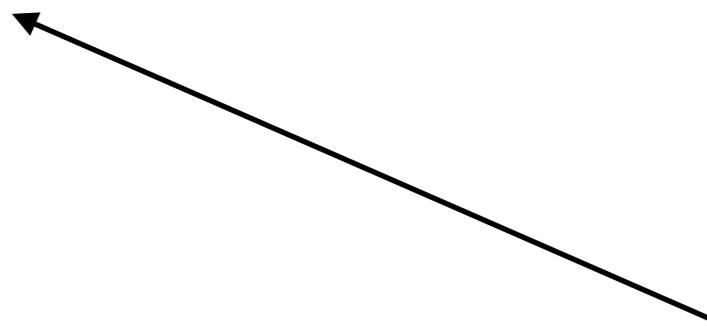
- HTTP (port 80) talks HTTP (amazingly)
- HTTPS(port 443) talks HTTP over an immediately-negotiated TLS session

# First, TLS is set up

```
* Hostname was NOT found in DNS cache
* Trying 2a00:1630:66:25:87:65:43:21...
* Connected to www.batten.eu.org (2a00:1630:66:25:87:65:43:21) port 443 (#0)
* successfully set certificate verify locations:
*   CAfile: none
  CApath: /etc/ssl/certs
* SSLv3, TLS handshake, Client hello (1):
* SSLv3, TLS handshake, Server hello (2):
* SSLv3, TLS handshake, CERT (11):
* SSLv3, TLS handshake, Server key exchange (12):
* SSLv3, TLS handshake, Server finished (14):
* SSLv3, TLS handshake, Client key exchange (16):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSLv3, TLS change cipher, Client hello (1):
* SSLv3, TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / DHE-RSA-AES256-GCM-SHA384
* Server certificate:
*   subject: C=GB; CN=batten.eu.org
*   start date: 2016-09-17 12:07:45 GMT
*   expire date: 2019-09-17 12:07:45 GMT
*   subjectAltName: www.batten.eu.org matched
*   issuer: C=IL; O=StartCom Ltd.; OU=StartCom Certification Authority; CN=StartCom Class 1 DV Server CA
*   SSL certificate verify ok.
```

# What is sent

```
GET /index.html HTTP/1.1
User-Agent: curl/7.38.0
Host: www.batten.eu.org
Accept: */*
```



Note there is a blank line here

# Line formats

- The **is no colon** in the GET/POST/etc command line (and in the status response to it)
- There **is a colon** in the remaining option lines.
  - Format taken from email headers
  - Key: Value
  - Lines starting with whitespace are continuations of the previous line. Yes, this is a pain to code.

GET /index.html HTTP/1.1

- Operation (others are POST, the less used PUT, the far less used DELETE, the sometimes useful HEAD)
- Name of resource to be fetch. Note, does **not** include hostname: dates back to era when there was a 1:1 relationship between IP numbers and name-spaces
- Note that this isn't the (obsolete) HTTP 1 protocol.

# User-Agent: curl/7.38.0

- Optional, but useful: sometimes a server will send different content based on its knowledge of the capabilities of the client
- Remember, people can lie (most browsers have a debug option to allow you to set pretty well any User-Agent field)

# Host: www.batten.eu.org

- Indicates the namespace, and allows multiple virtual servers to live on one IP number
- Historically, this required one IP number per namespace, which clearly isn't going to end well for (eg) Wordpress — even if they had enough IP numbers, there are limits to how many you can stick on one interface

Accept: \*/\*

- I will accept any content type (cf. `text/html`).

# What comes back?

```
HTTP/1.1 200 OK
Server: nginx/1.11.4
Date: Mon, 24 Oct 2016 21:17:31 GMT
Content-Type: text/html
Content-Length: 656
Last-Modified: Sun, 18 Sep 2016 20:47:08 GMT
Connection: keep-alive
ETag: "57defd4c-290"
Strict-Transport-Security: max-age=31536000
Public-Key-Pins:
    pin-sha256="z0rKil8EjF1RmB6shUYfITbsq2+r8HWhTgj0wIFtBcI=";
    pin-sha256="YwBM31CHmLowoHjIwSGTZX5PBj4w9i0Gt9dMDXpU87M=";
    pin-sha256="+Z7a6RhW5wAU+PTNmoAy2p/fVgQUdxXQ6U57dI7CK5E=";
    max-age=15552000
Accept-Ranges: bytes
```

# Or...

HTTP/1.1 200 OK

Date: Mon, 24 Oct 2016 23:06:08 GMT

Server: Apache

Vary: Accept-Encoding

Content-Type: text/html; charset=utf-8

# HTTP/1.1 200 OK

- Only compulsory element
- List of error codes taken from (essentially) SMTP and previously FTP
- See also 404 Not found, 403 Forbidden, etc

Server: nginx/1.11.4

Date: Mon, 24 Oct 2016 21:17:31 GMT

- Useful commentary
- Case can be made that both are revealing information of possible use to attacker, although (a) it's trivial to fingerprint servers and (b) everyone synch's their clock these days.

Content-Type: text/html

Content-Length: 656

- Type allows client to deal with content appropriately (in this case, hint to render it as HTML)
- Content-Length can optimise subsequent `read()`
  - Might be missing if length unknown prior to sending

# End of File: Nightmare

- FTP opens a separate TCP connection for each file (using a “control connection” to orchestrate them)
- File can then be sent unmodified (“BINARY MODE”) followed by closing the connection to mark its end.
  - Inefficient, complex, hard on the network
- SMTP, POP3, others use “.” on a line on its own to mean “end of file”, with an extra dot stuffed into lines that start with a dot.
  - Means sender and receiver have to scan whole file, and inserting a single character breaks the flow of “read block, send block”.
  - Prevents protocol from being “8 bit clean” as it has to avoid sequence “\r\n.\r\n” at all costs.
  - One of the most broken things about existing email standards.
- Byte count (IMAP, HTTP) allows sender to stat() a file and then read it as one operation and send it as one operation, and allows receiver to just route next  $n$  bytes to the client software. HTTP doesn’t mandate \r\n for data, and very high performance IMAP servers store messages with \r\n so they can do the same “stat(), open(), read()” trick.

Last-Modified: Sun, 18 Sep 2016 20:47:08 GMT  
ETag: "57defd4c-290"

- Last-Modified and ETag helpful for caching (storing copies of someone else's content)
  - Etag is opaque fingerprint of exact contents, which is a stronger check than comparing the last-modified string.

## Connection: keep-alive

- Indicates to client that the connection can be re-used for more requests once this one has completed
- Saves the building of a fresh connection and has performance benefits we will talk about later
- Early browsers built a fresh TCP connection for every item on a webpage: catastrophic performance

`Strict-Transport-Security: max-age=31536000`

- HSTS extension: says that this namespace is only available from HTTPS encrypted HTTP, and all requests for the unencrypted form should be rewritten to the encrypted version for the next 365 days
- Ignored unless send over HTTPS, cached by client
- Can be “pre-loaded”
- Use this whenever possible: all content should be encrypted to ensure authenticity

Public-Key-Pins: pin-sha256="z0rKil8EjF1RmB6shUYfITbsq2..."  
max-age=15552000

- HPKP Experimental Extension
- Lists the set of public keys that should be accepted for this domain over the coming period of time (here 180 days)
- HSTS+HPKP+HSTS pre-load substantially reduce risk of “rogue AP”
  - Network Security next semester

## Accept-Ranges: bytes

- Indicates server is willing to accept requests for partial transfers, specified by byte-offsets

# Then the content

Note blank line

Accept-Ranges: bytes

```
<html>
<head>
<!-- <base href="https://www.batten.eu.org/"> -->
<link rel="stylesheet" type="text/css" href="https://www.batten.eu.org/batten.css">
<title>batten.eu.org</title>
</head>
```

# Flexibility

- Trivially, the requested name can be a file in a directory rooted somewhere in the filestore
- But can just as easily be a key into a database, a parameter to a program, whatever.
- The server converts URNs into content however it wants (and, presumably, the user expects)

# By convention

- You can pass multiple parameters with
  - `/some/path?var1=value1&var=value2`
- But note this is just a convention (“CGI”, Common Gateway Interface) and nothing stops you turning URL `/foo/bar/baz` into a call to `foo` with arguments `bar` and `baz`.
- Passing arguments via GET makes them trivially available in logs

# POST

```
curl -d foo=bar -v -v -6 https://www.batten.eu.org/index.html
```

```
> POST /index.html HTTP/1.1
> User-Agent: curl/7.38.0
> Host: www.batten.eu.org
> Accept: */*
> Content-Length: 7
> Content-Type: application/x-www-form-urlencoded
```

# HEAD

```
HEAD /index.html HTTP/1.1
User-Agent: curl/7.38.0
Host: www.batten.eu.org
Accept: */*
```

Request sent

```
HTTP/1.1 200 OK
Server: nginx/1.11.4
Date: Mon, 24 Oct 2016 21:58:16 GMT
Content-Type: text/html
Content-Length: 656
Last-Modified: Sun, 18 Sep 2016 20:47:08 GMT
Connection: keep-alive
ETag: "57defd4c-290"
Strict-Transport-Security: max-age=31536000
Public-Key-Pins:
  pin-sha256="z0rKil8EjF1RmB6shUYfITbsq2+r8HWhTgj0wIFtBcI=";
  pin-sha256="YwBM31CHmLowoHjIwSGTZX5PBj4w9i0Gt9dMDXpU87M=";
  pin-sha256="+Z7a6RhW5wAU+PTNmoAy2p/fVgQUdxXQ6U57dI7CK5E=";
  max-age=15552000
Accept-Ranges: bytes
```

Response Received

# PUT

- Permits file upload
- Subtle differences to POST, which we can talk about if we have time to discuss REST

# DELETE

- Rarely used for files today, but used in RESTful web APIs.

# Accept-Encoding

- Client can indicate that it can handle encodings (distinguish from encryption) of the content
- Accept-Encoding: deflate, gzip
- Server can (at its discretion) respond with
- Content-Encoding: gzip

# Languages

- Similarly for Accept-Language and Content-Language.
- And Accept-Charset and Content-Charset.
- More generally, client sends requests and capabilities, server indicates formats and content.

# Usernames and Passwords

- HTTP Basic Auth is a cesspit of security problems
- But, if we must...
- If there is no authentication in the request, it is rejected as follows:

HTTP 401 Unauthorized

```
WWW-Authenticate: Basic realm="Some Tag"
```

# Usernames and Passwords

- If you have a login and password for the domain, you send
  - Authorization: Basic aWdi0mlnYjEyMw==
  - Where does the gibberish come from?
  - Base64 encoding of username:password

```
igb@pi-one:~$ echo -n 'igb:igb123' | base64  
aWdi0mlnYjEyMw==  
igb@pi-one:~$
```

# HTTPS Only

- Obviously, don't do this over unencrypted channels!
- There is a challenge/response mechanism (“Digest”, RFC 2069) using hashes, but
  - it involves the server storing plaintext passwords
  - almost no-one implements it anyway.

# Cookies

- Another security cesspit
- And wildly abused for a wide range of things they aren't suitable for

# Cookies

- Any operation can return a “Set-Cookie” operation, which logs a key=value pair against (usually) the current domain and host
- Whenever a request is made for that domain and host, all relevant cookies are sent with “Cookie” header
- Cookies can have lifetimes, various security properties

# Cookies

- One common use is/was to use HTTPS to protect username and password, returning a cookie for successful authentication, and then rely on that being safe to pass unencrypted to prove authentication has happened
  - HTTPS no longer expensive, and should be used for all operations, particularly...
  - ...those involving cookies
- More generally, anything which involves storing client-side state: navigation, shopping carts, etc
- Best limited to a random session-id and nothing else, with everything else stored server-side.

# RESTful Interfaces

- Later lecture (was requested last year, so I have it good to go)
- Way to interact with APIs over standard HTTP

# Caching

- Rises and falls in popularity
- Idea is that you can keep a copy of “all the internet that matters” at the edge of a **consuming** network
  - Ian F brought the cs.bham.ac.uk caching strategy to ftel.co.uk; when I came back, I brought its end
- Alternatively, a more usefully, you keep a copy of “all our content from lots of slow machines” at the edge of a **producing** network
- Caches and proxies tend to be the same boxes: we will talk about proxying when we talk about NAT.

# Caching

- Usually done with special software (“squid” is the most common) or with special appliances
- Trick is to cache stuff which is static and used repeatedly, while not interfering with content that changes rapidly or is rarely accessed
- My gut feel is that 2016-stylee, all caching is out of favour

# Networking: Other Transports, NAT

[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# Transports in wide use:

- UDP: thin wrapper over IP, unreliable, unsequenced
- TCP: complete transport service, offers reliable, sequenced delivery with guarantee of either success or a positive failure indication.
- Together majority of Internet traffic

# RTP

- Real-time Transport Protocol
- Used to transport voice (telephony) and video (streaming) in some applications.
- Doesn't do anything you can't do yourself with UDP.

# Problems for voice and video

- Consistent timing
- Choice between dropping and catching up
- Trade off with buffering

# For telephony...

- Usual claim is anything over 35ms latency is problematic for conversation (“toll quality”)
  - Figure has no experimental basis
  - Partly about echo cancellation, partly about difficulty in maintaining conversation
- 35ms is easy to achieve in traditional telephone networks (roughly 10k km speed of light) but is difficult to achieve reliably in IP based networks with slow/congested local links.

# Reality is more generous

- Latency over networks with complex compression (“codecs”) is higher, GSM for example.
  - Although GSM has no “side tone”, which is why people shout in mobile phones.
- Increasingly, people will tolerate GSM-quality voice (~3kbps) rather than “toll quality” voice (~56kbps).
- Counter example is difficulty people have with geo-stationary satellite communications (ie 1960s/70s phone calls to Australia), but there latency approaches 500ms with heavy echo cancellation.

# RTP

# RTP

- Each packet contains a sequence number, which can be used to spot gaps and re-order packets.
- But each packet also contains a time-stamp (resolution decided when the stream is set up)
  - Say, 8KHz for voice, as voice is most commonly 8KHz sampling rate, 4KHz bandwidth
  - Or frame-rate for video

# Difference with TCP

- No acknowledgements.
- Receiver knows when packet was sent, and how many were sent.
- Receiver can therefore discard packets in order to stay “current”, or can pause replay to wait for arrival of missing packets, or some other strategy.
- Duplicates are detected.

# RTP Setup

- RTCP (“real time control protocol”) used to set up video replay and similar
- SIP (“session initiation protocol” used to set up Voice over IP telephony.
- Co-ordination of RTCP/SIP session with RTP stream is difficult for firewalls: in voice-land, “Session Border Controllers” combine SIP and firewalling, while emptying your wallet.
- Most video streaming now uses traditional TCP with sufficient buffering to deal with variation in latency, plus heavy compression with MPEG/etc.

# SCTP

- Stream Control Transport Protocol
- Attempt to tunnel traditional voice signalling (“SS7”) over internet.
- Again, UDP with a few extra facilities
- Largely moribund

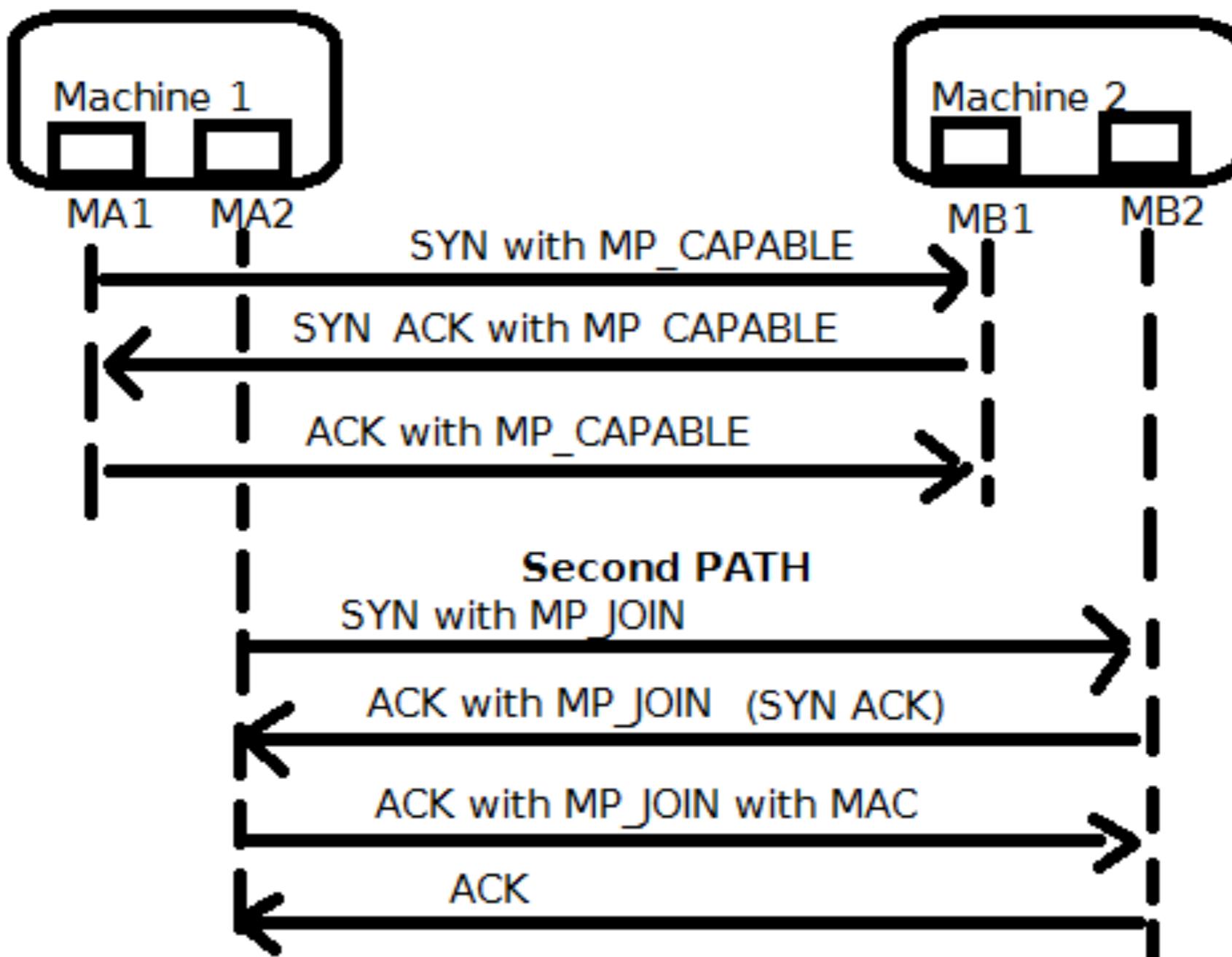
# DCCP

- Datagram Congestion Control Protocol
- Another UDP plus frills, again for time-sensitive delivery.
- Again, moribund
- General lesson: “UDP plus a bit” is too complicated if it is general, insufficiently attractive to implementors if it is too specific.

# Multipath TCP

- Now something more exciting!
- RFC6824 is well worth reading
- Allows multiple paths to be used by one TCP connection
  - For example, Wifi **and** 4G **simultaneously**

# Multipath TCP



# Not only performance

- By having a link multiplexed over WiFi and 4G, failure of one path appears as just some packet loss, and the link rapidly reconfigures.
  - This is very hard otherwise, as you will have different IP numbers in each realm
  - Also makes effective use of multiple network cards, particularly in networks with a lot of resilience / redundancy.

# New, but growing

- Implemented in iOS 7 et seq
- Reference implementation in Linux (much of the data centre world)
- Coming soon in Solaris (rest of the data centre world)
- Doesn't require significant application changes, most applications work unmodified (may require recompilation)
- Looks promising

# Address Translation

- Mechanism to extend scarce IP numbers
- Incidentally provides some security, although this was not a design goal and should be treated with care
- Breaks “end to end principle”
- Causes some people (such as me) to start shouting uncontrollably

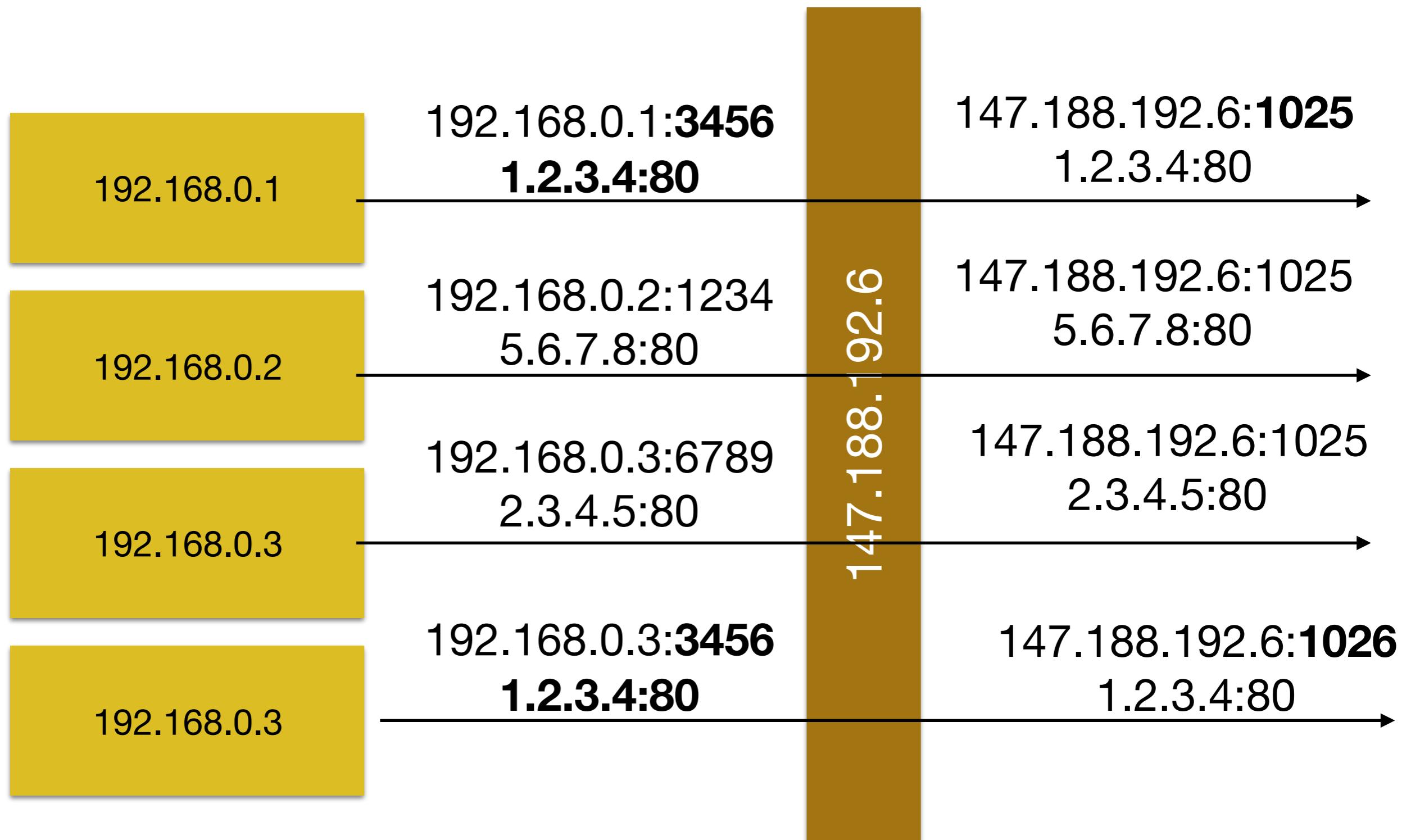
# Basic Principles

- Outbound NAT:
  - Connection is modified so that connections from multiple source IP addresses are encoded into port number space of a smaller number of addresses
- Inbound NAT
  - Connection is modified so that connections to multiple ports on a small number of IP addresses are expanded out to a large number of addresses

# Recall:

- TCP connection identified by source IP, source port, destination IP, destination port.
- So long as one element in the quad is different, it's a different (and distinguishable) connection
- Destination IP and port identify called service
- But the source can be changed

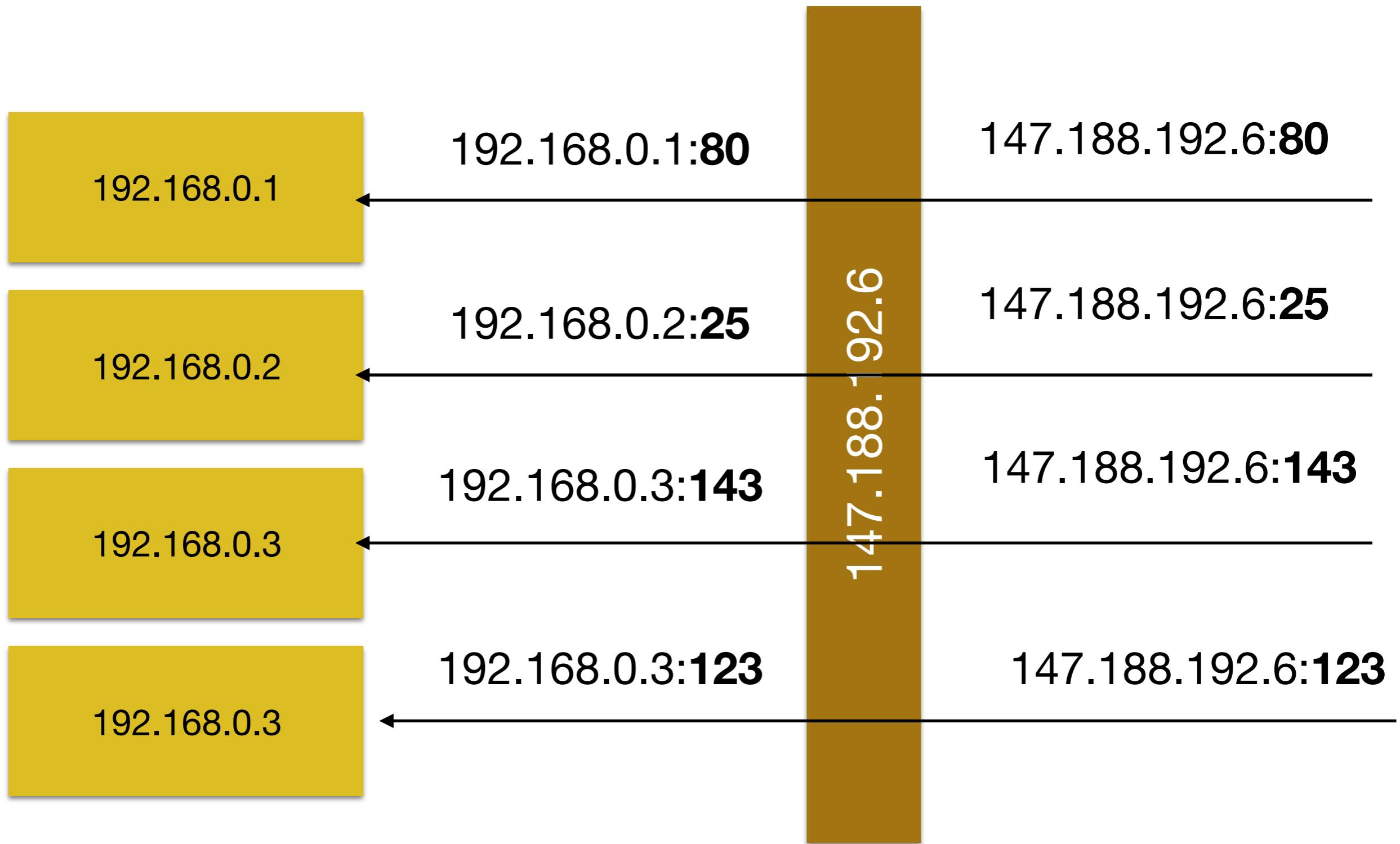
# Outbound (Source) NAT



# In reality...

- Often not necessary to overload port numbers as shown: each connection gets distinct source port number
  - Gives 65535 connections per IP number
- Large installations use multiple IP numbers at NAT point

# Inbound (Destination) NAT



# Inbound NAT

- Used to offer multiple services from single IP number (goes well with virtualisation to minimise attack surface)
- Also used in more complex situations to offer load balancing, failover, mobility, etc

# NAT for TCP

- NAT device sees “SYN” packet and builds a mapping between inside and outside addresses.
- Modifies TCP packet (including IP header, as involves change to source address to be own), recomputes check sums, sends packet
- On receipt of packets, looks at source IP and port and destination port, performs reverse mapping and sends packet.
- Tracks TCP state, and deletes entry from translation table when FINs have all completed.

