MATH0078/MATH0123: Networking

Russell Bradford

Contents

Acronyms

AAL	ATM Adaption Layer
ADSL	asymmetric digital subscriber line
AH	authentication header
ARPA	Advanced Research Projects Agency
ARP	Address Resolution Protocol
AS	autonomous system
ATM	asynchronous transfer mode
AUI	thick Ethernet socket
BGP	Border Gateway Protocol
BNC	British Naval Connector
BSS	Basic Service Set
CAN	Community area network
CCK	Complementary Code Keying
CDMA	code division multiple access
CE	congestion experienced
CIDR	classless interdomain routing
CNAME	canonical name
CRC	cyclic redundancy check
CSLIP	compressed SLIP
CSMA/CD	carrier sense, multiple access, collision detection
CSMA/CA	carrier sense, multiple access, collision avoidance
CTS	clear to send
DDOS	distributed denial of service
DF	don’t fragment
DMT	discrete multi tone
DNS	Domain Name System
DSL	digital subscriber line
DSSS	Direct Sequence Spread Spectrum
DWDM	dense wave division multiplexing
ECN	explicit congestion notification
ECT	ECN-capable transport
EGP	exterior gateway protocol
ESP	encapsulating security payload
ESS	Extended Service Set
FDDI	Fibre Distributed Data Interface
FHSS	Frequency Hopping Spread Spectrum
FQDN	fully qualified domain name
FTP	file transfer protocol
HTTP	hypertext transfer protocol
IANA	Internet Assigned Number Authority
IBSS	Independent Basic Service Set
ICANN	Internet Corporation for Assigned Names and Numbers
IEC	International Electrotechnical Commission
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IGP	interior gateway protocol
IGP	intradomain routing protocol
IKE	internet key exchange
IMP	interface message processor
IP	Internet Protocol

ISN	initial sequence number
ISO	International Standards Organisation
ISP	internet service provider
JANET	Joint Academic Network
LAN	local area network
LCP	link control protocol
LFN	long fat network
MAC	Media Access Control
MAN	metropolitan area network
MBONE	multicast backbone
MSL	maximum segment lifetime
MSS	maximum segment size
MTU	maximum transmission unit
NAT	network address translation
NCP	network control protocol
NFS	network file system
NIC	Network Information Centre
NSF	National Science Foundation
OFDM	orthogonal frequency division multiplexing
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
PAWS	protection against wrapped sequence numbers
PGP	Pretty Good Privacy
PPP	point-to-point protocol
RARP	reverse ARP
RED	Random Early Detection
RIPA	Regulation of Investigatory Powers Act
RIPE	Réseaux IP Européens
RIP	Routing Information Protocol
RPC	remote procedure call
RR	resource record
RTS	request to send
RTT	round trip time
SACK	selective acknowledgement
SLIP	serial line IP
SMTP	simple mail transfer protocol
SNA	Systems Network Architecture
SSL	secure socket layer
TCP	Transmission Control Protocol
TLD	domain!top level
TLD	top level domain
TLS	transport layer!security
TOS	type of service
T/TCP	TCP for transactions
TTL	time to live
UDP	User Datagram Protocol
UKERNA	United Kingdom Education and Research Networking Association
USB	Universal Serial Bus
UTP	unshielded twisted pair
WAN	wide area network
WAP	Wireless Application Protocol
WEP	Wired Equivalent Privacy
WML	wireless markup language

WWW	World Wide Web
XDR	external data representation
XML	extensible markup language

1 Introduction

1.1 What is the course about?

A network is any means of connecting entities—usually computers—together so that they can communicate. The means of connection can be wire, or optical fibre, or radio, or satellite, or sound waves or whatever, but the general idea is that we have channels capable of transmitting information between entities. People use networks for many reasons.

- Resource sharing. The "traditional" reason for having a network is so I can use that big supercomputer 100 miles up the road. Or I can use the department's high quality colour printer.
- Collaboration. I can work with people on a different continent, sharing data and writing papers. This includes video conferencing and email.
- Information gathering. If I need information about the latest developments in CPU design, I can look through the Web or USENET.
- Reliability through replication. If my super valuable database is replicated on another machine, and if my machine crashes, then the data is safe. Note this is also a protection against malicious attack.
- Entertainment. From static content such as today's newspapers or video on demand, to interactive applications like multi-player games