# NAT for UDP

- No “state” as such.
- Rewrite outgoing UDP and then accept return packets until there is silence for 10s (typically).
- Can also impose limit on number of replies, as for example DNS.

**Interfaces**

Filter Rules	NAT	Mangle	Service Ports	Connections	Address Lists	Layer7 Protocols		
Tracking								
181 items out of 192								
		Src. Address	Dst. Address	Prot...	Connec... Type	Connec... Mark	P2P	Timeout
-	A	147.188.254.236:59643	81.187.150.214:443	5 (tcp)				00:00:03
-	A	147.188.254.236:59648	81.187.150.214:443	5 (tcp)				00:00:03
-	A	147.188.254.236:59636	81.187.150.214:443	5 (tcp)				00:59:20
-	A	147.188.254.236:59644	81.187.150.214:443	5 (tcp)				00:00:03
-	A	147.188.254.236:59647	81.187.150.214:443	5 (tcp)				00:00:01
-	A	147.188.254.236:59641	81.187.150.214:443	5 (tcp)				00:59:32
-	A	128.204.195.144:58643	81.187.150.213:514	5 (tcp)				00:56:38
-	A	147.188.254.236:51232	81.2.79.220:1701	17 (udp)				00:00:02
-	A	147.188.254.236:51916	81.2.79.220:4500	17 (udp)				00:00:02
-	U	10.92.213.81:56283	10.92.213.255:8612	17 (udp)				00:00:06
-		10.92.213.44:18057	10.92.213.231:53	17 (udp)				00:00:10
-		10.92.213.44:37433	10.92.213.231:53	17 (udp)				00:00:10
-		10.92.213.44:55851	10.92.213.231:53	17 (udp)				00:00:09
-		10.92.213.44:36604	10.92.213.231:53	17 (udp)				00:00:09
-		10.92.213.44:33758	10.92.213.231:53	17 (udp)				00:00:09
-		10.92.213.44:43311	10.92.213.231:53	17 (udp)				00:00:08
-		10.92.213.44:37913	10.92.213.231:53	17 (udp)				00:00:08
-		10.92.213.44:52613	10.92.213.231:53	17 (udp)				00:00:09
-		10.92.213.44:58780	10.92.213.231:53	17 (udp)				00:00:08
-		10.92.213.44:43735	10.92.213.231:53	17 (udp)				00:00:08
-		10.92.213.44:38163	10.92.213.231:53	17 (udp)				00:00:08
-		10.92.213.44:42354	10.92.213.231:53	17 (udp)				00:00:09
-		10.92.213.44:60147	10.92.213.231:53	17 (udp)				00:00:09
-		10.92.213.44:38870	10.92.213.231:53	17 (udp)				00:00:08
-		10.92.213.44:41196	10.92.213.231:53	17 (udp)				00:00:08
-		10.92.213.44:60612	10.92.213.231:53	17 (udp)				00:00:08

TCP, 60m timeout

UDP, 10s timeout

# Problems with NAT

- It's evil :-)
- Makes it very difficult to authenticate and log users
- NAT logging is part of “carrier grade NAT”, but requires time alignment of log on remote server and at the NAT point

# Timing Problems

- my.popular.dom.ain server 1.2.3.4 has abusive connection from 147.188.192.6:1234 at 10:25:40
- 147.188.192.6 logging (if available) shows 1234 used for connections to 1.2.3.4 by 192.168.0.1 at 10:25:10 and 192.168.0.2 at 10:25:50.
- NAT logs won't include URL, just IP number
- Who called my.popular.dom.ain? Requires **retrospective** knowledge of clock offsets.

# Logging Problems

- Most web logging does not record source ports. It can, but usually doesn't.
- So very difficult to request logs from NAT point, as there will be multiple connections to the same popular service, distinguished only by source port
- Claimed by law enforcement to be a serious problem.

# Delays the IoT

- Internet of Things implies universal connectivity
- NAT delays universal connectivity, by making RFC1918 IP numbers usable for client devices.
- “Carrier Grade NAT” can even use RFC1918 for customer lines, NAT’d once at customer border and again at ISP border.

# IPv6 has no NAT

- IPv6 does not require NAT, as plenty of addresses for everyone.
- IPv6 implementations don't support NAT
- There are already proposals for IPv6 NAT, because of (bogus) security concerns.

# NAT “Security”

- NAT is conceptually a stateful firewall: each TCP connection is being tracked for state, each UDP “connection” is being at least monitored for volume and duration
- Tendency to regard this as an actual firewall, cf. PCI-DSS requirement for NAT on low-end companies.
- NAT products not certified or designed for security
- To complicate matters, often common code (Linux NAT functionality is in iptables firewall).

# Inbound NAT

- This is particularly confusing for inbound NAT
- Inbound permits connection to port 80 on outside of NAT to appear as connection to port 80 on internal machine.
- There is **no security** in this at all: even if the NAT point is regarded as a firewall, this is a complete pass-through.
- Yet inbound NAT is still used as a “security” feature.

# Complications for NAT

- Protocols which embed IP numbers in control streams break under NAT, because the IP numbers are wrong.
- FTP is the worst offender, and requires custom NAT modules to re-write the contents of the control stream.
- Modified FTP (“Passive Mode”, “PASV”) is better solution, or just don’t use FTP (please, just don’t use FTP).

# Complications for NAT

- IP-address based authentication schemes lose resolution, because all of a site appears as one address.
- Such schemes were arguably broken anyway, but are popular in academic publishing. Solutions involve complex proxying, but real solution is better authentication strategies.

# Extra NAT protocols

- UPnP (“Universal Plug ‘n’ Play” — who, one has to ask, names these protocols?)
- Allows “inside” devices to communicate with a NAT point and request inbound NAT, effectively automating a bypass of any firewall.
- Used heavily in residential products like Web Cams and “personal cloud” type products, as well as VoIP.
- UPnP is a dream for malware, as it makes opening a connection to a command and control server particularly easy.

# Summary

- Quite a few alternatives to TCP and UDP, mostly used only for voice.
- Multipath TCP looks very promising.
- NAT is a necessary evil, but please, IPv6.

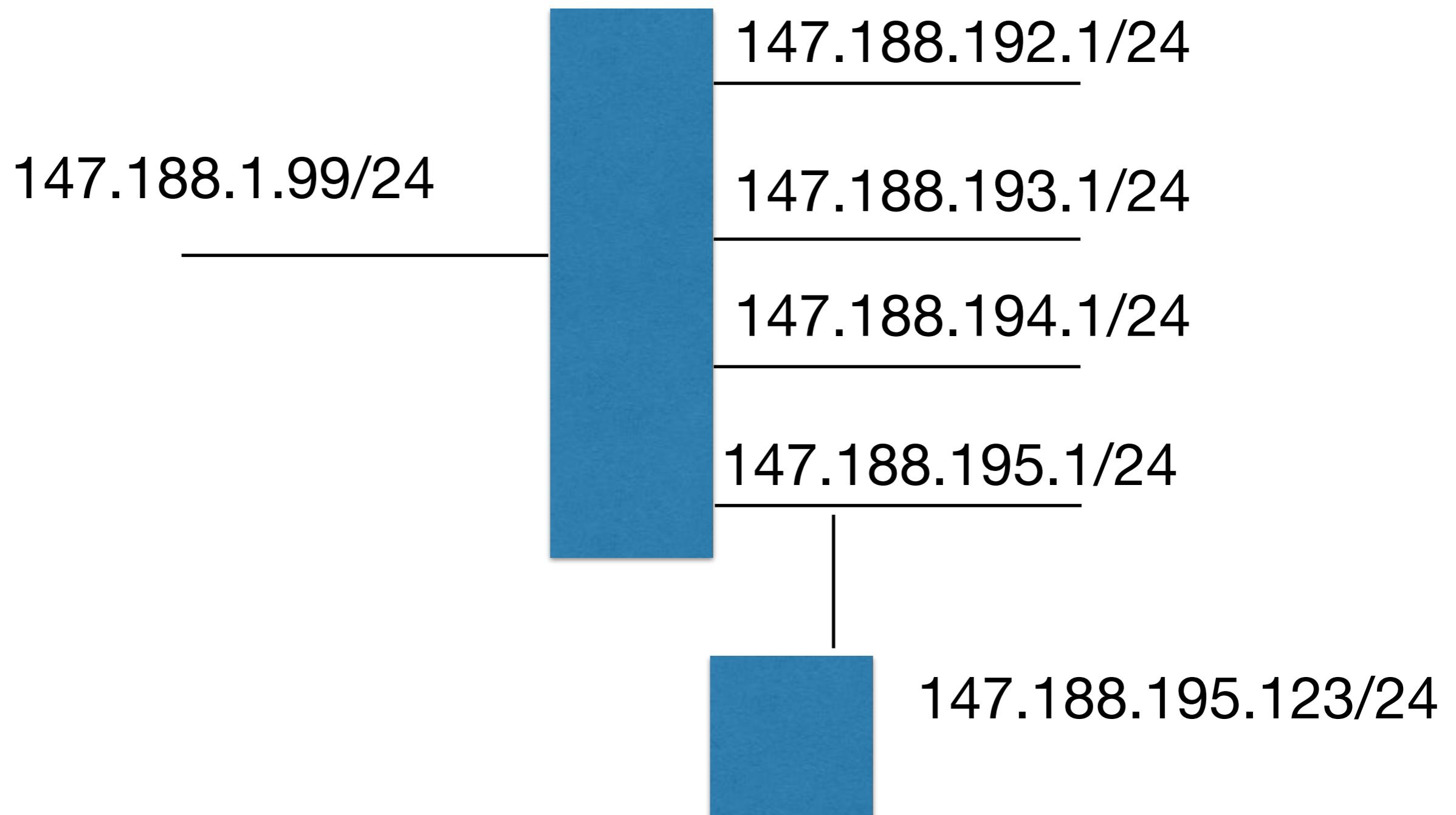
# Networks 12: Interfaces, Plural

[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# Multi-Home, Multi-Path, etc

- Typical desktop machine only has one interface (although virtual networks might be in use).
- Typical server has multiple interfaces
- What are they used for?
  - Lots of things!

# Routing



# Why do this?

- Historically, LAN routers were very expensive (two interface Cisco IGS/L or 2514 was price of decent Unix workstation) with difficult licensing terms.
- Firewalling and/or routing protocols made this worse (required extra-cost feature sets)
- Unix/Linux box, probably an old one, quite attractive proposition.

# Why do this?

- Other reason performance.
- Early 1990s: big fileservers can saturate 10baseT, multiple times
- Late 1990s: big fileservers can saturate 100baseT or FDDI multiple times
- Sometimes need to segregate backup traffic from production traffic

# Programming issues?

- Complicates writing servers, and clients need to be aware.
- Usual technique is to bind to INADDR\_ANY, with specific port for server and an arbitrary port for a client.
  - You get this for free, as the local address is usually bzero'd and INADDR\_ANY is all-zeros
  - Do you want to listen / send on / to all interfaces?
  - How do you ensure packets exit from correct interface?

# How are source addresses set?

- Machine has multiple IP numbers, so the source address is chosen either by the programmer or the operating system
- Operating system will, with some variations, choose the IP number of the interface the packet is routed out over, either per-packet (UDP) or per-connection (set at SYN-time)

# Clients to multihomed servers

- Clients should cope with servers with multiple interfaces, not all of which are up
- Client should try each interface in turn (doing it in parallel usually considered a bad idea)
- Caching “live” interface considered OK, choose a sensible time-out

# Simple Clients

- Naive clients assume that the server they are connecting to only has one IP number, almost certainly IPv4 (gethostbyname is IPv4-only)
- Various DNS hacks mean that there is some “round robin” behaviour, and connections will be to some extent spread over multiple interfaces

```
ians-macbook-air:MSCJAVA14 igb$ dig +short rb2011-1.batten.eu.org  
81.2.79.220  
81.187.150.214  
ians-macbook-air:MSCJAVA14 igb$ dig +short rb2011-1.batten.eu.org  
81.187.150.214  
81.2.79.220  
ians-macbook-air:MSCJAVA14 igb$
```

# Slightly Smarter Clients

- Will try all interfaces in order returned by `gethostbyname`
- Not in parallel: that will establish  $n$  connections and then drop  $n-1$ , which is abusive

# Very Smart Clients

- Use getaddrinfo
  - Confusingly, getaddrinfo is names->addresses, like gethostbyname, getnameinfo is addresses->names, like gethostbyaddr.
- Returns IPv6 and IPv4 addresses
- Probably OK to parallelise IPv4 and IPv6, perhaps with one or other leading slightly, practice will emerge, policy best left to libraries.

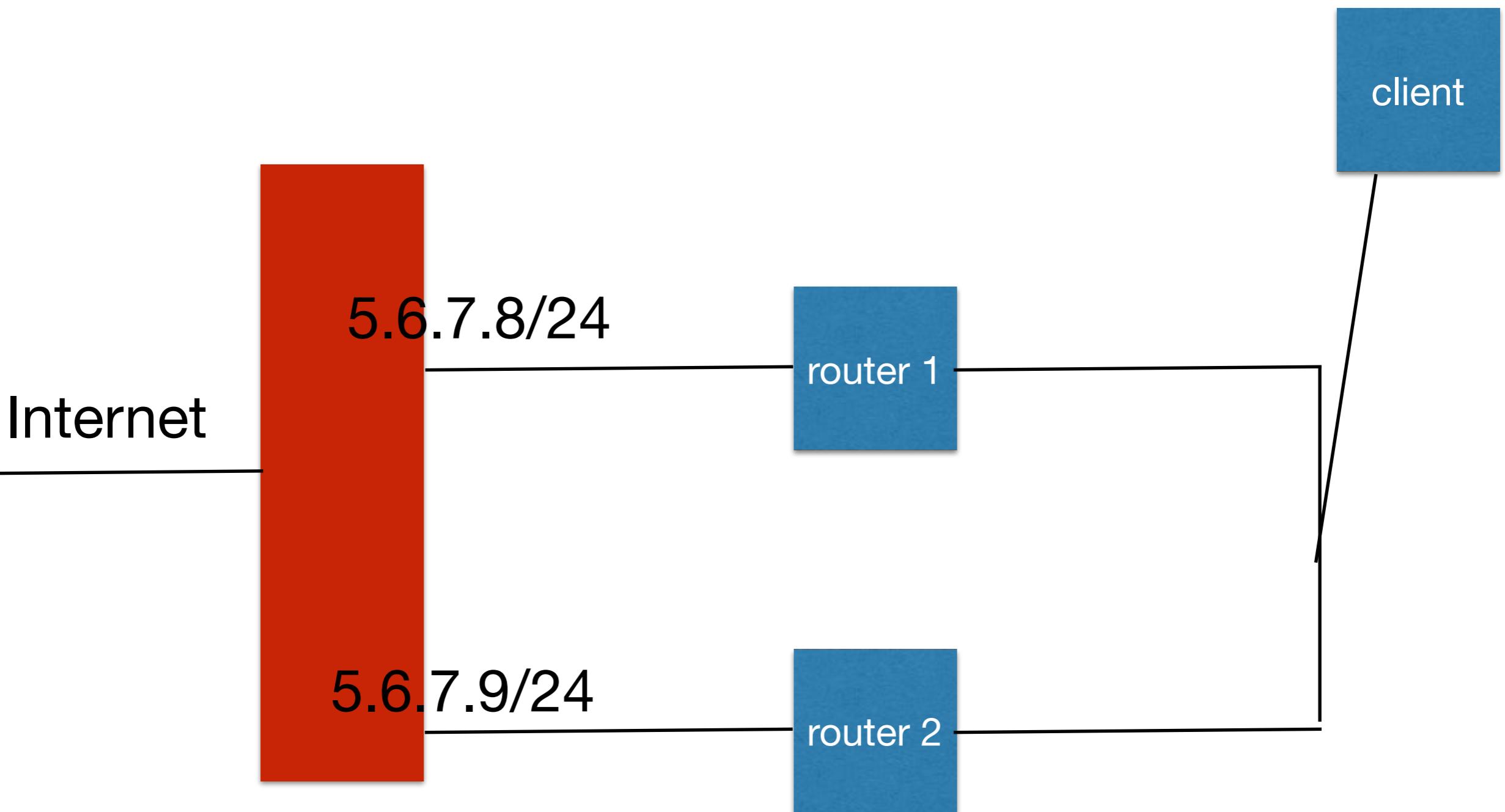
# Use getaddrinfo anyway

- `gethostbyname` and `gethostbyaddr` are **vile**
- Worst problem is that they maintain static state, and therefore aren't re-entrant (you can't use them in threads)
- Re-entrant versions exist (`gethostbyname_r`, etc) but are messy to use
- `getaddrinfo` solves all this anyway
- Used correctly means code is IPv6-aware from the outset

# Servers

- What about on the server side?

# Problem



# Asymmetric Routes

- Packets may travel via different routes out and back
- But must cross same NAT point or same stateful firewall with same source address: almost impossible to deal with connections where NAT/FW only sees half of traffic or it is mis-labelled
  - Interesting research topic, incredibly hard to implement with modern speeds and feeds

# UDP

- It's important that if you receive a query sent to a particular IP number, the response goes out from the same IP number
- So query sent to 5.6.7.8:53 must have a reply coming from 5.6.7.8:53, as otherwise sender will discard it (even if a firewall will pass it, which it usually won't)

# UDP

- If routing table changes, then source address of packets to same destination may change
- And you cannot guarantee packets arrive over the interface that a reply will leave over (asymmetric routing)

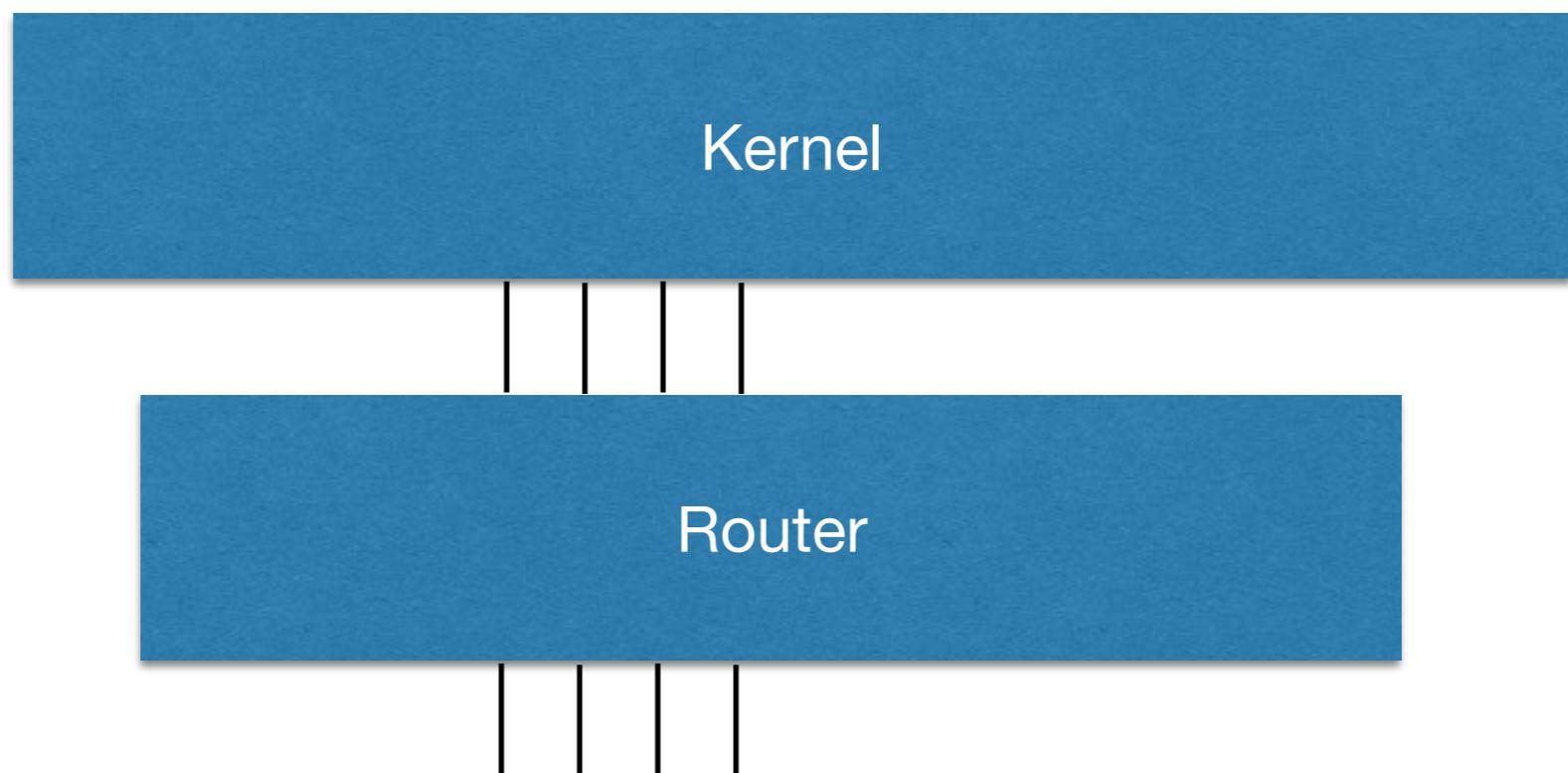
# UDP

- Two techniques:
  - listen on 0.0.0.0:53, look at destination address on each packet delivered to application, copy destination address into source address of response
  - listen on each interface individually, so you have (effectively) multiple copies of the application, each only knowing about one interface
  - second technique more flexible, but means application needs to iterate over installed interfaces (SIOCGIFCONF is notoriously painful to use) and will need to respond to changes to interface config.

# Internal Routing

- You can set the source address to **any** local addresses, and the kernel will still route the packet out over the interface it believes to be closest to the destination
  - In fact, most operating systems will let you set UDP source addresses to any address, without checking it is local. You might need to be root, but you had to be root to bind to port<1024 anyway.

# Conceptually...



# Forwarding Switch

- All operating systems now have a switch to determine if packets are forward between interfaces
  - Used to be default on for anything with more than one interface, now always default off
- Behaviour if you try to send packet with source address X to a destination routed via interface Y with forwarding turned off slightly unclear
  - but in practice, switch only applies to non-local packets arriving over interfaces

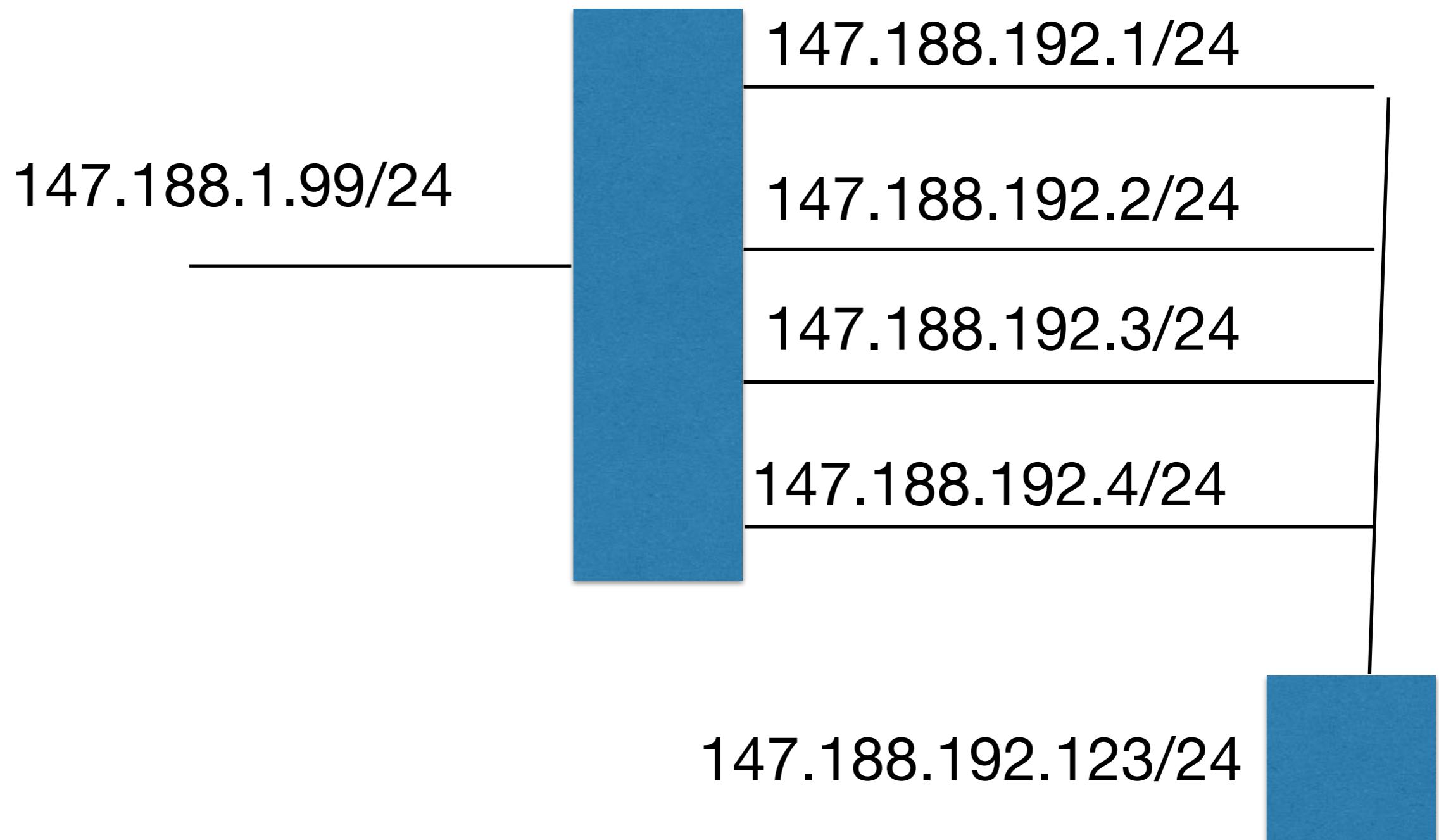
# TCP

- Kernel ensures that replies have same source address as original destination address (definition of TCP)
- So much easier to just listen on 0.0.0.0: reasons to listen on specific addresses more about security than connectivity
- No control over which interface packets leave over

# Multihomed clients

- Might need to explicitly set a source address: actually trickier than server for TCP
- Usually config option: by default bind outgoing requests to INADDR\_ANY and let operating system decide, but give administrator option on a per-application basis

# Interfaces into same network



# Similar problems...

- Usually done for performance or reliability reasons
- Better techniques available if switches support it (see later)
- Usually one IP number is treated as primary, and used for all source addresses, with packets spread out over interfaces.
- Other interfaces might listen, but are deprecated (some operating systems have DEPRECATED state, which means address is never used as source)

# Useful for transition

```
mailprod0: flags=100001000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,PHYSRUNNING> mtu 1500 index 2
          inet 147.188.192.250 netmask ffffff00 broadcast 147.188.192.255
mailprod0:1: flags=100001000843<UP,BROADCAST,RUNNING,MULTICAST,IPv4,PHYSRUNNING> mtu 1500 index 2
          inet 147.188.192.251 netmask ffffff00 broadcast 147.188.192.255
```

147.188.192.251 address of older system, now decommissioned: doesn't need physical interface

# Link-Agg

- If switch supports it, you can put interfaces into a Link Aggregation group, controlled with LACP
- Appears to operating system as one “**logical**” interface, one IP number, one entry in routing table, etc.
- Modern solution for performance and availability
  - Failure detection much faster and simpler
  - Fancy equipment lets you link-agg to multiple switches, or at least multiple independent cards in same switch

# Benefit of LinkAgg

- Only one IP number
- Fast failover
- Efficient multiplexing and load spreading (header hashes, sometimes done by hardware)

# Problem of LinkAgg

- Only goes as far as your nearest switch
- Doesn't provide for (say) spreading over multiple internet connections

# VLANs

- Way to get multiple networks delivered over single cable
- Ethernet frames have optional extra “tag” (0–1023 minimum, 0–4095 more common) identifying which network it is
- Untagged frames are in “default” network
- OS sees them as two separate networks

# VLAN host-side

Untagged

```
igb@pi-one:~$ ifconfig -a
eth0      Link encap:Ethernet Hwaddr b8:27:eb:8c:0c:34
          inet addr:10.92.213.231 Bcast:10.92.213.255 Mask:255.255.255.0
          inet6 addr: 2001:8b0:129f:a90f:3141:5926:5359:1/64 Scope:Global
          inet6 addr: fe80::ba27:ebff:fe8c:c34/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:44938697 errors:0 dropped:0 overruns:0 frame:0
          TX packets:33296383 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:49825123 (47.5 MiB) TX bytes:441607425 (421.1 MiB)

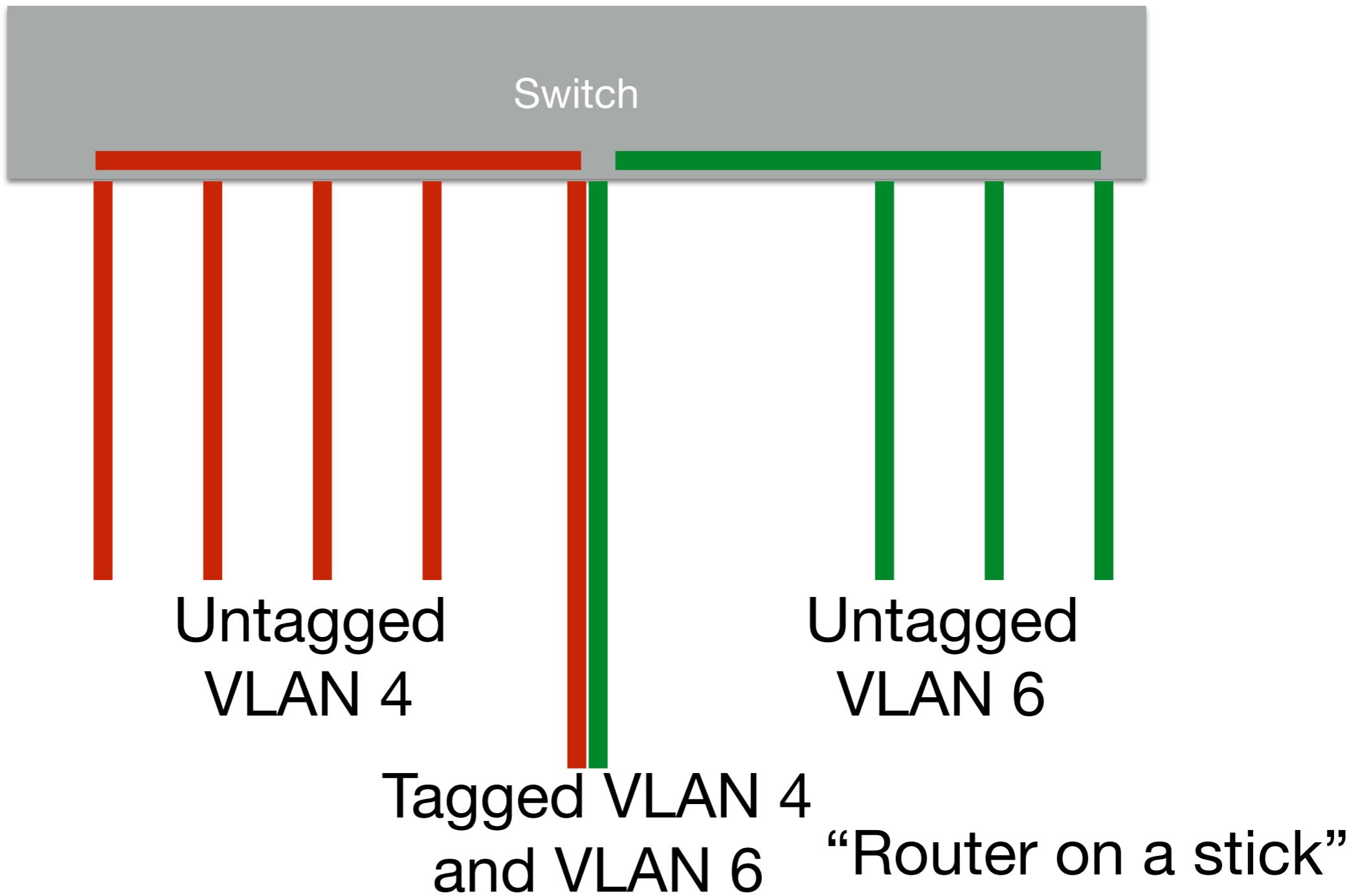
eth0.5    Link encap:Ethernet Hwaddr b8:27:eb:8c:0c:34
          inet addr:81.187.150.211 Bcast:81.187.150.223 Mask:255.255.255.240
          inet6 addr: 2001:8b0:129f:a90e:3141:5926:5359:1/64 Scope:Global
          inet6 addr: fe80::ba27:ebff:fe8c:c34/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:25759488 errors:0 dropped:0 overruns:0 frame:0
          TX packets:21680755 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2927238990 (2.7 GiB) TX bytes:2669983251 (2.4 GiB)
```

VLAN tag = 5

# VLAN on switches

- Each VLAN has a tag
- Each port of a switch can output one or more VLANs, one of them optionally untagged
- Flexible, and a good way to confuse yourself
- Ports outputting untagged frames sometimes called “access”, tagged frames “trunk”
- Can mix tagged and untagged on one cable, untagged packets assumed to be VLAN 0

# Logically divide switches



# Untagged ports

- AKA “access ports”
- Devices attached to them don’t have to understand VLANs
- Packets from one VLAN on the switch are transmitted with the tag stripped
- Packets arriving without a tag have a pre-defined tag added
- Can also have tagged packets mixed in, to/from VLAN-aware devices
  - Freshly installing a machine to use a tagged interface is sometimes tricky, so attractive to use untagged for “primary” interface while still having access to other VLANs, cf. my home example

# Tagged Ports

- All packets are transmitted with a tag, and only tagged packets are expected to be received
- Common for inter-switch links
- Tags can be filtered for security reasons (not very good security)
- Tags can also be re-written, although that way madness lies

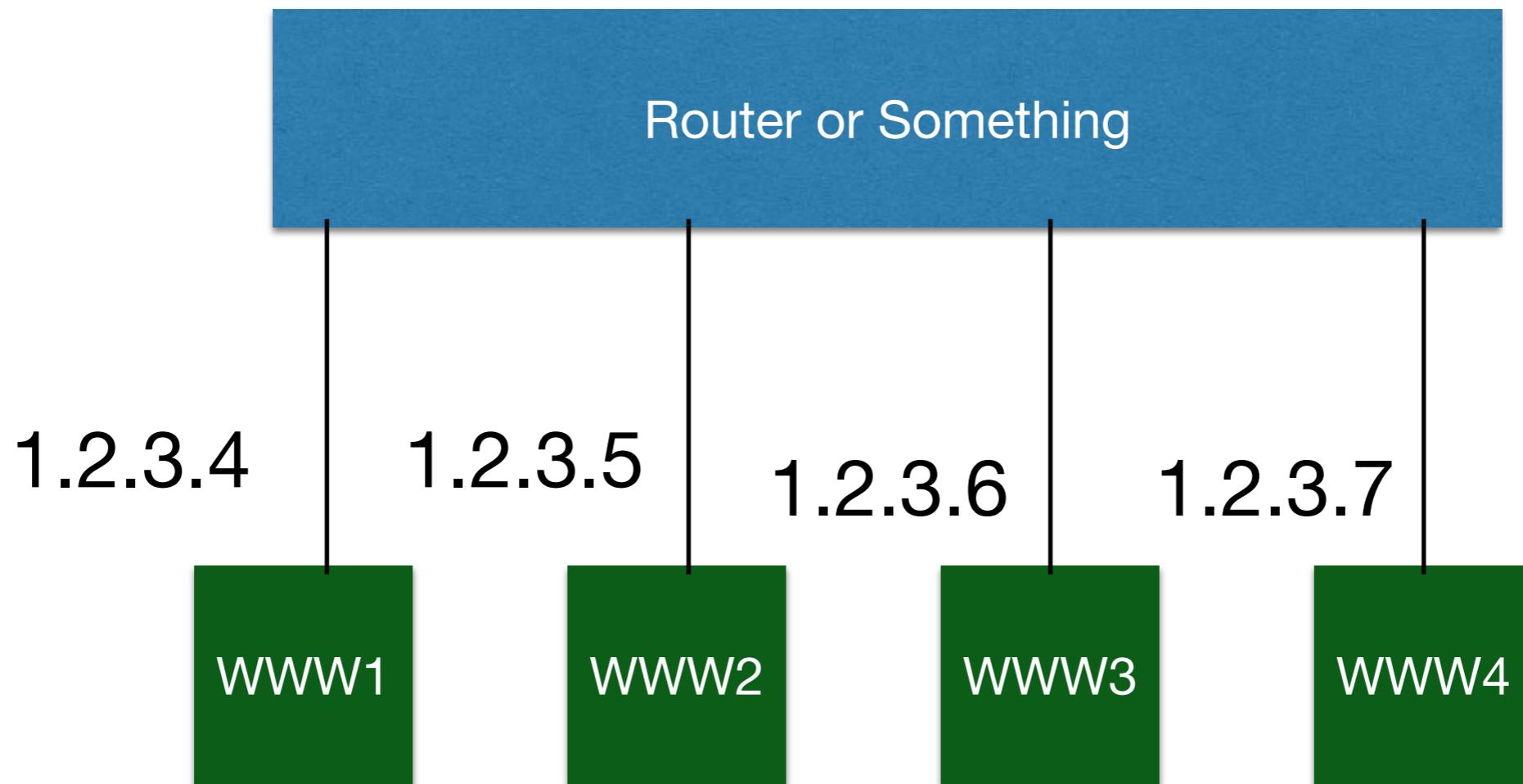
# VLAN security

- Quite weak
- Akin to writing “private” on an envelope but not sticking the flap down
- Splitting a switch with untagged ports might be OK, as might inter-switch links, but in general using VLANs for anything involving hard separation will fail assurance
  - Anyone clever enough to use VLANs for this is clever enough to not use VLANs for this
  - All security fails in face of attacker who can reconfigure switch, and most switches aren’t very secure

# Link-Agg + VLANs

- You can deliver two or more networks over two or more cables, with full load balancing and failover
- Aggregate using Link Aggregation
- Then apply VLAN tagging for the separate networks
- Not enough people do this: it is very, very effective

# Load Balancing



[www.domain](http://www.domain) -> four addresses

# Naive Approach

- Treat as one machine with four addresses
- Rely on DNS round-robin to deal with balancing
  - Very rough balance
  - Poor tolerance of failure if clients don't try other addresses, slow tolerance of failure anyway

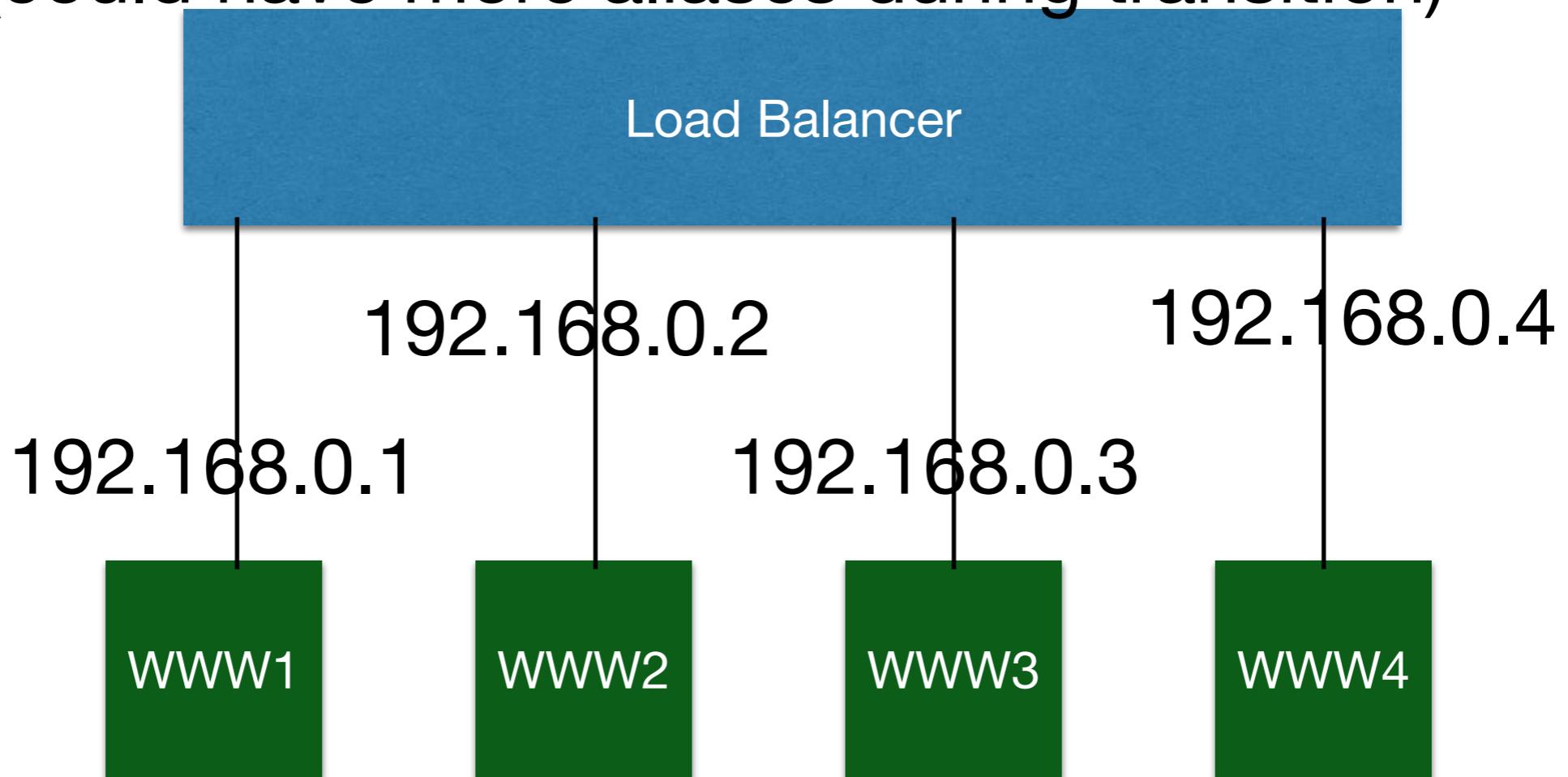
# Load Balancers

- Inbound NAT
- Constantly ping devices and/or count live connections
- Sends incoming connection to least loaded, active machine
  - Can also use weighted balances, magic protocols to report load, measures of bandwidth, etc.
- Service lives on **one** IP number, balanced over multiple servers

# Load Balance

1.2.3.4

(could have more aliases during transition)



www.dom.ain -> one address

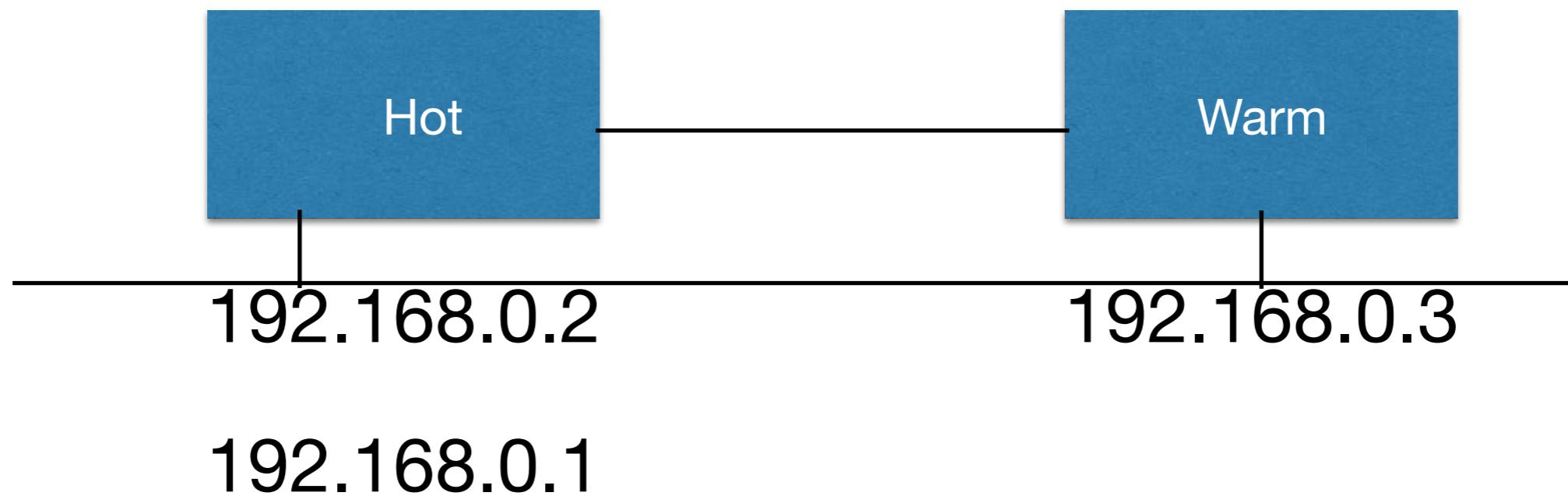
# Load Balancing HTTP and HTTPS

- Sometimes needs slightly more than Inbound NAT, as connections to servers need to be slightly “sticky” – successive TCP connections from same source need to go to same destination because of session cookies and so on.
- Load balancer stores source of old sessions, uses same server for subsequent requests from same source.
  - Also a problem for load-spread web caches, but those are much less common in 2017

# VRRP

- What about routers, firewalls, etc which need to see all traffic?
- Can't easily load balance, but can do failover

# VRRP



# VRRP

- Advertise shared IP number as real service address
  - Router for DHCP, IMAP service for IMAP, etc
- Mutual monitoring and takeover of shared address
- Requires multiple redundant networks between peers, as “split brain” situation potentially quite nasty

# So why Multipath TCP?

- Solves some of this
  - In 2017, presumption that new protocols will be TCP-based
  - Many traditional UDP protocols have TCP analogues (DNS, NFS, sadly not NTP) which work better anyway

# Multipath TCP

- Establish link from any interface to any other interface
- Hosts exchange information about other interfaces they have, which might be in the DNS, or might be a little more private
- Hosts can then attempt to make more connections
- Kernel marshals them as required
- Applications only see the original connection
- Will handle IPv4 and IPv6 on single connection
  - Still needs application to try both initially to deal with single-stacked case

# Summary

- Multi homing is complex
- Applications should deal with multiple interfaces, but often don't and “deal with” isn't a simple definition
- Load balancing and failover is complex but beneficial
- Multipath TCP solves some of the issues
- Use `getaddrinfo()` / `getnameinfo()` in all new code

# Networks 13: DNS

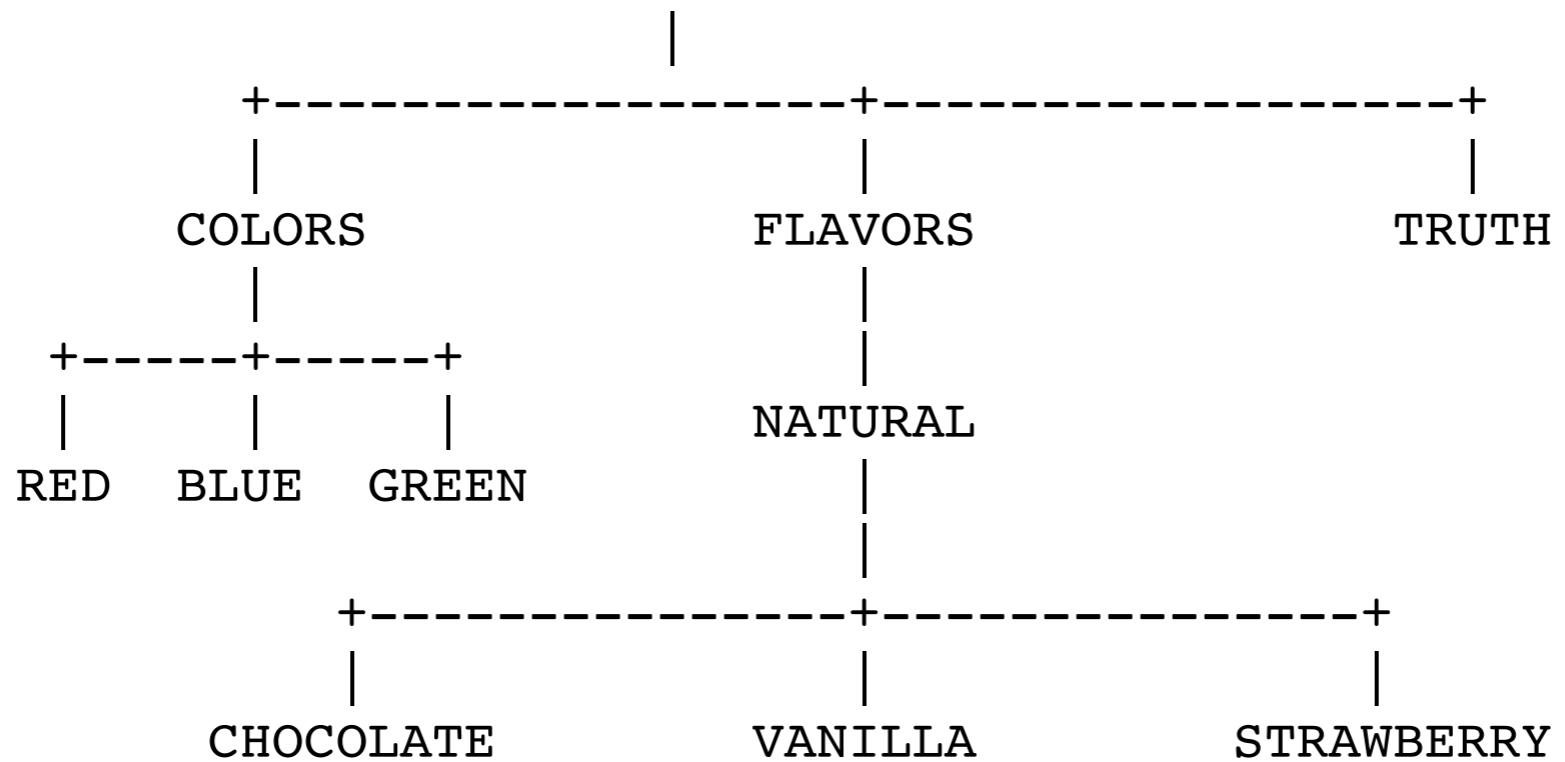
[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# What is DNS?

- Absolutely fundamental to Internet: maps names to IP numbers (v4 and v6), numbers to names, locates resources
- One of the oldest protocols in regular use (RFC 882, November 1983, current protocols pretty much as they now are in RFC1034/RFC1035, November 1987)
- Hideously insecure, complex implementations

# DNS Concept

From RFC882



# Outline

- Everyone that needs one can get a domain name and operate a nameserver at that point in the hierarchy.
- Once they have that domain name, they can create **resource records** within the domain, to an arbitrary depth
  - Not quite: DNS names are limited to 255 bytes, 63 bytes per label
- They can also **delegate** portions of the namespace to other nameservers
- A group of resource records served from one nameserver is called a **zone**

# DNS as Database

- The DNS forms a loosely coupled distributed database, containing key/value pairs
  - Sometimes grossly abused for that purpose, as we will see later
- Lots of caching and redundancy, dating back to a slower, less reliable, less connected Internet
- As a general rule, everyone's DNS infrastructure is broken, and the definition of “not broken” is the topic of much debate
  - I hope I'm treading a middle-of-the-road position

# Resource Records (RR Sets)

- Map a name to some data, plus have some book-keeping data in them.
- Simple case, A records contain IPv4 addresses
- AAAA records contain IPv6 addresses

Name	TTL	Class	Type	Data
gromit.cs.bham.ac.uk.	86400	IN	A	147.188.193.16
research-1.batten.eu.org.	86400	IN	AAAA	2001:630:c2:3263:8:20ff:fe89:b5a0

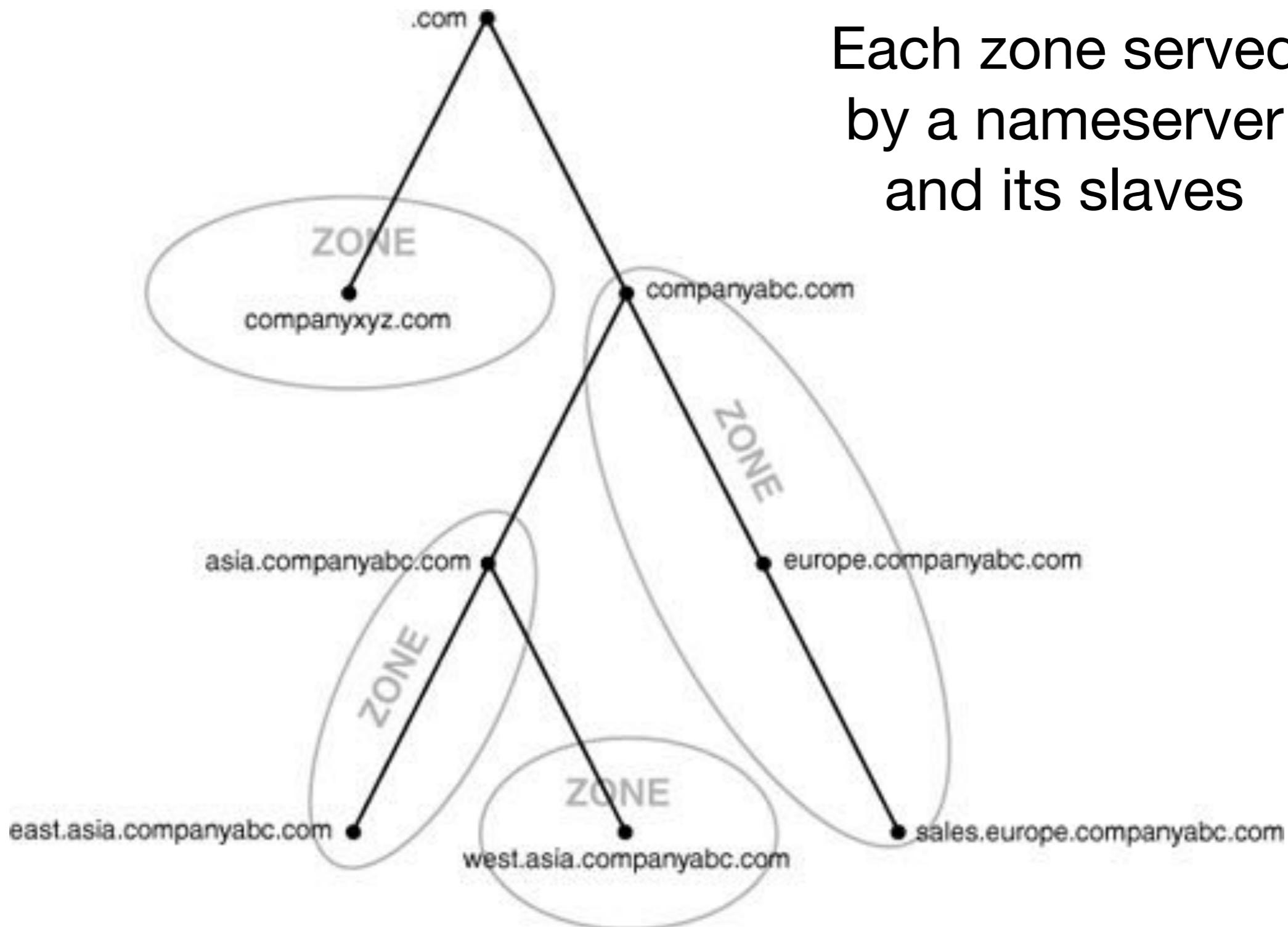
# RR Sets

- Class is always IN for the INternet (older classes no longer relevant, wise implementors reject them out of hand)
- Types include A, AAAA, PTR (address->name), MX (mail exchangers), NS (nameservers), CNAME (aliases), SOA (authority records) and TXT (random dumping ground for textual information)
- Can be multiple records for a given name, hence **RR sets**.

# TTL

- TTL: “Time to Live”, usually in seconds
  - You quickly learn that 3600 is an hour, 86400 is a day, 604800 is a week.
- You can cache an RR for that long

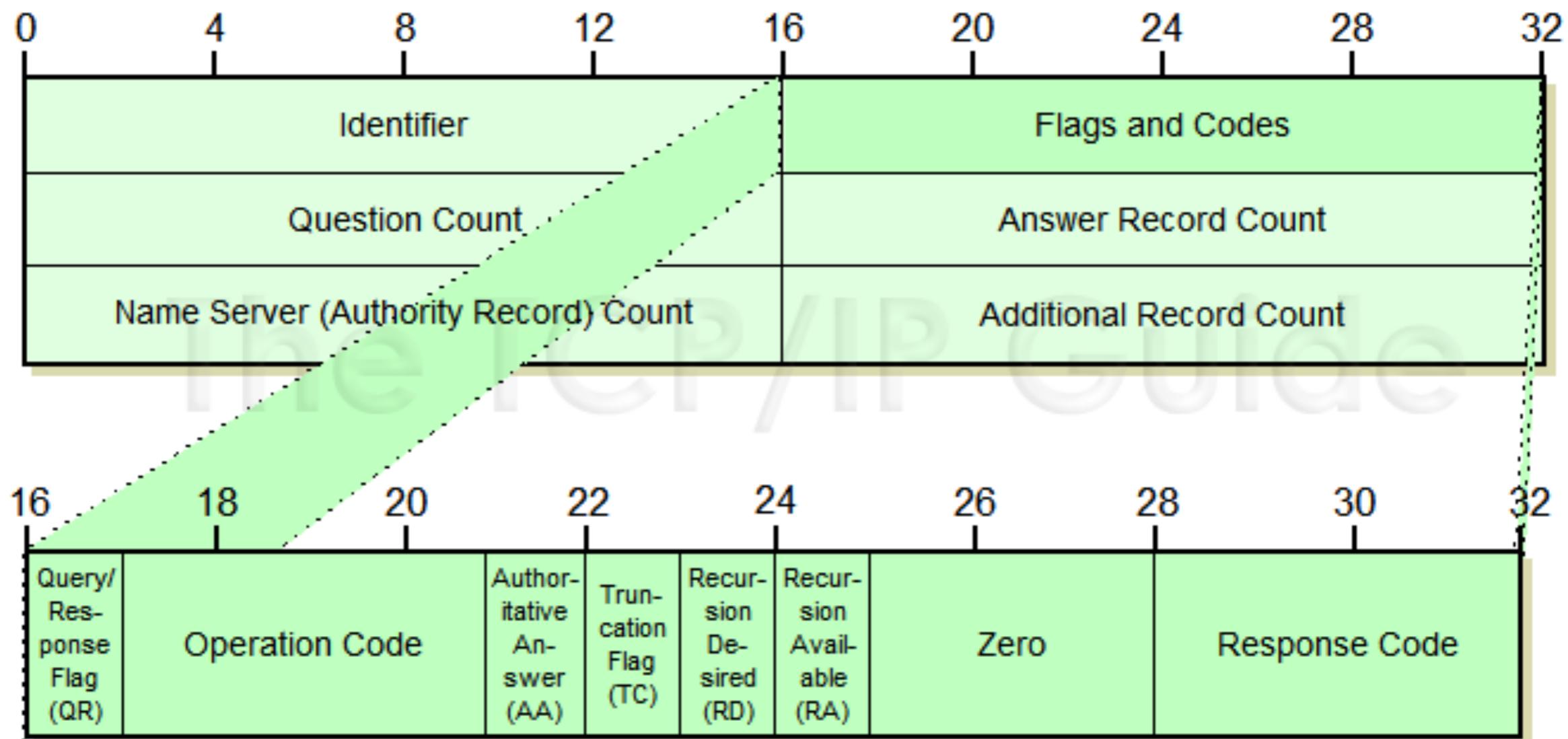
# Zones



# DNS Components

- **Clients:** ask questions to **recursive servers** and receive answers
- **Recursive Servers:** (sometimes called **caching servers**) can be asked a complete question (“what is the address of some.machine.dom.ain?”) and will give a complete, but sometimes **inauthoritative** answer
- **Authoritative Servers:** (sometimes called **iterative servers**) only give **authoritative** answers about zones they are configured to know about.

# DNS Packets



# Header

- ID: sixteen bit identifier for this question and answer. 16 bits massively too small: details later
- QR: 0 is a query, 1 is a response to a query
- Op:
  - 0 is a query (or an answer to a query if QR=1)
  - 1 is an IQUERY (obsolete since 2002)
  - 2 is a STATUS (rarely implemented)
  - 4 is NOTIFY (master informing slave that zone has changed)
  - 5 is UPDATE (dynamic DNS, where used)

# Flags

- AA: Authoritative Answer
  - 1 means either I am authoritative, or this answer has freshly come from an authoritative server. 0 means it's come from a cache.
- TC: Truncation
  - 1 means that there is more than 512 bytes of response, so the client should establish a TCP connection and fetch data that way.
  - If it's just “additional information” that has been truncated, may not bother. 512 bytes relates to old Arpanet MSS of 560 bytes.

# Flags

- RD, RA
  - Recursion Desired means “please answer this question in its entirety”.
  - Recursion Available means “I would do recursion if you asked me for it”, which is worth recording
  - Error Codes:
    - See RFCs

# Resource Records

- Name TTL Class Type Data
- foo.domain.mytld is represented as [3] foo [6] domain [5] mytld [0] where [3] is a byte with the value 3, 0x3, 00000011.
- Labels can be compressed

# Label Compression

- Labels are maximum 63 bytes, so largest value for a length field is 00111111.
- Length fields starting 11 (ie,  $\geq 192$ ) are special. If length field starts 11 (ie  $\text{length} \& \text{C0} == \text{C0}$ ) then next six bits plus the next byte, total fourteen bits, are special.
- 14 bits are read as “read from this offset in message to next zero”.

# Compression

- Suppose [3] foo [3] dom [3] ain [0] starts at byte 48 in the message.
- Suppose we want to represent the name mail.foo.dom.ain.
- Choice 1: [4] mail [3] foo [3] dom [3] ain [0], 18 bytes.
- Choice 2: [4] mail [192] [240], 7 bytes
- Particularly effective as queries about things in domain x.y.z tend to have a lot of x.y.z in them.
- Compression goes to end of name, you can't "reuse" labels like www and mail from other domains.

# Clients

- An utter shambles on most operating systems
- DNS clients are also known as “resolvers”
- You can make direct queries to the DNS using the facilities of libresolv (`res_mkquery`)
- Normally you call `getaddrinfo()` or `gethostbyname()` and that chooses mechanism from DNS, local files, LDAP, NIS/YP (is Ronald Reagan still president?) and so on
  - `/etc/nsswitch.conf` is common, originally Ultrix (I think), then Solaris, now Linux as well — maps queries (hosts, passwd, printers) to sources (DNS, files, NetInfo, whatever)
- Modern systems maintain a cache over all this (nscd on most Unixes).
  - nscd caches tend to be sloppy with TTL handling and cache everything for up to an hour.

# Recursive Servers

- Know how to answer a complete question (mechanism to follow)
- Should be access-control and firewall restricted to local network
- Once they have resolved a name they can cache the result, and can answer repeated questions from the cache so long as they appropriately decrement the TTL

# Caching In Action

```
ians-macbook-air:clocks igb$ dig aaaa rsync.batten.eu.org  
AA = Authoritative Answer  
; <>> DiG 9.8.3-P1 <>> aaaa rsync.batten.eu.org  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15509  
;; flags: qr aa ra; QUERY: 1, ANSWER: 1, AUTHORITY: 7, ADDITIONAL: 6  
  
;; QUESTION SECTION:  
;rsync.batten.eu.org. IN AAAA  
  
;; ANSWER SECTION:  
rsync.batten.eu.org. 86400 IN AAAA 2001:630:c2:3263:8:20ff:fee9:4d41  
ians-macbook-air:clocks igb$ dig aaaa rsync.batten.eu.org  
!AA  
; <>> DiG 9.8.3-P1 <>> aaaa rsync.batten.eu.org  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 232  
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0  
  
;; QUESTION SECTION:  
;rsync.batten.eu.org. IN AAAA  
  
;; ANSWER SECTION:  
rsync.batten.eu.org. 86245 IN AAAA 2001:630:c2:3263:8:20ff:fee9:4d41
```

# Full Initial Response

```
ians-macbook-air:clocks igb$ dig aaaa rsync.batten.eu.org
```

```
; <>> DiG 9.8.3-P1 <>> aaaa rsync.batten.eu.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15509
;; flags: qr aa ra; QUERY: 1, ANSWER: 1, AUTHORITY: 7, ADDITIONAL: 6

;; QUESTION SECTION:
;rsync.batten.eu.org. IN AAAA

;; ANSWER SECTION:
rsync.batten.eu.org. 86400 IN AAAA 2001:630:c2:3263:8:20ff:fee9:4d41
```

```
; AUTHORITY SECTION:
batten.eu.org. 86400 IN NS offsite6.batten.eu.org.
batten.eu.org. 86400 IN NS ns5.he.net.
batten.eu.org. 86400 IN NS ns3.he.net.
batten.eu.org. 86400 IN NS ns2.he.net.
batten.eu.org. 86400 IN NS mail.batten.eu.org.
batten.eu.org. 86400 IN NS ns4.he.net.
batten.eu.org. 86400 IN NS offsite7.batten.eu.org.
```

```
; ADDITIONAL SECTION:
offsite6.batten.eu.org. 86400 IN A 128.204.195.144
offsite6.batten.eu.org. 86400 IN AAAA 2a00:7b80:3019:12::579c:4928
mail.batten.eu.org. 300 IN A 147.188.192.250
mail.batten.eu.org. 300 IN AAAA 2001:630:c2:3263:8:20ff:fed7:92e1
offsite7.batten.eu.org. 86400 IN A 64.188.45.237
offsite7.batten.eu.org. 86400 IN AAAA 2607:f2e0:10f:14:4321:4321:5e6:9ee6
```

“Here’s where I got it from, these are the sources of truth”

These might be useful

# Cached Response

```
ians-macbook-air:clocks igb$ dig aaaa rsync.batten.eu.org

; <>> DiG 9.8.3-P1 <>> aaaa rsync.batten.eu.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 232
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;rsync.batten.eu.org.      IN AAAA

;; ANSWER SECTION:
rsync.batten.eu.org. 86245 IN AAAA 2001:630:c2:3263:8:20ff:fee9:4d41

;; Query time: 80 msec
;; SERVER: 147.188.129.250#53(147.188.129.250)
;; WHEN: Tue Feb 24 09:33:38 2015
;; MSG SIZE  rcvd: 65
```

# How are names resolved?

- Clients ask recursive servers (“rd” = “recursion desired”)
- Recursive servers start at the root, and work downwards asking for the nameserver of the next label down, until they can finally ask the last nameserver for the required RR

# Flow of packets

- Recursive nameserver is pre-configured with addresses of nameservers for “.”, the root of the domain space.
- To answer a query for “A foo.dom.ain”, asks one of the root nameservers “NS? foo.dom.ain”.
- Will get back either the location of the nameservers for “ain”, or “dom.ain”, or “foo.dom.ain”, depending on what knowledge the server has.
- Can then ask next server down the same question, until someone answers with an A record.
  - Called “iteration”, although actually feels somewhat recursive.

# Command Line Example

```
ians-macbook-air:~ igb$ dig +norecurse @a.root-servers.net. ns uk | awk '$4=="NS"' | head -1
uk.      172800 IN NS nsd.nic.uk.
ians-macbook-air:~ igb$ dig +norecurse @nsd.nic.uk. ns ac.uk | awk '$4=="NS"' | head -1
ac.uk.    172800 IN NS ns1.surfnet.nl.
ians-macbook-air:~ igb$ dig +norecurse @ns1.surfnet.nl. ns bham.ac.uk | awk '$4=="NS"' | head -1
bham.ac.uk. 86400 IN NS dns0.bham.ac.uk.
ians-macbook-air:~ igb$ dig +norecurse @dns0.bham.ac.uk. www.bham.ac.uk
```

You can experiment with this: sometimes the answer is returned as an answer, sometimes as authority records, depending the precise configuration of the server for the zone you are querying.

# Command Line Example

```
ians-macbook-air:~ igb$ dig +norecurse @dns0.bham.ac.uk. www.bham.ac.uk

; <>> DiG 9.8.3-P1 <>> +norecurse @dns0.bham.ac.uk. www.bham.ac.uk
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24883
;; flags: qr aa ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2, ADDITIONAL: 2

;; QUESTION SECTION:
;www.bham.ac.uk.           IN A

;; ANSWER SECTION:
www.bham.ac.uk.    172800 IN CNAME corp501.bham.ac.uk.
corp501.bham.ac.uk. 172800 IN A  147.188.125.39

;; AUTHORITY SECTION:
bham.ac.uk.        172800 IN NS dns1.bham.ac.uk.
bham.ac.uk.        172800 IN NS dns0.bham.ac.uk.

;; ADDITIONAL SECTION:
dns0.bham.ac.uk.   172800 IN A  147.188.128.2
dns1.bham.ac.uk.   172800 IN A  147.188.128.102
```

# Caching

- At each stage, everyone caches the data they get
- Particularly, the recursive server will rapidly build a cache of the NS records for popular domains, only refreshed once per TTL seconds
- In original protocol, anyone can supply cached information about anything and be believed
- Modern systems are more sceptical

# Authoritative Servers

- For a given domain, there will be one or more nameservers specified in some zone logically closer to the root (not necessarily one node up).
- Each of these specified nameservers is **authoritative** for the specified domain.
- The NS record that points to them is called a **delegation in the parent**.

# Authoritative Servers

- Why multiple servers?
  - Load balancing and redundancy
  - Ideally, spread over the world, spread over multiple ASes
  - Anycasting is useful: a domain with just one NS record may be using it
- They are all equally authoritative: no concept of “master” or “slave” is exposed
- They hold “zone files” for the zone files they are authoritative for
  - Either as real files, or as “zones” within the server software

# Authority Record

```
cs.bham.ac.uk. 86400 IN SOA dns0.cs.bham.ac.uk. hostmaster.cs.bham.ac.uk. (  
    2015022000 ; serial  
    10800      ; refresh (3 hours)  
    3600       ; retry (1 hour)  
    604800     ; expire (1 week)  
    86400      ; minimum (1 day)  
)
```

Serial, Refresh, Retry and Expire for benefit of slaves

Minimum now redefined for negative caching

# NS record

Should match in bham.ac.uk and cs.bham.ac.uk zones

cs.bham.ac.uk.	86400 IN NS dns1.cs.bham.ac.uk.
cs.bham.ac.uk.	86400 IN NS dns0.cs.bham.ac.uk.
cs.bham.ac.uk.	86400 IN NS dns0.bham.ac.uk.
cs.bham.ac.uk.	86400 IN NS ns0.susx.ac.uk.
cs.bham.ac.uk.	86400 IN NS <u>ext-proxy.ftel.co.uk.</u>

## Additional Information

dns0.cs.bham.ac.uk.	86400 IN A 147.188.192.4
dns1.cs.bham.ac.uk.	86400 IN A 147.188.192.8
ext-proxy.ftel.co.uk.	86400 INA 192.65.220.99
ext-proxy.ftel.co.uk.	86400 INA 192.65.220.98

Note: incomplete

# Old Attack

- Old nameserver software blindly accepted additional information and cached it
- Allows you to supply an IP number you control as “google.com 604800 IN A 1.2.3.4”; anyone who visits your nameserver has a chance of caching a bad nameserver for Google (or a bank, or whatever).
- Now stopped with “out of balliwick” controls: you only accept additional information that the server can reasonably be assumed to be authoritative for

# Master / Slave

- DNS protocol has support for replicating zones between master and slaves
  - Master does not have to be visible, “hidden master” is a popular pattern
  - Multiple nameservers can be updated by other means (SQL replication, rsync, people carrying USB sticks)

# Pro Tip

- A common pattern on small networks is for the authoritative server to also be the recursive / caching server for local clients
- **DO NOT DO THIS.**

# Delegation

- Suppose we have a nameserver for batten.eu.org. How do we create the domain home.batten.eu.org on another nameserver (or at least another zone file, possibly with different access rules)?

# Delegation: Just NS records

## In batten.eu.org

```
home.batten.eu.org. 86400 IN NS pi-two.home.batten.eu.org.  
home.batten.eu.org. 86400 IN NS dns-2.batten.eu.org.  
home.batten.eu.org. 86400 IN NS pi-one.home.batten.eu.org.
```

### ; ; ADDITIONAL SECTION:

```
dns-2.batten.eu.org. 86400 IN A 64.188.45.237  
dns-2.batten.eu.org. 86400 IN AAAA 2607:f2e0:10f:14:4321:4321:5e6:9ee6  
pi-one.home.batten.eu.org. 86400 IN A 10.92.213.231  
pi-one.home.batten.eu.org. 86400 IN AAAA 2001:8b0:129f:a90f:3141:5926:5359:1  
pi-two.home.batten.eu.org. 86400 IN A 10.92.213.238  
pi-two.home.batten.eu.org. 86400 IN AAAA 2001:8b0:129f:a90f:3141:5926:5359:2
```

# Glue Records

- How do you locate the A record for “dom.ain 86400 IN NS ns1.dom.ain”?
- The zonefile dom.ain in that case has an A record for ns1.dom.ain as well as an NS record for dom.ain.
- This is called a **Glue Record**

# Mail Exchangers

**;; ANSWER SECTION:**

batten.eu.org. 86400 IN MX **4** bham-mx2.bham.ac.uk.  
batten.eu.org. 86400 IN MX **4** bham-mx3.bham.ac.uk.  
batten.eu.org. 86400 IN MX **2** mail.batten.eu.org.  
batten.eu.org. 86400 IN MX **4** bham-mx1.bham.ac.uk.

**;; ADDITIONAL SECTION:**

bham-mx2.bham.ac.uk. 86350 IN A 147.188.128.219  
bham-mx3.bham.ac.uk. 86350 IN A 147.188.128.221  
mail.batten.eu.org. 300 IN A 147.188.192.250  
mail.batten.eu.org. 300 IN AAAA 2001:630:c2:3263:8:20ff:fed7:92ef  
bham-mx1.bham.ac.uk. 86350 IN A 147.188.128.129

# Zone File Maintenance

- You can edit zone-files manually, but it is very prone to error
- Most sites generate the zone files from some other source of information, usually a database or some XML (classic “greybeard” shell scripts, which scare everyone once the author leaves)
- Also dynamic DNS

# Dynamic Update

```
ians-macbook-air:~ igb$ nsupdate -k update-key
> server offsite7.batten.eu.org
> update add some-spurious-rrset.batten.eu.org 86400 in a 1.2.3.4
>
> ians-macbook-air:~ igb$ dig @offsite7.batten.eu.org some-spurious-rrset.batten.eu.org

; <>> DiG 9.8.3-P1 <>> @offsite7.batten.eu.org some-spurious-rrset.batten.eu.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 64938
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 7, ADDITIONAL: 13

;; QUESTION SECTION:
;some-spurious-rrset.batten.eu.org. IN A

;; ANSWER SECTION:
some-spurious-rrset.batten.eu.org. 86400 IN A 1.2.3.4
```

# Works worldwide!

```
ians-macbook-air:~ igb$ dig @8.8.8.8 some-spurious-rrset.batten.eu.org.  
;  
; <>> DiG 9.8.3-P1 <>> @8.8.8.8 some-spurious-rrset.batten.eu.org.  
; (1 server found)  
;; global options: +cmd  
;; Got answer:  
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 15394  
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0  
  
;; QUESTION SECTION:  
;some-spurious-rrset.batten.eu.org. IN A  
  
;; ANSWER SECTION:  
some-spurious-rrset.batten.eu.org. 21599 IN A 1.2.3.4
```

# Dynamic Update

- Useful for DHCP servers in trusting environments
  - You can update the DNS to reflect equipment entering and leaving the network
  - Allows anyone on the network to pretend to be anyone else by careful use of DHCP client ids

# Reverse Mapping

- IP number is transformed into reverse order and looked up in special in-addr.arpa or .ip6.arpa domain:

250.192.188.147.in-addr.arpa. 86400 IN PTR offsite.batten.eu.org.

1.0.0.0.9.5.3.5.6.2.9.5.1.4.1.3.e.0.9.a.f.9.2.1.0.b.8.0.1.0.0.2.ip6.arpa. 86400 IN PTR pi-one.batten.eu.org.

# Why the strange formats?

- in-addr.arpa format allows delegation on 8-bit boundaries.
- Makes delegation of /28s (say) difficult
- Various messy solutions: look them up.
- Lesson learnt, so ipv6 reverse mapping allows delegation on 4-bit boundaries in hierarchy.

# Reverse Attacks

- Suppose I control 2.3.4.in-addr.arpa
- I can create RR “1.2.3.4.in-addr.arpa PTR something.bham.ac.uk” and try to pose as part of Birmingham network for purposes of libraries, Apple discounts, etc.
- Check is to look up something.bham.ac.uk and see if it matches: only bham.ac.uk admin can install required “something.bham.ac.uk A 1.2.3.4” record

# DNS Security

- DNS Sec exists to sign zones, providing evidence that packets haven't been tampered with
- Topic for network security lectures to come
  - Complex
  - Didn't scale without major modifications
  - Very low adoption after ~20 years
  - Doesn't solve common use-cases

# Networks 16: Other Applications

[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# Applications

- TCP and below are complex, so that it's easy to write applications
- IETF slogan “rough consensus and running code” means that applications tend to be proposed, implemented rapidly and then developed incrementally: grand architectures don't achieve consensus.
- “Simple” is the watchword.

# User Protocols

- FTP (RFC959) File Transfer
  - goes back to RFC114 April 1971, so very obviously not TCP-based
- SMTP (RFC5321) Simple Mail Transfer
  - goes back to RFC821 August 1982 – prior to that FTP was used for the purpose
- HTTP (RFC7230–7235) HyperText Transport
  - goes back to RFC2616 June 1999

# User Protocols

- POP3 (RFC1939) Post Office
  - Simple, easy to implement
- IMAP4rev1 (RFC2060) Internet Message Access
  - Complex, Lisp-Like syntax, complete implementations are surprisingly rare

# User Protocols

- ssh (RFC 4251–4253) Secure Shell
  - Replaces telnet, rlogin, rsh, other things, all of which are very insecure and should never be used
  - Permits remote login (pseudo-tty), remote command execution, remote copying
  - Encrypted, various secure authentication mechanisms

# File Sharing

- NFS (originally from Sun)
  - v2 RFC1094, v3 RFC1813, v4 RFC5661
  - Implementing from earlier standard difficult, as lots of Unix-semantics assumptions
- SMB aka CIFS (originally from Microsoft)
  - **Not standardised**, lots of reverse engineering followed by proprietary documentation which may or may not be accurate.

# Infrastructure

- DNS (*lectures passim*)
- SNMP Simple Network Management
  - Not Simple At All
- (S)NTP (Simple) Time

# Conventions...

- Commands are case insensitive
  - “Four character” convention on older protocols (on mainframes, comparison of four-character strings is particularly efficient)
- Lines are terminated with \r\n, \015\012, control-m control-j, carriage return line feed, because Elvis is still alive.
- Responses often consist of a three-digit code followed by explanatory text
- SMTP is the best exemplar

# ...there to be broken

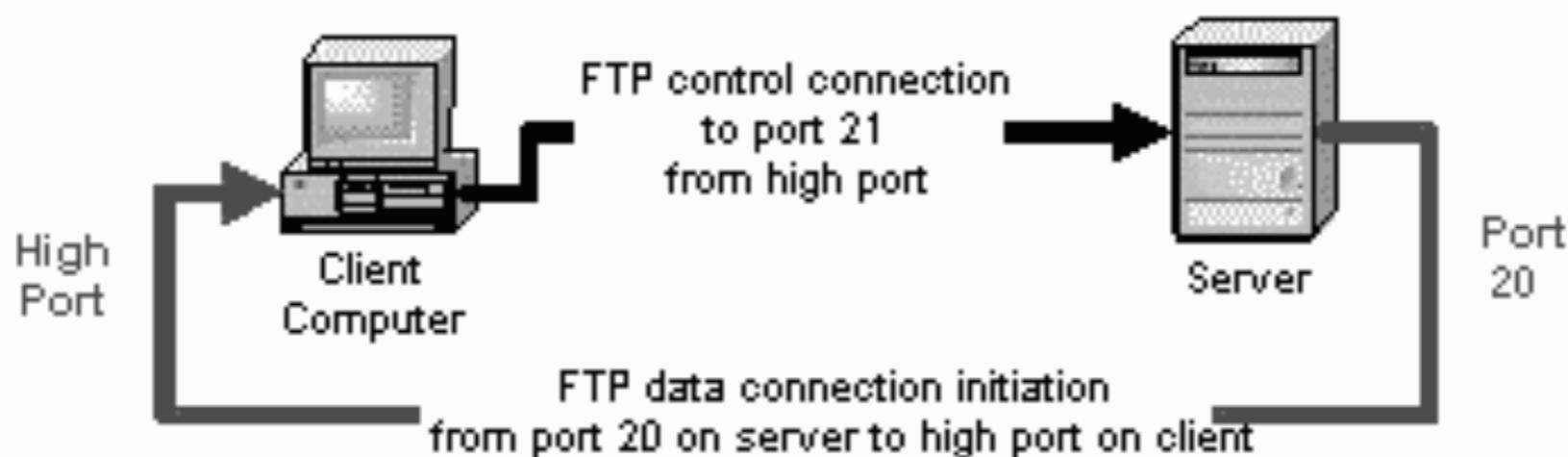
- POP3 returns result codes in the first character of the response
- IMAP4 uses “tags” to match responses to requests
- And so on

# FTP

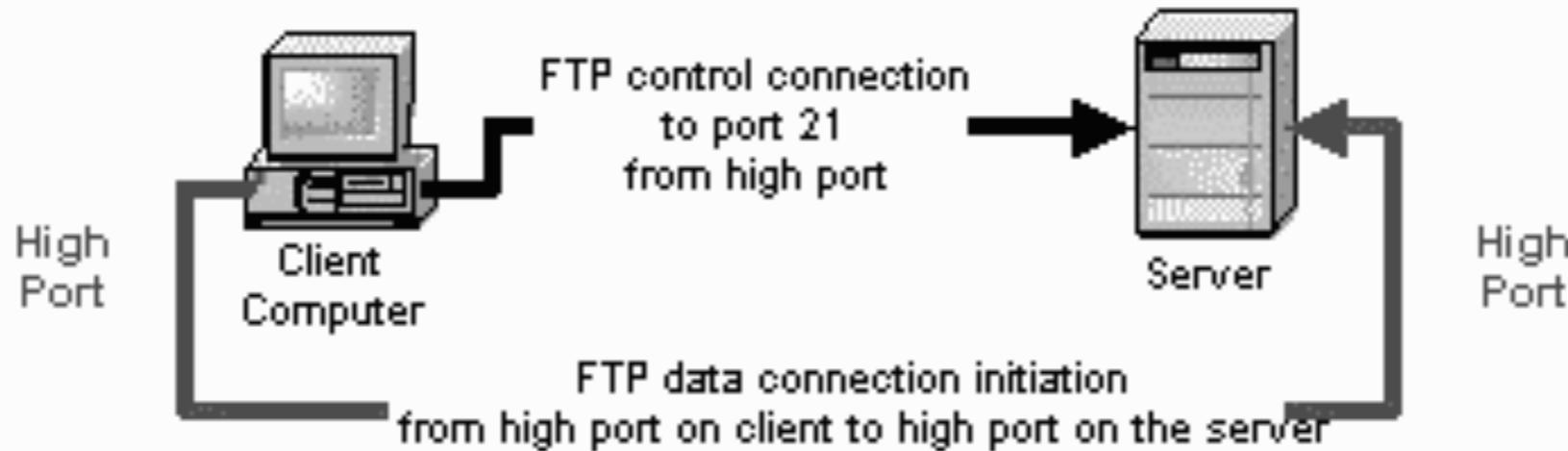
- Very old protocol (first RFC nearly 45 years ago)
- Requires extensive support to pass through NAT, Firewalls, etc
- Large and complex server (difficult to audit and secure) supporting lots of conversion modes for TOPS-20, VMS, VM/CMS, block, record, stream...

# FTP

## Active FTP



## Passive FTP



# Mechanism

- Control connection goes to port 21 on server
- When transferring files, CLIENT listen()s on a high numbered port, tells the server where it is, and the server calls back with a source port of 20
- This is all NCP-style, from the 1970s

# New Mechanism

- PASV (Passive)
- Now the server listen()s on a high port, and the client calls to it from a high port
- Still requires firewall / NAT support, because high port to high port
- Firewall /NAT support involves snooping on the control connection

# FTP: Avoid

- Doesn't do anything that you both (a) need in 2017 and (b) you can't do with something more modern (HTTP or scp)
- Firewall and NAT support for FTP is complex and arguably insecure: certainly open to abuse
- Probably not as true now, but a few years ago FTP NAT support was >50% of total LoC in a NAT implementation
  - Prediction: it'll be used in some attack
- That there is an IPv6 profile for FTP is beyond all reason

# FTP Daemons

- System-shipped FTP daemons are usually frighteningly insecure
- ProFTPD is full featured but very complex to configure
- vsftpd is better and built for security
- As a minimum: drop root and chroot()

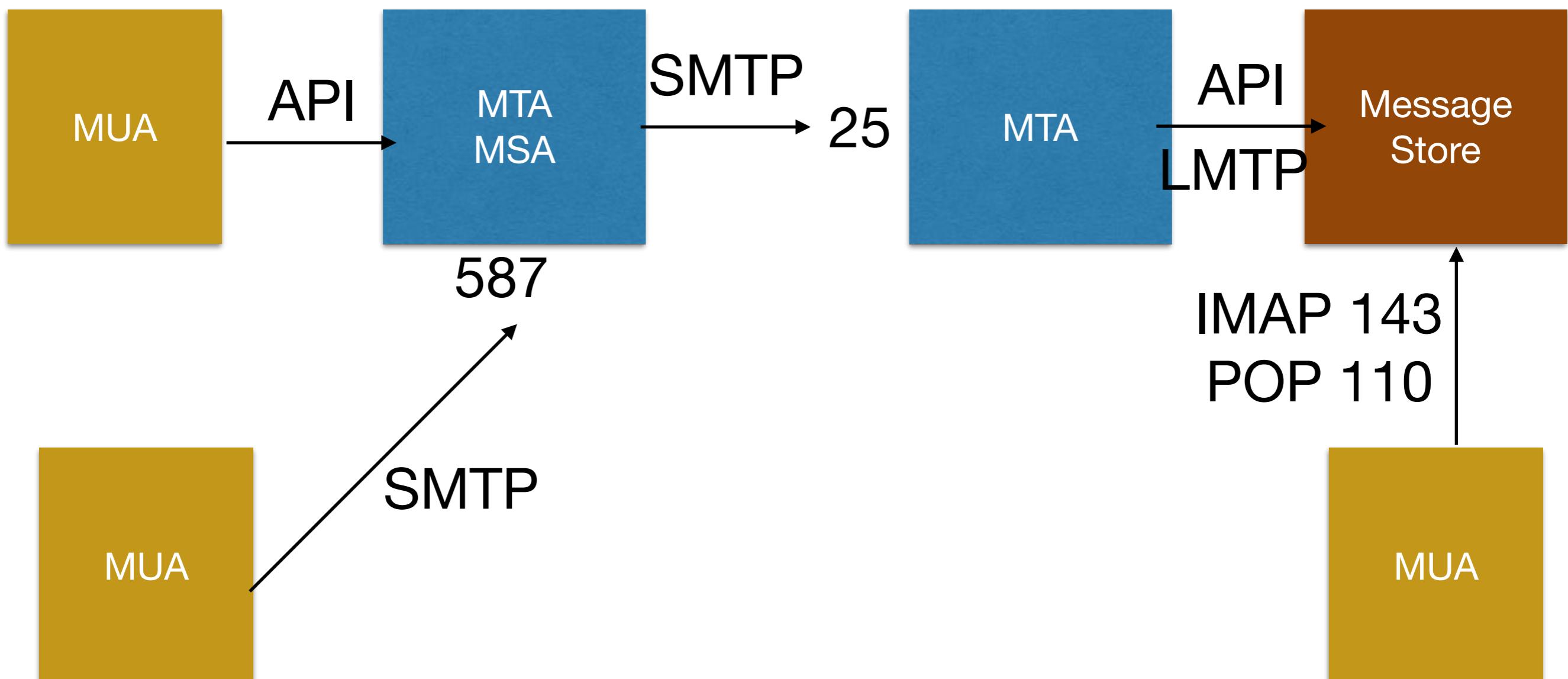
# SMTP

- Handles sending of mail
- Has extensive extensions for authentication, encryption, etc
- Mail User Agents talk to Mail Transport Agents and Mail Submission Agents. MTAs talk to MTAs.

# SMTP

Sendmail  
Exim  
Postfix

Cyrus  
Courier  
Dovecot



# SMTP

```
>>> EHLO research-1.batten.eu.org
250-mail.batten.eu.org Hello [IPv6:2001:630:c2:3263:8:20ff:fe89:b5a0], pleased to meet
250-ENHANCEDSTATUSCODES
(etc)
250 HELP
>>> MAIL From:<igb@research-1.batten.eu.org> SIZE=6
250 2.1.0 <igb@research-1.batten.eu.org>... Sender ok
>>> RCPT To:<igb@batten.eu.org>
>>> DATA
250 2.1.5 <igb@batten.eu.org>... Recipient ok
354 Enter mail, end with "." on a line by itself
>>> .
250 2.0.0 t257tb9n015843 Message accepted for delivery
igb@batten.eu.org... Sent (t257tb9n015843 Message accepted for delivery)
Closing connection to mail.batten.eu.org.
>>> QUIT
221 2.0.0 mail.batten.eu.org closing connection
```

# SMTP

- HELO [sic]
- MAIL FROM
- RCPT TO
- DATA
- . (dot on a line on its own)
- QUIT

# POP3

- Crude protocol for downloading email
- Connect, count messages, download
  - Assumes user has exactly one device to which they will be downloading the mail for further processing
- Can be grossly abused to provide sharing between devices, but this will end in tears. Always prefer...

# IMAP

- Complex and full-featured protocol for access to mailboxes
- Can be used as POP for pure download if you want, but it is much better to leave your mail on the server for access from everywhere

# IMAP

```
cmd1 login igb xxxpasswordxxx
cmd1 OK [CAPABILITY IMAP4rev1 LITERAL+ ID ENABLE ACL RIGHTS=kxte QUOTA MAILBOX-REFERRALS NAMESPACE UIDPLUS NO_ATOMIC_RENAME UNSELECT CHILDREN MULTIAPPEND BINARY CATENATE CONDSTORE ESEARCH SORT SORT=MODSEQ SORT=DISPLAY THREAD=ORDEREDSUBJECT THREAD=REFERENCES ANNOTATEMORE LIST-EXTENDED WITHIN QRESYNC SCAN XLIST URLAUTH URLAUTH=BINARY LOGINDISABLED AUTH=SCRAM-SHA-1 AUTH=DIGEST-MD5 AUTH=CRAM-MD5 AUTH=LOGIN AUTH=PLAIN COMPRESS=DEFLATE IDLE] User logged in SESSIONID=<mail.batten.eu.org-15852-1425542394-1>
cmd2 examine INBOX
* 3236 EXISTS
* 0 RECENT
* FLAGS (\Answered \Flagged \Draft \Deleted \Seen Junk NonJunk $MDNSent $NotJunk $Junk JunkRecorded $Forwarded NotJunk Forwarded Old $MailFlagBit0 $MailFlagBit1 $MailFlagBit2 Redirected)
* OK [PERMANENTFLAGS ()] Ok
* OK [UNSEEN 2796] Ok
* OK [UIDVALIDITY 1371846329] Ok
* OK [UIDNEXT 118743] Ok
* OK [HIGHESTMODSEQ 42689] Ok
* OK [URLMECH INTERNAL] Ok
cmd2 OK [READ-ONLY] Completed
cmd3 fetch 2796 full
* 2796 FETCH (FLAGS ($NotJunk NotJunk) INTERNALDATE "21-Feb-2015 06:37:10 +0000" RFC822.SIZE 87547 ENVELOPE ("Sat, 21 Feb 2015 06:35:37 +0000" "@bbcquestiontime tweeted: Our most retweeted comment came from @NicolaSturgeon : catch up on @BBCiPlayer #bbcqt" (("Popular in your network" NIL "info" "twitter.com")) ((("Popular in your network" NIL "info" "twitter.com")) ((("Popular in your network" NIL "info" "twitter.com")) ((("Popular in your network" NIL "info" "twitter.com")) ((("Batten TV Feed" NIL "tv" "batten.eu.org")) NIL NIL NIL "<E9.F1.35735.93728E45@twitter.com>") BODY ((("TEXT" "PLAIN" ("CHARSET" "UTF-8")) NIL NIL "QUOTED-PRINTABLE" 1379 34) ("TEXT" "HTML" ("CHARSET" "UTF-8")) NIL NIL "QUOTED-PRINTABLE" 82934 1676) "ALTERNATIVE"))
cmd3 OK Completed (0.000 sec)
cmd4 fetch 2796 body[header]
* 2796 FETCH (BODY[HEADER] {2909}
Return-Path: <b0487a57e25tv=batten.eu.org@bounce.twitter.com>
Received: from mail.batten.eu.org ([unix socket])
    by mail.batten.eu.org with LMTPA;
    Sat, 21 Feb 2015 06:37:10 +0000
X-Sieve: CMU Sieve 2.4
...
cmd4 OK Completed (0.000 sec)
```

# IMAP

- Supports remote searching, which has proven to be very useful for phones although was originally intended for a very different model
  - Use “elm” while logged into someone else’s VAX
  - Author Marc Crispin is a big TOPS-20 man, so IMAP does not have quite the same Unix-centric feel of other protocols
- Also supports download messages by parts, which is useful on slow links

# ssh

- Very complex commercial and licensing history, but now shipped on every \*nix.
- Windows versions (PuTTY best known) widely available.
- Encrypted transport layer more suited to small messages (ie, character-by-character typing) than TLS
- Various authentication mechanisms (password, public key, certificates)
- Well audited, so far robust against attacks

# NFS

- Ian's favourite!
- Developed by Sun for access to filesystems over ethernet
  - Diskless workstations
  - Dataless workstations

# NFS / ONC

- Part of suite called “ONC”, Open Network Computing.
- Included NIS, aka YP, for access to password files, host databases where DNS not available, “all the stuff usually in /etc”.
- Built on RPC (remote procedure call, Nelson and Birrell) and XDR (external data representation)

# RPC / XDR

- Mechanism to perform procedure calls remotely
- Arguments are encoded with XDR to deal with cross-platform byte order, float precision, padding, alignment, etc.
- Servers written as daemons and register with the portmapper
- Clients contact the port mapper (port 111) to find port for version x of service y, then make procedure calls
- Given technology is 30+ years old, works surprisingly well.

# NFS

- Standard Unix filesystem semantics, can be bodged to support other things
  - `close()` can error on disk-full conditions, still causing pain thirty years on
- Originally done over UDP for performance reasons, although RPC always supported TCP.
- Now almost always done over TCP

# NFS

- v2: create, statfs, getattr, link, lookup, mkdir, null, read, readdir, readlink, rename, remove, rmdir, root (unused), setattr, symlink, wrcache (unused), write
- v3 adds: access, mknod, readdirplus, fsinfo, pathconf, commit

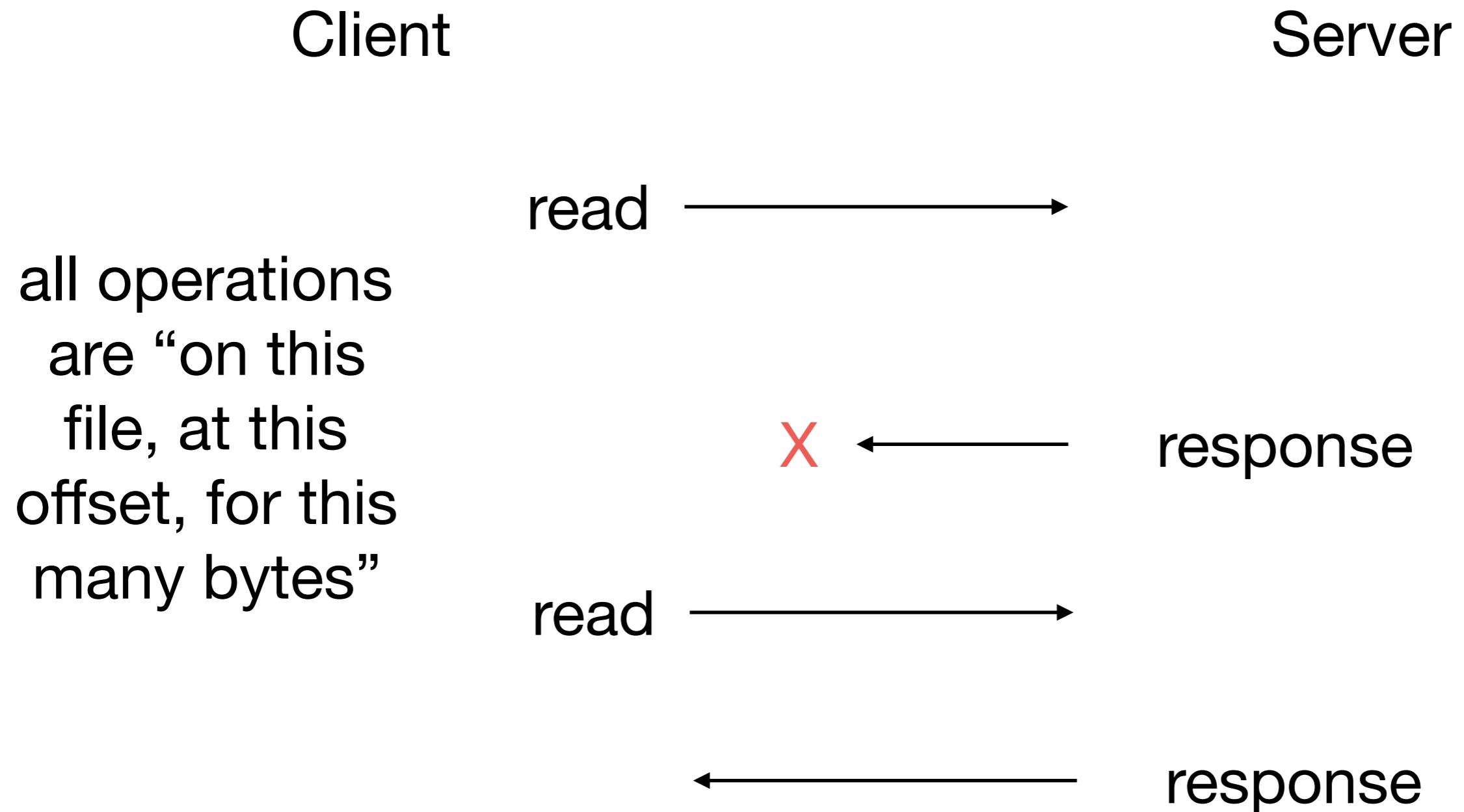
# NFS Statelessness

- Plan is to survive server crash and restart without needed (much) special handling on the client, and to avoid needing book-keeping on the server to track clients
  - Server notes when a client mounts a filesystem, but it's not used for anything serious.
- Because initially done over UDP, responses to operations that have in fact been completed can be lost silently.
- TCP doesn't fix this for persistent connections, as TCP doesn't have synchronisation points
  - And roll-back of filesystem operations very difficult and invasive
- Idea is that most operations can safely be repeated and get the same result (repeated reads, repeated writes)
- Causes various complexities in implementation, because some operations are inherently stateful (although they are rarer and less performance critical)

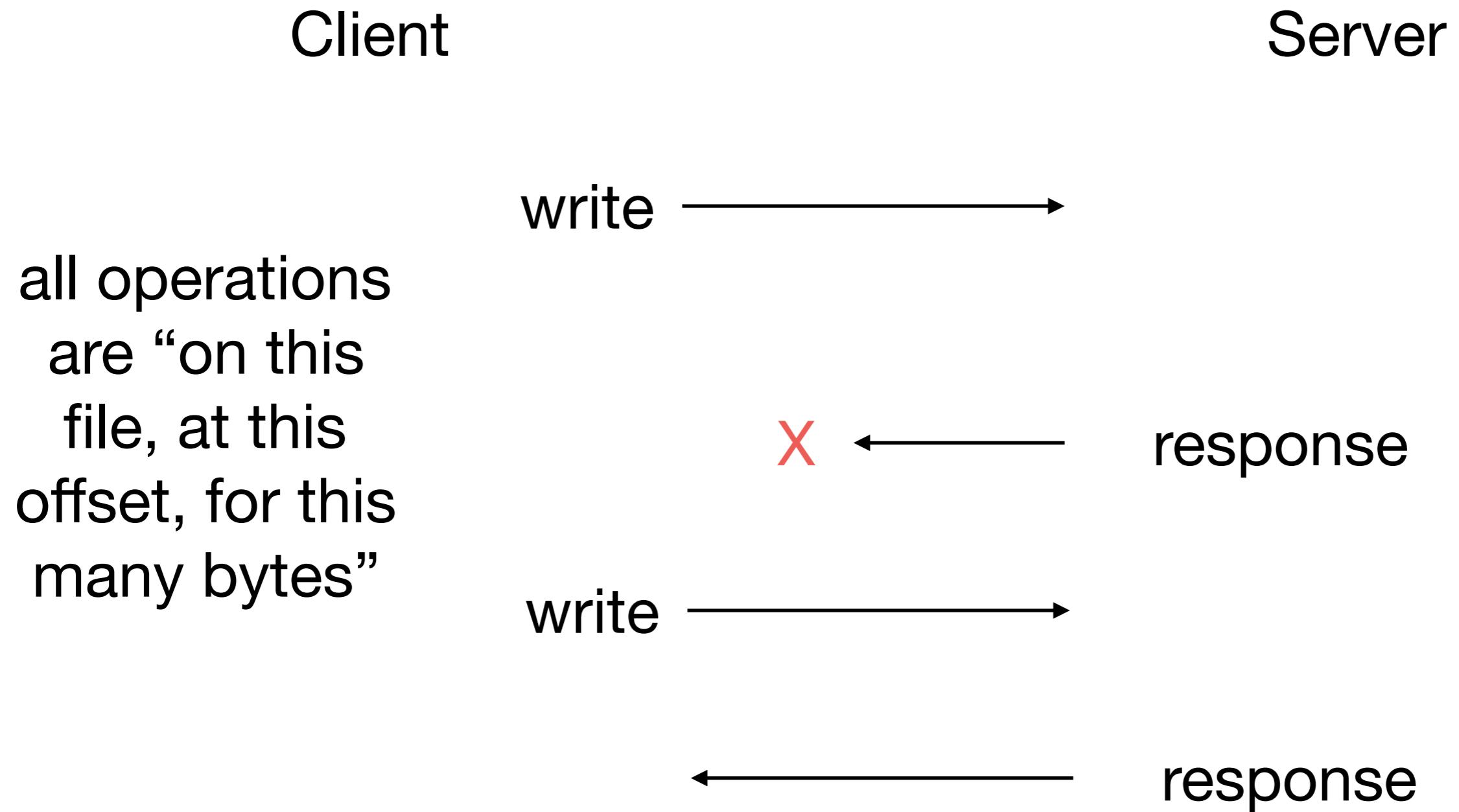
# Mount

- Mount involves sending requested pathname to server, returns “file handle” which encodes device number and other bookkeeping information
- No state on server: the filehandle is all you need, so you don’t need to remount after server crash and restart, you just continue using filehandles

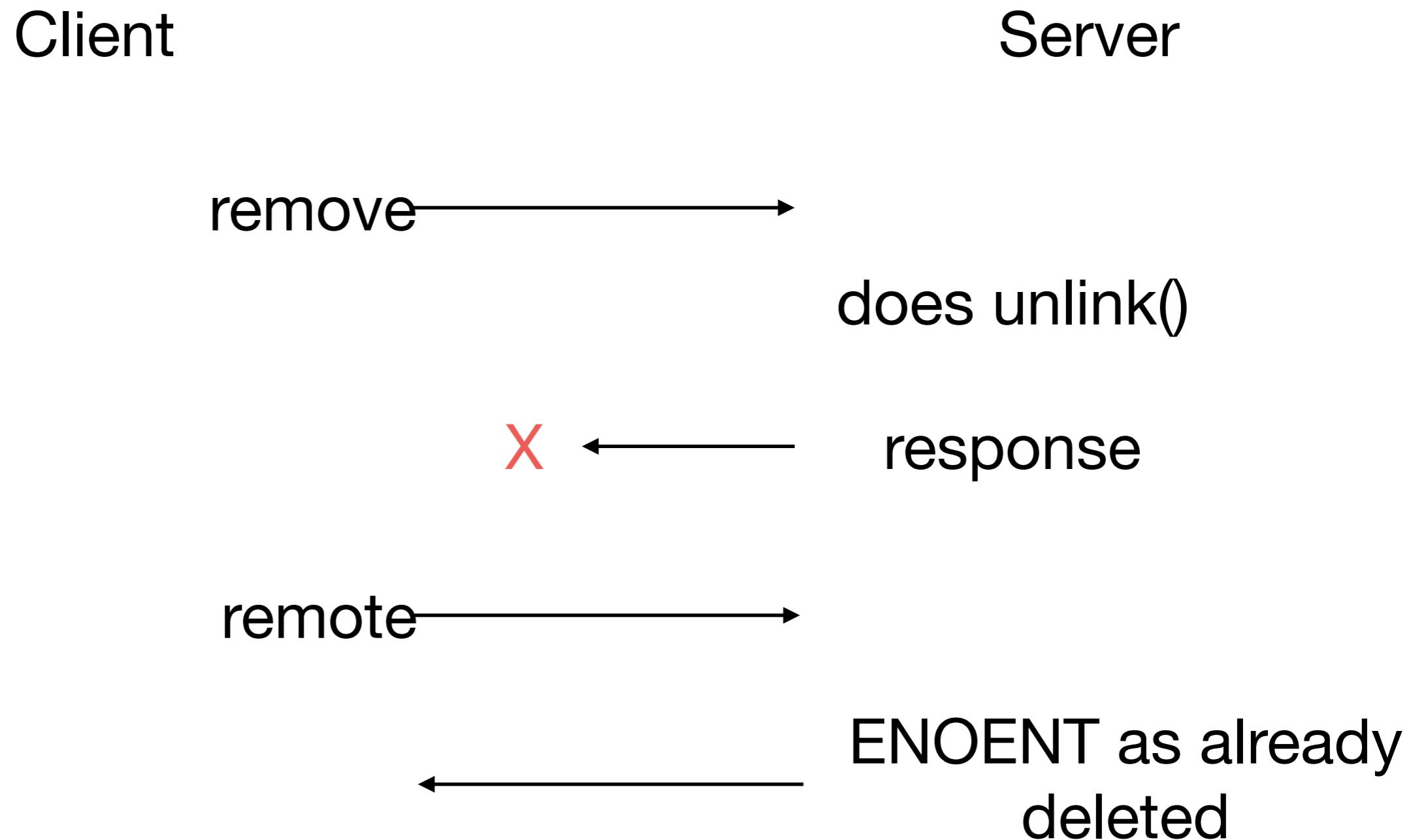
# Repeated Read OK



# Repeated Write OK



# Repeated Remove Not OK



# Why does TCP not fix this?

- You only find out that a TCP segment wasn't delivered to the other end when you call `close()` or `shutdown()` and get an error
- Some of the more complex OSI transport layers included “synchronisation points” to address this issue, where you can write a marker and get confirmation marker was received
  - Requires tricky API extensions to do this without blocking

# Idempotency Cache

- Server caches the responses it previously sent to stateful operations (also mknod, create, etc)
- If it receives a repeat request, it sends response from cache
- Fun ensues in the sequence:
  - remove() (etc)
  - server does unlink() and then crashes
  - server restart with empty idempotency cache
  - Client resends remove()

# NFS performance

- `write()` insists that data has gone to stable storage
- Big market in “Prestoserve” NVRAM caches
  - Auspex and others made a business out of it
- NFSv3 has “commit” so you can send data which you don’t demand is written, and then commit as appropriate
  - Big NVRAM caches were big business, Pillar, ZFS Appliances, NetApp, etc all survived until recently (NetApp still going, Pillar sort-of still going)
  - SSD means need for specialist fileservers somewhat diminished

# NFS Performance

- Good NFS servers faster than local disk (<3ms write common, difficult to match with rotating disk)
  - Writing to/reading from exotic 10-way stripes with large NV/RAM caches in front of them via GigE
- Good NFS servers easier to manage and provide good facilities for backup, thin provisioning, etc
  - Like having an EMC without needing expensive Fibre Channel infrastructure, and giving sharing as well
- SSD changes the equations, jury still out on best approaches, but hard to see how network filesystems can compete with latency of SSD

# NFS Locking

- Locking (of files or regions of files) is inherently stateful
- NFS cheats (no!) by claiming locking isn't part of the file serving protocol and bolting a separate locking protocol on the side
- Client crashes now a problem: lock file, crash, what happens?
- Often subtly broken outside homogenous environments, and always has been. Auspex cheated by using Sun code on embedded Solaris processor, and when they moved to their own code it broke. Linux, NetApp and Pillar have their own implementations which have varying problems

# SMB

- Similar to NFS, but stateful (clients have to remount after server crash, server knows who has what mounted and what open).
- Because stateful, incorporates locking model and better caching model.
- SMB harder to do very quickly, but (it pains me to say) probably easier to use correctly.

# SMB v CIFS

- SMB: implementations other than Microsoft's own can be temperamental, but most of them have had the bugs worked out by now
- Driving NFS out of the marketplace because there's a lot more Windows than there is Unix: NFS not shipped as standard on any Windows build.
- Almost all servers now “bi-lingual”: NetApp, Oracle ex-Pillar Axioms, Oracle ex-Sun ZFS Appliances, etc.
- Apple moving to SMB as well (previously supported NFS and AFS)

# SNMP

- Simple Network Management Protocol
- Response to various OSI initiatives, notably CMIP
  - Common Management Information Protocol: frighteningly complex, running over an OSI session layer that was completely impractical on embedded devices
  - Even CMOT (“CMIP over TCP”) is impractical
- The *Simple* is relative to CMIP: it’s actually rather complex.
- Much more formality than the typical Internet Protocol

# SNMP

- Intention was to use to manage switches, routers and other hardware, including configuration
- In reality, 90% of usage is monitoring, 9% is fault reporting, 1% (if that) is configuration.
- *Agents* sit on equipment and answer requests, *managers* or *clients* make the requests.
- Agents can also raise asynchronous alerts (“traps”) when they act as *trap sources* and send them to *trap sinks*.

# SNMP Concepts

- Data is grouped into a MIB, Management Information Base.
- The contents are described using a subset of ASN1, the Abstract Syntax Notation from OSI.
- Data is encoded on the wire using BER, Basic Encoding Rules, again from OSI.

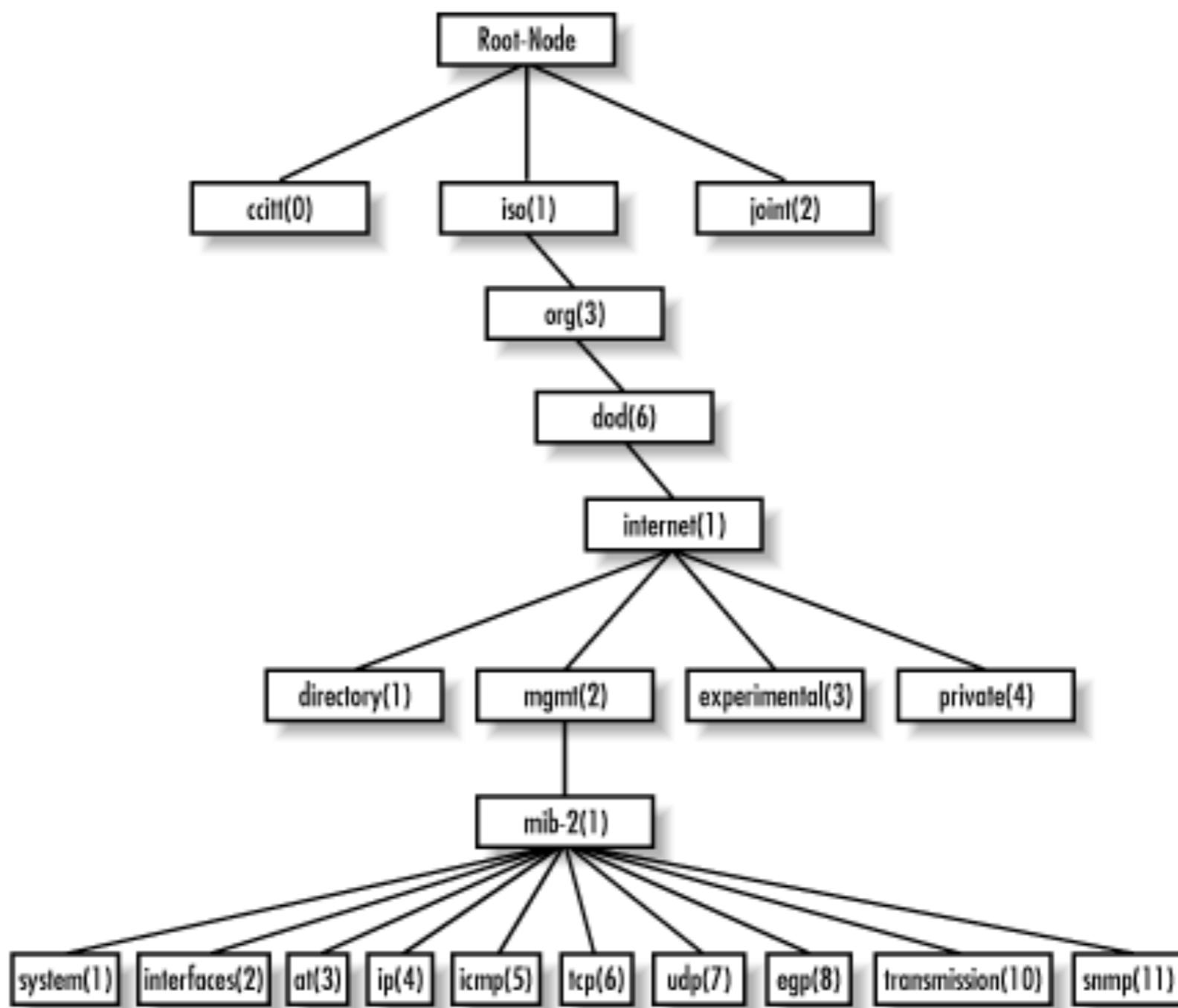
# MIBs

```
IfEntry ::=  
SEQUENCE {  
    ifIndex           InterfaceIndex,  
    ifDescr           DisplayString,  
    ifType            IANAifType,  
    ifMtu             Integer32,  
    ifSpeed           Gauge32,  
    ifPhysAddress     PhysAddress,  
    ifAdminStatus     INTEGER,  
    ifOperStatus      INTEGER,  
    ifLastChange      TimeTicks,  
    ifInOctets        Counter32,  
    ifInUcastPkts    Counter32,  
    ifInNUcastPkts   Counter32, -- deprecated  
    ifInDiscards      Counter32,  
    ifInErrors        Counter32,  
    ifInUnknownProtos Counter32,  
ifOutOctets       Counter32,  
    ifOutUcastPkts   Counter32,  
    ifOutNUcastPkts  Counter32, -- deprecated  
    ifOutDiscards     Counter32,  
    ifOutErrors       Counter32,  
    ifOutQLen         Gauge32,  -- deprecated  
    ifSpecific        OBJECT IDENTIFIER -- deprecated  
}
```

# IF-MIB::OutOctets

```
[igb@offsite7 ~]$ snmpbulkwalk -v3 -u cacticacti -l authpriv -X '-deleted-' -A '-deleted-' \
    udp6:rb2011-1.batten.eu.org IF-MIB::ifOutOctets
IF-MIB::ifOutOctets.1 = Counter32: 0
IF-MIB::ifOutOctets.2 = Counter32: 0
IF-MIB::ifOutOctets.3 = Counter32: 0
IF-MIB::ifOutOctets.4 = Counter32: 997357259
IF-MIB::ifOutOctets.5 = Counter32: 890378998
IF-MIB::ifOutOctets.6 = Counter32: 0
IF-MIB::ifOutOctets.7 = Counter32: 18064840
IF-MIB::ifOutOctets.8 = Counter32: 70886570
IF-MIB::ifOutOctets.9 = Counter32: 61591757
IF-MIB::ifOutOctets.10 = Counter32: 55158595
IF-MIB::ifOutOctets.11 = Counter32: 318834666
IF-MIB::ifOutOctets.17 = Counter32: 93797603
IF-MIB::ifOutOctets.18 = Counter32: 47336681
IF-MIB::ifOutOctets.20 = Counter32: 2673743280
IF-MIB::ifOutOctets.24 = Counter32: 47324115
IF-MIB::ifOutOctets.26 = Counter32: 10708521
IF-MIB::ifOutOctets.15728640 = Counter32: 3100226
[igb@offsite7 ~]$
```

# MIB Structure



# Numeric OIDs

```
[igb@offsite7 ~]$ snmpbulkwalk -On -v3 -u cacticacti -l authpriv \
-X '-deleted-' -A '-deleted-' udp6:rb2011-1.batten.eu.org IF-MIB::if0utOctets
.1.3.6.1.2.1.2.2.1.16.1 = Counter32: 0
.1.3.6.1.2.1.2.2.1.16.2 = Counter32: 0
.1.3.6.1.2.1.2.2.1.16.3 = Counter32: 0
.1.3.6.1.2.1.2.2.1.16.4 = Counter32: 997366016
.1.3.6.1.2.1.2.2.1.16.5 = Counter32: 890411588
.1.3.6.1.2.1.2.2.1.16.6 = Counter32: 0
.1.3.6.1.2.1.2.2.1.16.7 = Counter32: 18070148
.1.3.6.1.2.1.2.2.1.16.8 = Counter32: 70903630
.1.3.6.1.2.1.2.2.1.16.9 = Counter32: 61702815
.1.3.6.1.2.1.2.2.1.16.10 = Counter32: 55172277
.1.3.6.1.2.1.2.2.1.16.11 = Counter32: 318884246
.1.3.6.1.2.1.2.2.1.16.17 = Counter32: 93833332
.1.3.6.1.2.1.2.2.1.16.18 = Counter32: 47411302
.1.3.6.1.2.1.2.2.1.16.20 = Counter32: 2673780229
.1.3.6.1.2.1.2.2.1.16.24 = Counter32: 47398246
.1.3.6.1.2.1.2.2.1.16.26 = Counter32: 10714986
.1.3.6.1.2.1.2.2.1.16.15728640 = Counter32: 3106731
[igb@offsite7 ~]$
```

# Descriptions

```
[igb@offsite7 ~]$ snmpbulkwalk ... IF-MIB::ifDescr  
.1.3.6.1.2.1.2.2.1.2.1 = STRING: sfp1-spare  
.1.3.6.1.2.1.2.2.1.2.2 = STRING: ether1-imac  
.1.3.6.1.2.1.2.2.1.2.3 = STRING: ether2-switch-master  
.1.3.6.1.2.1.2.2.1.2.4 = STRING: ether3-airport  
.1.3.6.1.2.1.2.2.1.2.5 = STRING: ether4-netgear-link  
.1.3.6.1.2.1.2.2.1.2.6 = STRING: ether5-spare  
.1.3.6.1.2.1.2.2.1.2.7 = STRING: ether6-printer  
.1.3.6.1.2.1.2.2.1.2.8 = STRING: ether7-pi-one  
.1.3.6.1.2.1.2.2.1.2.9 = STRING: ether8-pi-two  
.1.3.6.1.2.1.2.2.1.2.10 = STRING: ether9-pi-three  
.1.3.6.1.2.1.2.2.1.2.11 = STRING: ether10-dsl320b  
.1.3.6.1.2.1.2.2.1.2.17 = STRING: pppoe-aa-uplink  
.1.3.6.1.2.1.2.2.1.2.18 = STRING: bridge-vlan-5-redzone  
.1.3.6.1.2.1.2.2.1.2.20 = STRING: bridge-default-vlan  
.1.3.6.1.2.1.2.2.1.2.24 = STRING: vlan-5-pi-one  
.1.3.6.1.2.1.2.2.1.2.26 = STRING: vlan-5-mac-mini  
.1.3.6.1.2.1.2.2.1.2.15728640 = STRING: <l2tp-igb-l2tp>  
[igb@offsite7 ~]$
```

# SNMP Operations

- GET fetches a requested OID
- GETNEXT fetches the next OID after the requested one
- snmpwalk (and friends) can fetch entire MIB, or subtrees thereof, using GETNEXT
- GETBULK asks for a number of GETNEXT operations to be performed and returns them all in one operation (not always available)
  - Absolutely fantastic for Amplification Attacks

# Counters

- Interface statistics are provided as counters since restart, not as rates or counts in interval
  - No guarantee device has accurate clock
  - May be multiple clients
- 32-bit counters wrap around trivially (every 4GB)
- Heuristic is that if a counter has gone backwards it's actually a wrap-around
  - If there's a risk of wrap around inside polling interval, hardly difficult with 5 minute polling, you need 64 bit counters

# SNMP Implementation

- Originally UDP, because of fear that TCP was too hard for embedded devices and TCP handshakes too inefficient
- Most implementations ended up being UDP only
- With hindsight, wrong decision: you need management information to be reliable
- SNMP now often available over TCP, which is preferred
  - SNMP Proxying allows TCP over WAN and then UDP over LAN to less-capable devices

# SNMP Proxy

```
[igb@offsite7 ~]$ snmpbulkwalk -v3 -u cacticacti -l authpriv \
-X '-deleted-' -A '-deleted-' \
-n gs108tv2-1 tcp6:snmp-proxy.batten.eu.org IF-MIB::ifOutOctets
IF-MIB::ifOutOctets.1 = Counter32: 2487668549
IF-MIB::ifOutOctets.2 = Counter32: 875703010
IF-MIB::ifOutOctets.3 = Counter32: 0
IF-MIB::ifOutOctets.4 = Counter32: 374489695
IF-MIB::ifOutOctets.5 = Counter32: 2888671020
IF-MIB::ifOutOctets.6 = Counter32: 1245556106
IF-MIB::ifOutOctets.7 = Counter32: 1879710128
IF-MIB::ifOutOctets.8 = Counter32: 4050225960
IF-MIB::ifOutOctets.13 = Counter32: 273726231
IF-MIB::ifOutOctets.14 = Counter32: 0
IF-MIB::ifOutOctets.15 = Counter32: 0
IF-MIB::ifOutOctets.16 = Counter32: 0
IF-MIB::ifOutOctets.17 = Counter32: 0
[igb@offsite7 ~]$
```

# SNMP Security

- snmp v1, v2, v2c: absolute joke
- A “community string” is placed, in clear, in the packet. Any packet with the right community string is OK.
  - Killed use of SNMP for SET operations stone dead, and makes traps (UDP!) dangerous
  - Community string is usually “public”, and you rely on firewalls to provide some privacy

# SNMP v2

```
mini-server:~ igb$ snmpget -v2c -c public \
  batten-eu-org-dual-band.home.batten.eu.org SNMPv2-MIB::sysDescr.0
SNMPv2-MIB::sysDescr.0 = STRING: Apple AirPort - Apple Inc., 2006-2012.
mini-server:~ igb$
```

# SNMPv2

09:49:24.465620 IP mini-server.home.batten.eu.org.49233 > batten-eu-org-dual-band.home.batten.eu.org.snmp:

```
GetRequest(28) system.sysDescr.0
0x0000: 0024 369e 04f6 0026 bb60 07ce 0800 4500 .$.6....&.`....E.
0x0010: 0047 ab68 0000 4011 0000 0a5c d5b1 0a5c .G.h..@....\...\
0x0020: d59a c051 00a1 0033 c048 3029 0201 0104 ...Q...3.H0)....
0x0030: 0670 7562 6c69 63a0 1c02 045b a9e9 cd02 .public....[....
0x0040: 0100 0201 0030 0e30 0c06 082b 0601 0201 ....0.0...+....
0x0050: 0101 0005 00 .....
```

09:49:24.466352 IP batten-eu-org-dual-band.home.batten.eu.org.snmp > mini-server.home.batten.eu.org.49233:

```
GetResponse(88) system.sysDescr.0="Apple AirPort - Apple Inc., 2006-2012. All rights Reserved."
0x0000: 0026 bb60 07ce 0024 369e 04f6 0800 4500 .&.`....$6....E.
0x0010: 0083 8d81 0000 4011 2ce5 0a5c d59a 0a5c .....@.,..`....\
0x0020: d5b1 00a1 c051 006f 6f4e 3065 0201 0104 ....Q.ooN0e....
0x0030: 0670 7562 6c69 63a2 5802 045b a9e9 cd02 .public.X..[....
0x0040: 0100 0201 0030 4a30 4806 082b 0601 0201 ....0J0H..+....
0x0050: 0101 0004 3c41 7070 6c65 2041 6972 506f ....<Apple.AirPo
0x0060: 7274 202d 2041 7070 6c65 2049 6e63 2e2c rt.-.Apple.Inc.,
0x0070: 2032 3030 362d 3230 3132 2e20 2041 6c6c .2006-2012...All
0x0080: 2072 6967 6874 7320 5265 7365 7276 6564 .rights.Reserved
0x0090: 2e
```

# SNMPv2

- Clearly unacceptable for use in WAN environments
- Lack of error reporting in most agents plus lack of rate limiting means community-string guessing is also a reasonable attack
- Exposes data useful to attacker (descriptions, topology)

# SNMPv3 security

- Multiple usernames with different views and access, “contexts”, etc.
- Authentication and encryption
- Authentication:
  - Insert string into packet, hash the whole packet, replace where the string was with the hash
  - Encryption using pre-shared key
- Also now available over TLS

# Sadly . . .

- SNMPv3 is tricky to set up, as a result
  - Not all devices support it (depressingly common to only have SNMPv2 over UDP IPv4, cf. Apple Airports, Netgear switches, etc)
  - Alternatively, it's there, but it's buggy, and it's CLI-only with the GUI only doing v2
  - Not easy manager side, either
- Even systems that do support v3 end up being used with SNMPv2.

# Lots of broken implementations

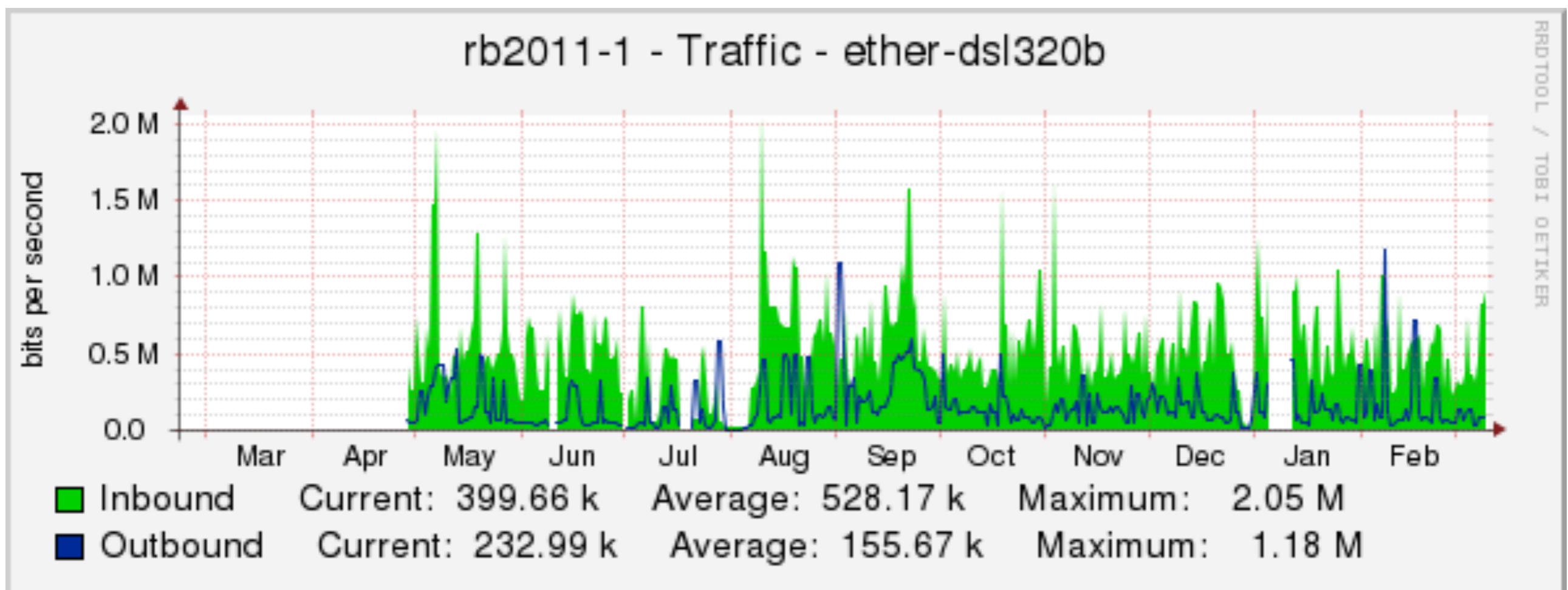
Showing Rows 1 to 19 of 19 [1]			
States	Status	In State	Hostname
	Up	-	tcp6:snmp-proxy.batten.eu.org
	Up	-	tcp6:batten-mac-mini.batten.eu.org
	Down	0d 1h 15m	tcp6:downstairs-imac.batten.eu.org
	Up	-	tcp6:snmp-proxy.batten.eu.org
	Up	-	tcp6:snmp-proxy.batten.eu.org
	Up	-	127.0.0.1
	Up	-	tcp6:mini-server.batten.eu.org
	Up	-	tcp6:mail.batten.eu.org
	Up	-	tcp6:offsite6.batten.eu.org
	Up	-	udp:127.0.0.1
	Up	-	tcp6:pl-one.batten.eu.org
	Up	-	tcp6:pl-three.batten.eu.org
	Down	1d 12h 45m	tcp6:pl-two.batten.eu.org
	Up	-	udp6:rb2011-1.batten.eu.org
	Disabled	-	udp6:rb2011-1.batten.eu.org
	Down	0d 0h 55m	tcp6:ruths-mac-mini.batten.eu.org
	Disabled	3d 2h 20m	srw2008.home.batten.eu.org
	Up	-	tcp6:snmp-proxy.batten.eu.org
	Down	6d 16h 15m	tcp6:vpn.batten.eu.org
Showing Rows 1 to 19 of 19 [1]			

Airports: v2 over UDP over IPv4 only

Netgear: UDP IPv4 only, v3 limited to only one user and therefore needs to know “admin” password

MikroTik: Does v3 over IPv6, but does not support TCP.

# SNMP Uses



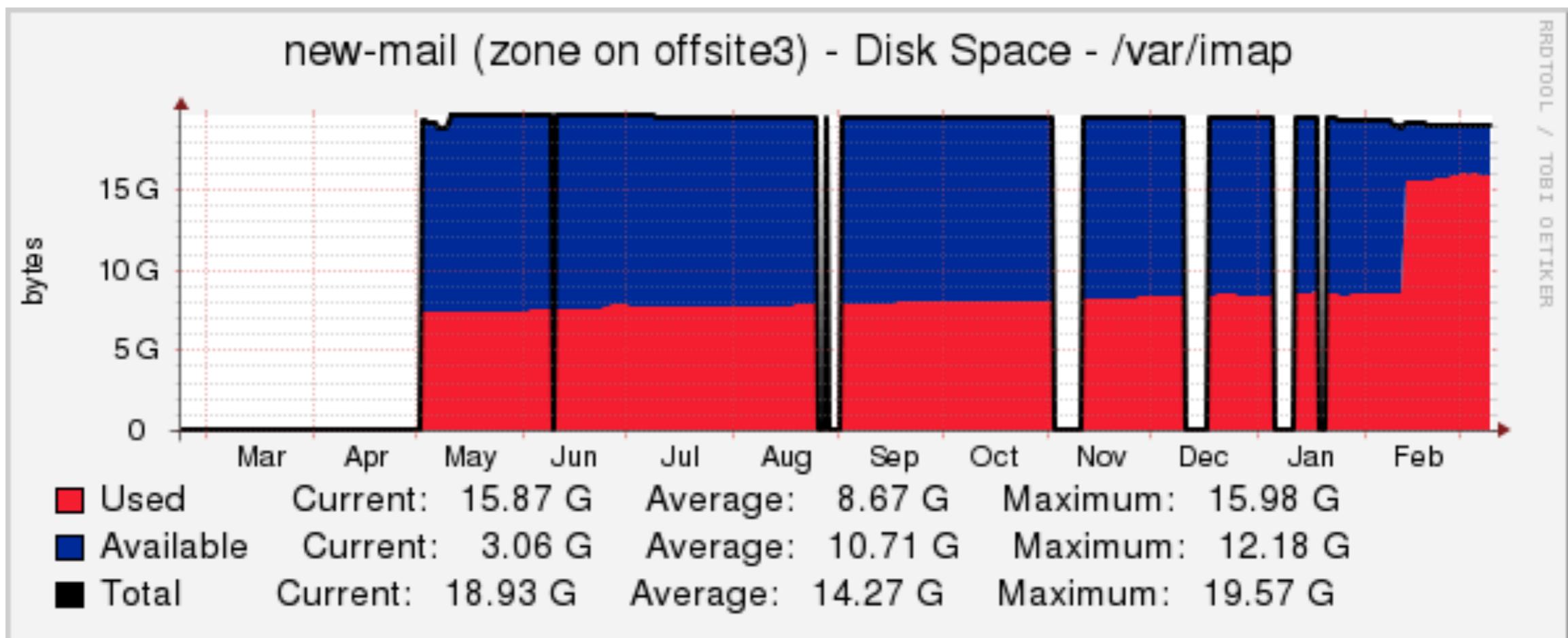
# SNMP Traps

- Less widely used, because only sent once over insecure and unreliable transport
- Much better to poll devices for problems than rely on single cry for help
- Typical use is to use traps to provoke polling, but poll anyway
  - “Cold Start” is useful because it indicates things like interface numbers might have changed

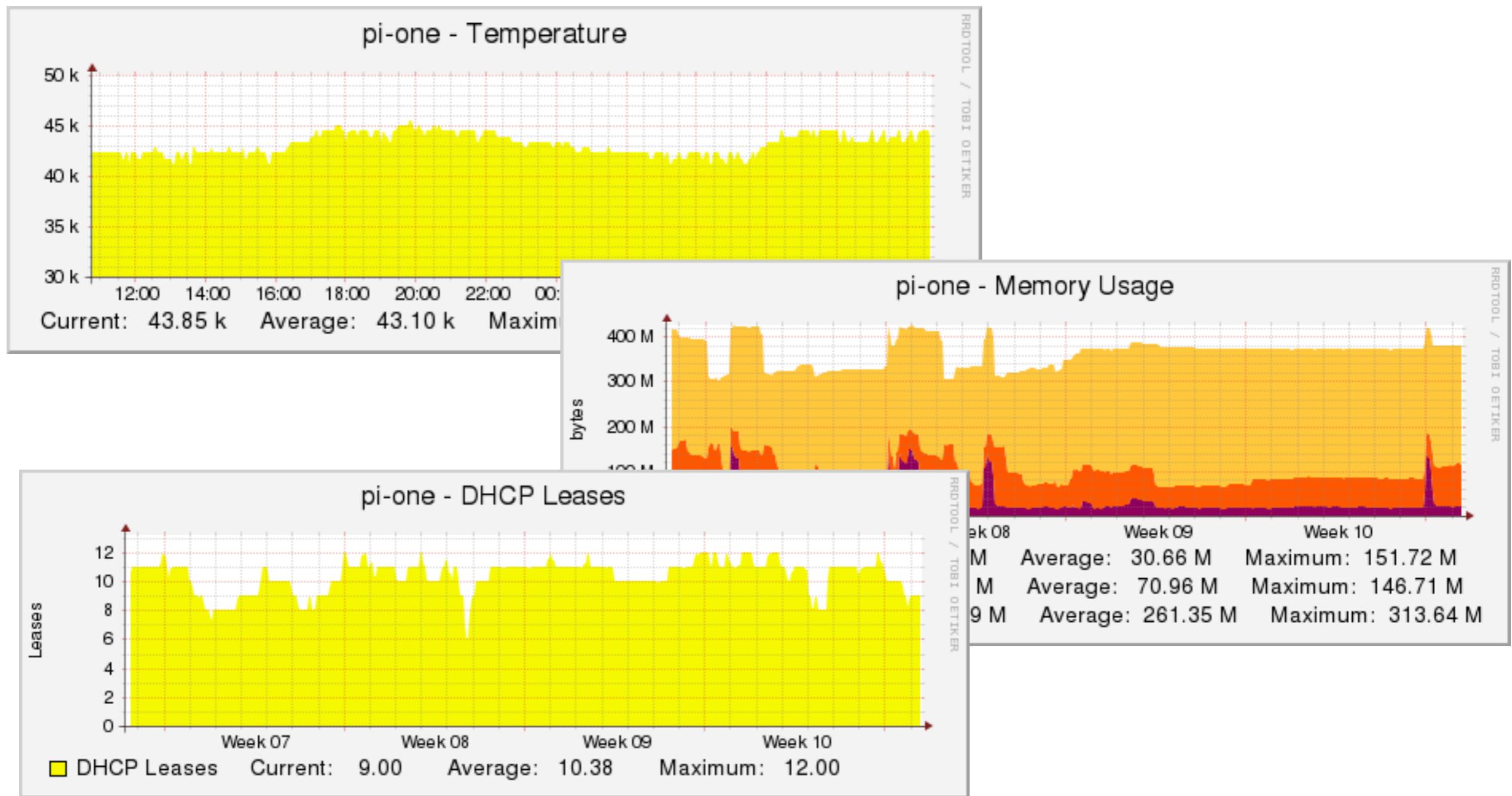
# Not-network use

- Handy for monitoring and graphing disk, memory, temperature, etc on hosts
- Handy for getting reboot notifications via Cold Start traps
- MIBs exist for Apache, Cyrus, etc, etc, mostly to tie into graphing packages like Cacti.

# Disk Usage



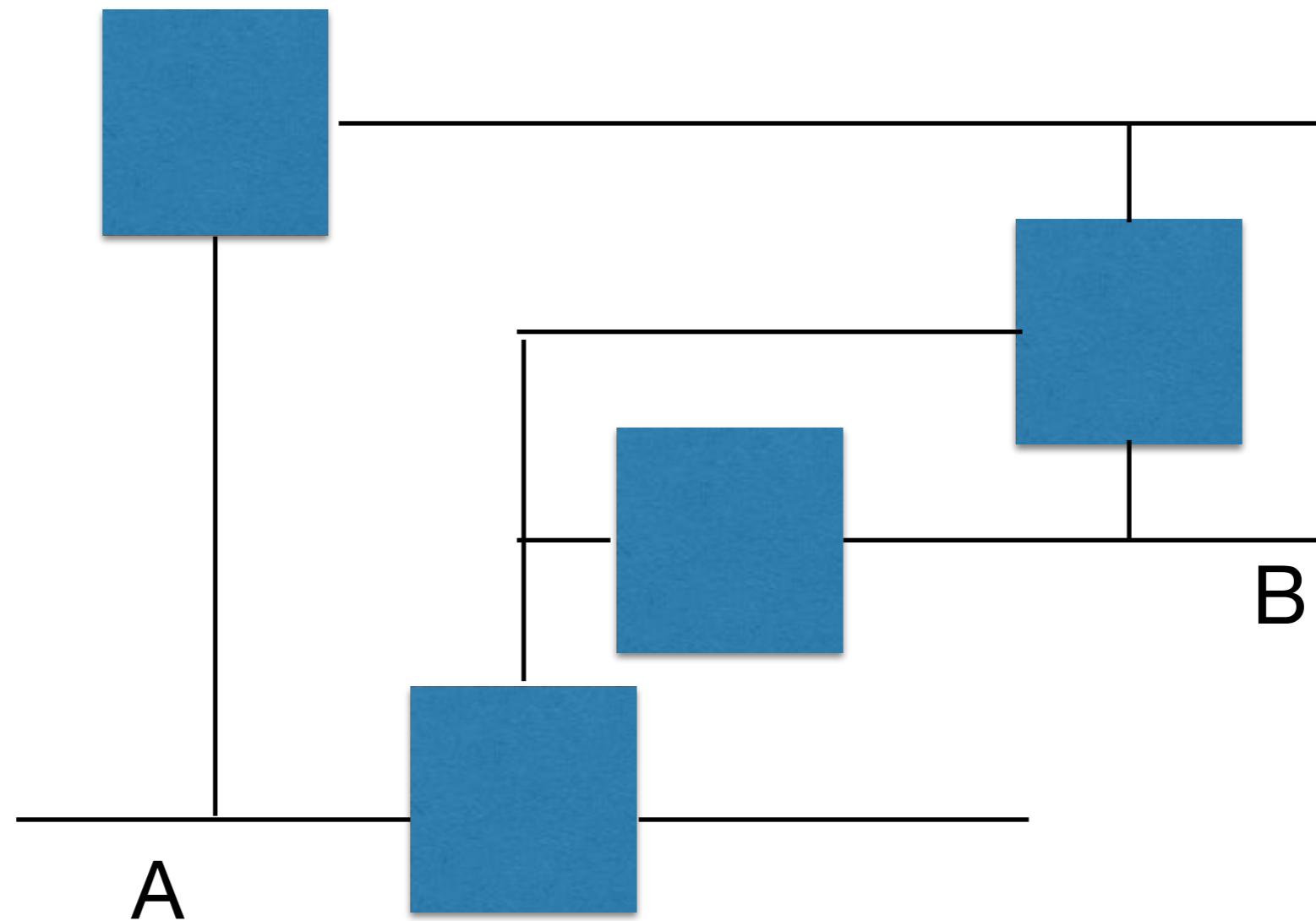
# Stuff



# Networks 19: Routing Protocols

[i.g.batten@bham.ac.uk](mailto:i.g.batten@bham.ac.uk)

# The problem



# IP Routing

- Network doesn't do it for us: we are responsible for knowing the best “next hop”
- We can't (in 2017) dictate the route beyond the next hop
- There is no global (or local) protocol for co-ordinating routing. Loops can arise.

# IP Resilience

- There are various ways to spare routers
  - VRRP, HSRP, proprietary N+1 inside or outside switches
- There are various ways to spare links at layer two
  - LCP and Link Agg, “Spanning Tree”, proprietary or media-specific buddyng systems
- But you really want complete resilience in the event of major failures

# Static Routing

- In simple network, you set routing tables up by hand
- “This network goes to this router, this network goes to this router, everything else goes to the Internet via this router”.
- No resilience, but unlikely to go wrong so long as all the links stay up

# Interior v Exterior Routing

- Different mechanisms used for routing inside the enterprise and outside the enterprise.
- “Interior” protocols are used when all the equipment is to some greater or lesser extent under one management
- “Exterior” protocols are used between enterprises

# Interior

- Far fewer networks (10s is a lot, 100s is rare, 1000s very unusual)
- More trust between equipment and effective sanctions available if it goes wrong
- Potentially a central management authority or system
- Reasonable to maintain a full set of routing tables

# Exterior

- In the limit, needs to be able to handle the entire Internet ( $2^{24}$  routes and in the future up to  $2^{64}$ )
- Partial routing tables a reasonable response
- No central authority and no effective sanctions, so needs to be robust and secure (FSVO “secure”).

# Objectives

- In simple networks, there is only one sensible path between points and the task is to find it
  - Spanning tree which doesn't change (and there's a unique solution)
- In more complex networks, there are multiple paths between points and the task is to find the best
  - Minimal spanning tree which changes rarely (possibly need to choose between alternatives)
- In the largest networks, there are multiple paths between points and a high rate of change, so the task is to find the best **now**.
  - Minimal spanning tree with requirements for stability (lots of options, lots of change)

# Metrics

- Minimal spanning trees require costs associated with edges
- Those costs normally expressed as simple integers
- Problem in networks is that we might be interested in bandwidth, latency, reliability, security, cost...
  - OS routing tables don't deal with workload-specific routing, and protocols which purport to handle this usually reduce to a single integer by magic formulae.
  - Interesting research topic ("policy based routing")

# Distance Vector

- Simplest and earliest algorithm
- Doesn't require computation of a spanning tree by any one party
- Quick and easy to implement
- Gives quick and dirty results

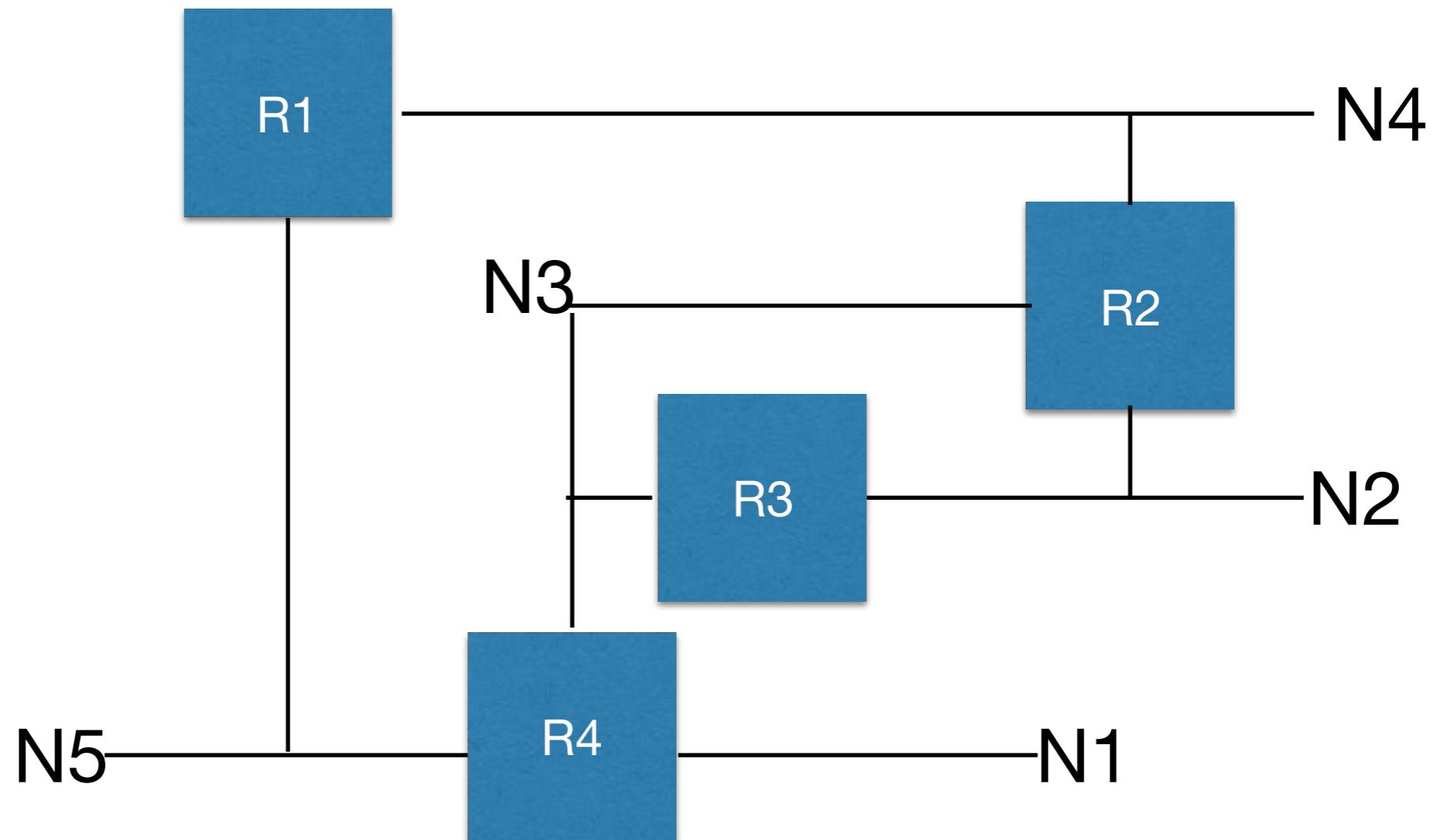
# RIP

- Each connected network is one “hop” away
- Each node broadcasts all the networks it knows how to reach, with their “hop count”
- When you receive a routing update, you add one to the included “hop counts” and add it to the routing database.
- Best options are used for kernel routing tables

# RIP

- Metrics are only 0 to 16, 16 = unreachable
- Sets maximum diameter on network
- Limits ability to use metrics >1 to indicate slow or unreliable links

# The problem, labelled



# As a matrix

	N1	N2	N3	N4	N5
R1				X	X
R2		X	X	X	
R3		X	X		
R4	X		X		X

# Consider Routers 1 and 2

- R1 broadcasts “I can reach N4 and N5, directly connected”
- R2 broadcasts “I can reach N2, N3, N4, directly connected”
- R2 learns a route to N5 via R1 over their shared N4
- R1 learns a route to N2 and N3 via R2 over their shared N4.
- R1 now broadcasts “I can reach N4 and N5, connected, and N2 and N3, 1 hop away”.
- R2 now broadcasts “I can reach N2, N3, N4, connected, and N5, 1 hop away”.

# Exercise

- Spend five minutes working out the routing tables everyone gets
- Assume the only metric in use is hop count

# Computed Routes

	N1	N2	N3	N4	N5
R1	R4	R2	R4/R2	X	X
R2	R3	X	X	X	R1
R3	R4	X	X	R2*	R4
R4	X	R3/R2	X	R1	X

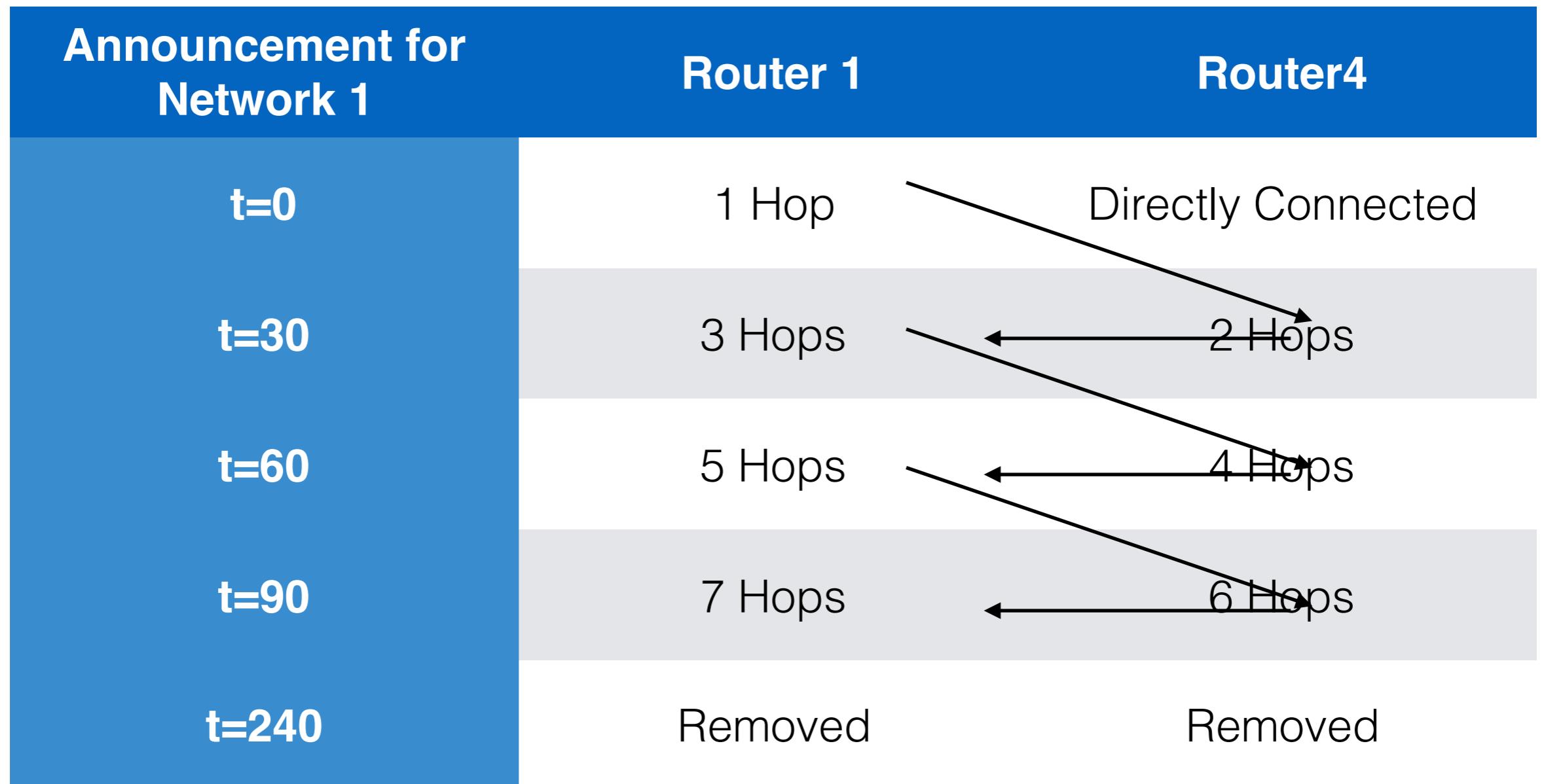
# Problems with Distance Vector

- Routing loop
  - Router1 believes it has a route to Network1 via Router4.
  - Router4's interface to Network1 goes down
  - Router4 will believe an update from Router1 saying "I know how to get to Network 1, two hops away"
  - Takes some time to damp this problem down

# Problems with routing loops

- Packets are sent forwards and backwards until the packet TTL hits zero
- Initial TTL in packets has been increasing in recent years as diameter of Internet increases: could be as much as 60
- In process generates lots of ICMP messages, triggers IDSes, etc, etc, etc, as well as burning bandwidth.

# R4 i/f N1 breaks at t=10



# Solution 1: Flash Update

- Standard RIP sends a packet every 30s
- Therefore can take four minutes for routes to converge after a topology change
- Instead, send a packet on every update which makes changes, and send a packet on every change of local interface status
- Same progression, but much quicker

# Solution 2: Split Horizon (now mandatory)

- Instead of making same announcement out of each interface, only announce from each interface routes reachable from other networks
  - Don't announce routes which involve turning packets around and sending them back out of the interface they arrived over
- Prevents simple routing loops like this, but more complex analogues exist
- Requires more book-keeping and more computation (but not very much)

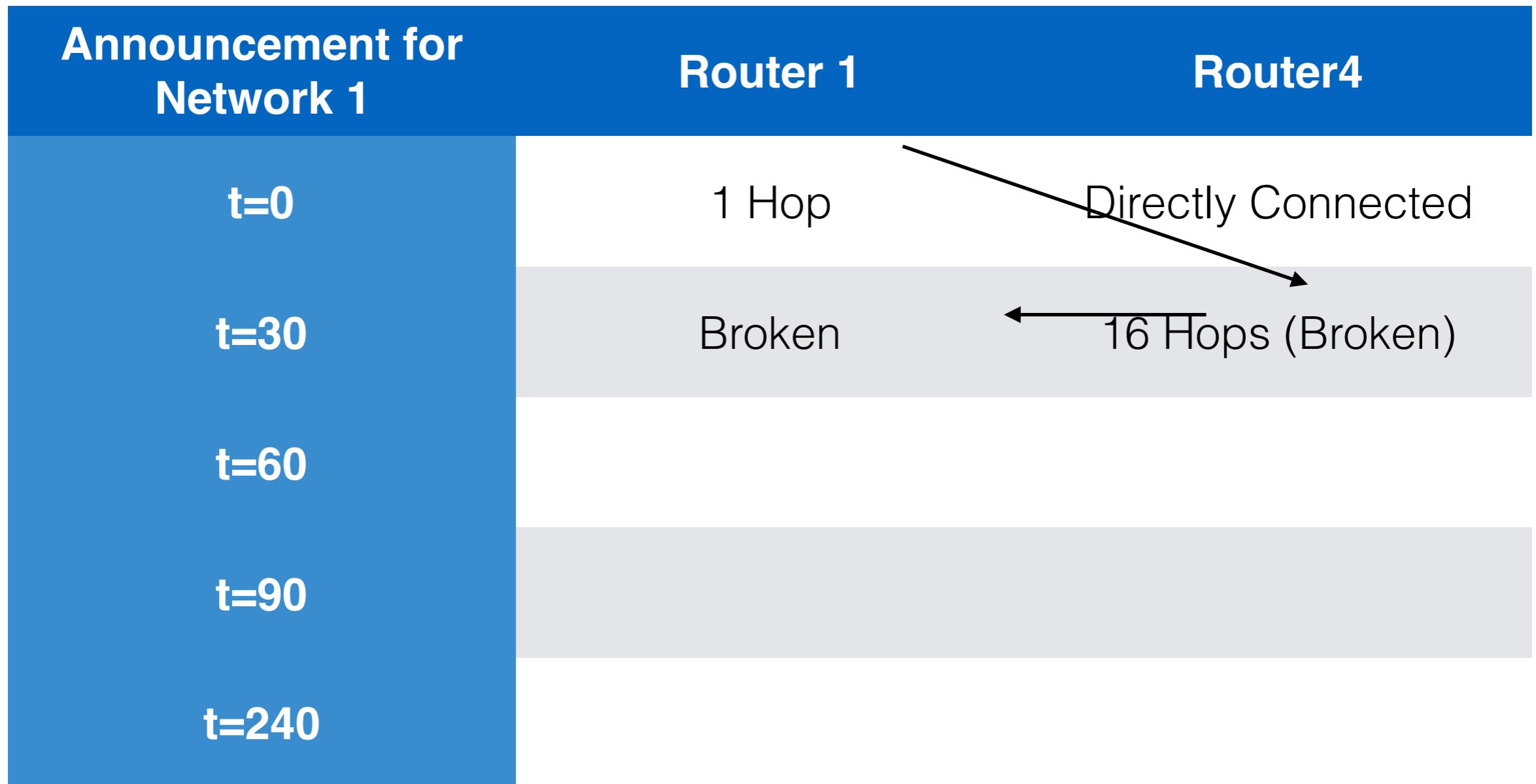
# R4 interface N1 breaks at t=10: split horizon

Announcement for Network 1		Router 1	Router4
t=0		1 Hop	Connected
t=30		1 Hop	Nothing
t=60		1 Hop	Nothing
t=180 (or 210)		Nothing	Nothing

# Solution 3: Poison Reverse

- When a link fails, immediately send an update with metric 16
- Then proceed as normal

# R4 interface N1 breaks at t=10, Poison Reverse



# RIP with mod cons

- RIP with split horizon, poison reverse and flash updates works tolerably well on small networks
- Still limited by small limit on diameter and crude metrics
- Topology changes spread quite slowly
- Modern implementations are more complex (“hold down timers”, for example)
- Attempts to change protocol timers lead to wild instability unless implemented network-wide at the same time

# RIPv2

- RIPv1 is from when dinosaurs walked the earth, and therefore is “classful”
  - Networks starting 0 are /8, networks starting 10 are /16, networks starting 110 are /24
  - Impossible to use in modern networks with subnetting
- RIPv2 incorporates subnet masks, and will run over multicast rather than broadcast
- Not widely adopted

# RIP Security

- RIPv2 introduces MD5-based authentication, in the same style as SNMPv3
  - Insert string, hash packet, put hash where the string was.
  - Sequence numbers used to prevent replay attacks
- Used by...almost no-one.

# RIPng

- RIPv2 modified to handle IPv6
- Networks simple enough to work with RIPng are simple enough to not need RIPng, I suspect
  - Much more layer 2 switching means fewer layer 3 networks.

# Link State Protocols

- Instead of sending out reachability information, devices in an *area* exchange information about all links that are active.
- Every device on network can therefore calculate a minimal spanning tree
- Link information is flooded throughout the area, so convergence is quick: devices switch from one consistent set of routing tables to another

# OSPF

- Open Shortest Path First
- Link State Protocol
- Usable on very large networks, and has additional features to help with this
- Supports authentication (same method, and caveat, as RIPv2)

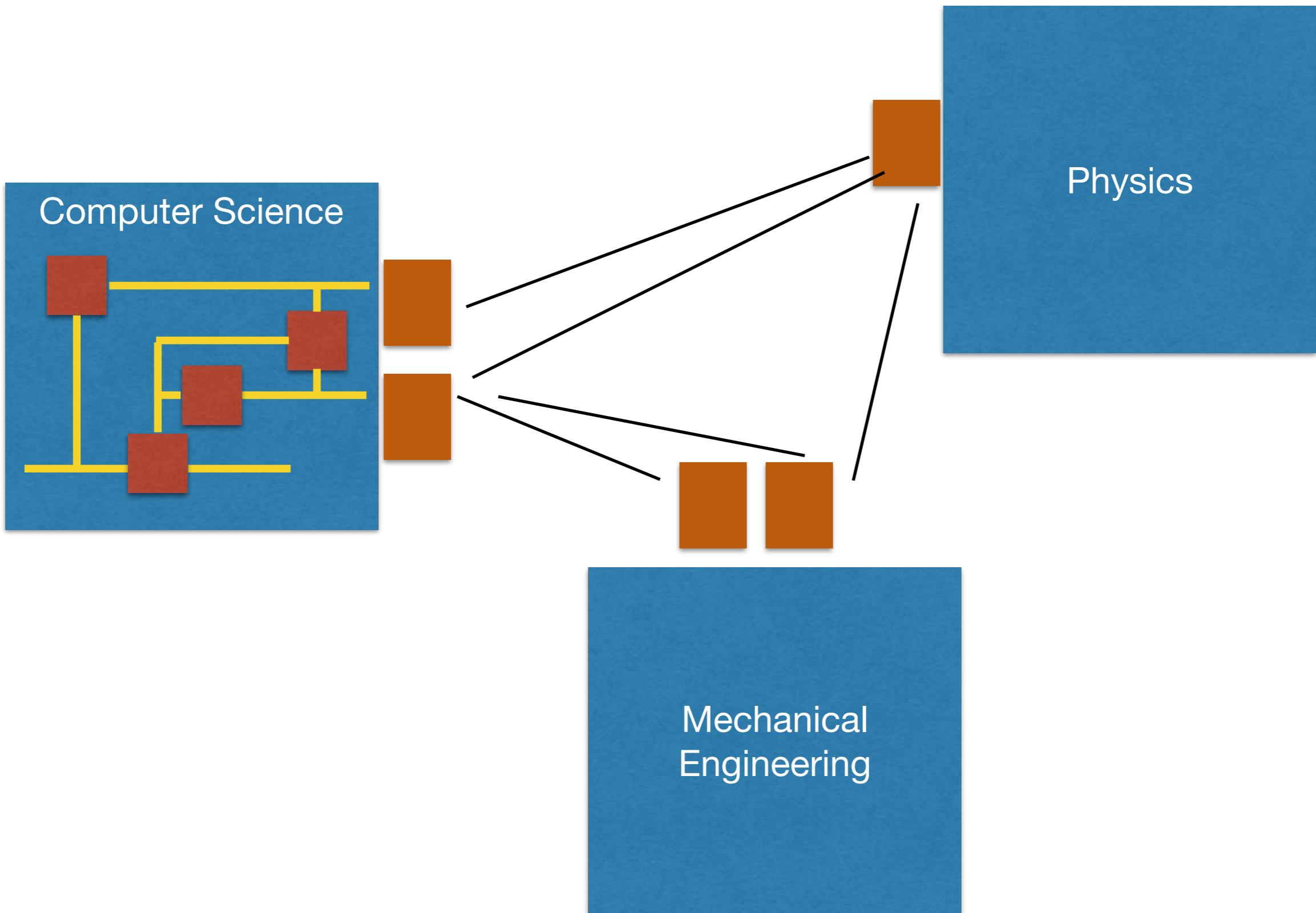
# OSPF Basic Operation

- Devices on a subnet exchange HELLO packets to learn about their local neighbours
- They elect a “Designated Router” and a “Backup Designated Router” from the devices that have multiple interfaces (based on preferences pre-configured into the routers, then router ID as a tie-break)
- The DR and BDR exchange link state advertisements (LSA) with neighbouring routers
- When the LSA information changes, the routing algorithm is used to recompute a set of routing tables
- The DR and BDR announce a complete set of non-local routes to other systems on the local network

# OSPF Areas

- Most networks have a small group of highly connected core routers, with networks connected to one of the core routers
  - For example, core router per department, meshed, with departmental networks connected to departmental router
  - Pointless to send updates about changes inside a department campus wide, as always reached via departmental network
- OSPF can divide networks into Areas, with “out of area” routes summarised rather than re-calculated by everyone

# OSPF Areas



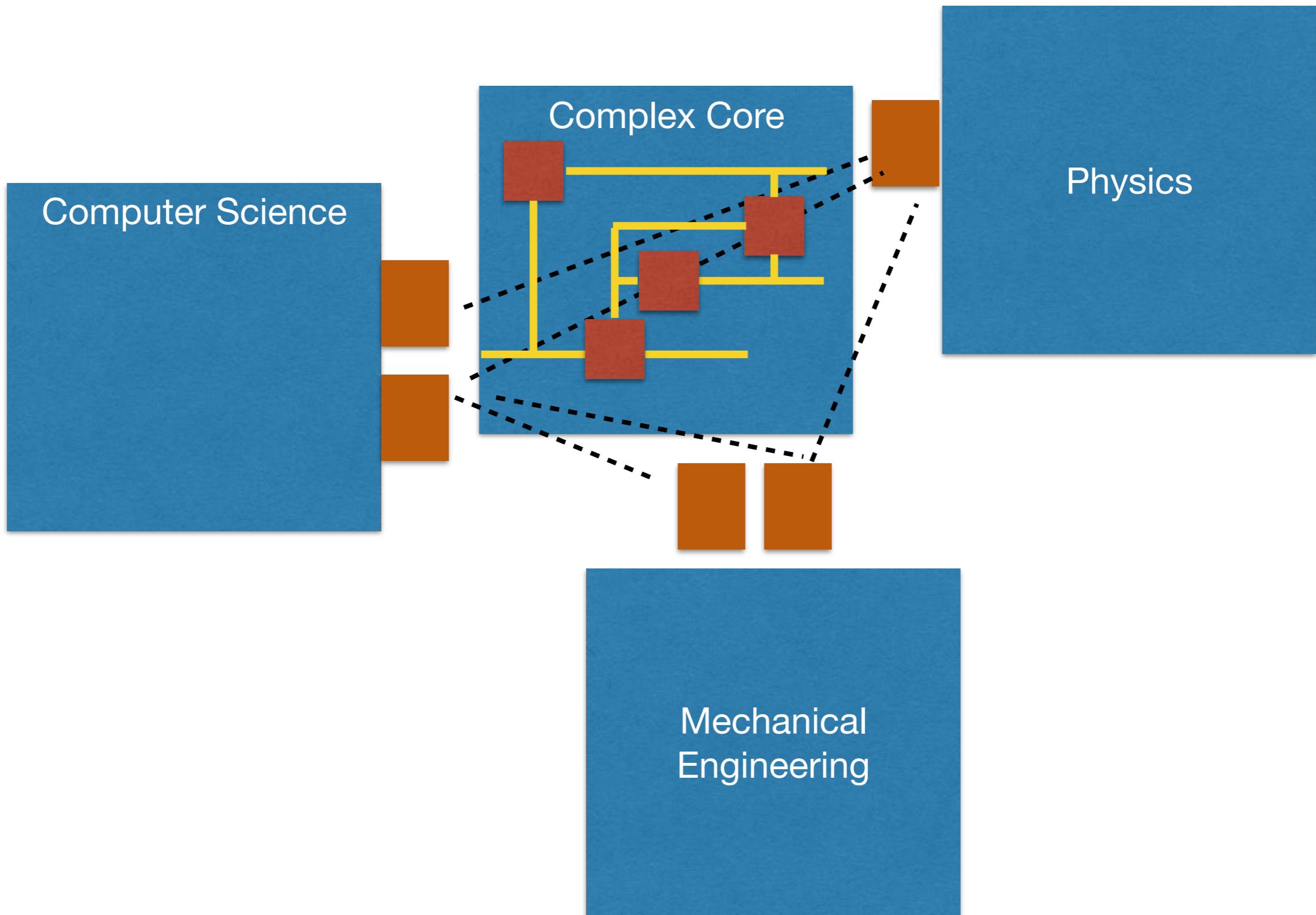
# OSPF Areas

- So Computer Science's routers will advertise all the internal networks as part of the area they provide access, and other routers will compute routes to an area that includes all those networks as a group.
- Reduces opportunities for fine-grained decisions about which core link is “closer” to individual networks in an area
- But as we have an  $n/\log n$  problem computing the minimal cost tree, reducing  $n$  is always good

# OSPF Areas

- OSPF doesn't really support a hierarchy of areas: the intention is that there is a “backbone” network which is simply connected, and each area connects to a backbone network.
- In more complex cases, the backbone can rely on transit through a more complex area with “virtual links”
  - Core routers know about each other, and pretend they are directly linked

# OSPF Virtual Links



# Advantages of OSPF

- Converges within a few seconds of topology change, as DR and BDR routers exchange LSAs on demand
  - Routers can pass on LSA information before they have performed the recomputation: DV protocols pass on “cooked” information, while LS protocols pass on “raw” information
- Routing loops are rare or short-lived, because everyone is working from the same information

# Problems with OSPF

- (Historically) CPU intensive, with solutions (“Inter-area announcements”) quite difficult to get right.
- Complex to configure correctly
- Shortage of good open-source implementations (“gated” had complex licensing, “zebra” and “quagga” niche and difficult to use).
- Proprietary alternatives sometimes preferred

# Alternatives

- IGRP (classful) and EIGRP (classless) from Cisco
- Proprietary, but widely reverse-engineered; standard released in 2013
- Distance Vector protocol with more sophisticated metrics and additional information
  - Does incremental updates rather than sending the whole routing table each time
- Usable for large, complex networks, but tricky if it is heterogenous.

# The real world

- OSPF is required for large, complex networks
  - Rate of change, even manual change, too high for static routing
  - In un-spared networks, static routing probably better than RIP
  - OSPF Areas allow statically routed departmental networks with dynamic core routing

# Load Balancing

- Implementations can load balance over links with the same metric (RIP) or with metrics that are within some bound of each other (OSPF, EIGRP).
- Layer-3 load balancing can play very badly with firewalls and NAT (“asymmetric routing”)
- Probably better ways to do it today.

# Summary

- RIP is easy to understand, easy to use and usually the wrong answer
- OSPF is complex to understand, very complex to use correctly and very effective
- Static routing isn't cool, but is worth sticking with as long as you can.

# External Routing

- Autonomous Systems
  - Large networks on the Internet are called “autonomous systems”
  - They have an AS Number to identify them
  - Every routable network is a member of exactly one AS

# Autonomous System

- Typical AS is “an ISP and all of its small and medium customers”
- But what happens when you want to have multiple ISPs, for reasons of resilience, negotiating power, geography, etc?
- Answer: get an AS Number and some provider-independent IP address space (not easy these days)
  - Only went from two-byte to four-byte ASNs in 2010

# AS a bit like OSPF Area

- Assumption that everyone inside the AS knows how to reach everyone else inside the AS, and that there is a small number of entry/exit points to the AS from which those networks can be reached.
- In fact, BGP sometimes used an an Interior Routing Protocol by very large organisations for whom OSPF doesn't scale

# BGP

- Routers peer over **TCP**
  - Makes dealing with message loss and node failure easier
- Updates are incremental
  - Routing tables changed on each update, rather than all at once
- Uses slightly different approach to RIP or OSPF

# Path Vectors

- Conceptually, BGP is a distance vector algorithm: the route to a network reached by another router has a cost equal to the cost that router advertises, plus the cost of getting to the router.
  - BGP has extensive support for policy-based decisions, load spreading and so on

# AS Path

- But each route also carries with it a vector containing an ordered list of all the ASes that the route passed through
- So if AS45 peers with AS90, and AS90 advertises that route to its peer AS135, which advertises it to AS200, the AS Path (45, 90, 135, 200) will be known to AS200.
- Allows easy detection of routing loops

# Pro Tip

- I once asked the architect of the UK's first ISP, who went on to found the London Internet Exchange (LINX), whether I should get an ASN and start talking to two ISPs
- “You'll have more trouble from that than your single ISP will cause you: get redundant links to a single ISP and talk OSPF to them”.
- Ten years later, as I was leaving, we got an ASN and two ISPs to satisfy a customer requirement and started talking BGP
- From what I hear, Keith was absolutely right...

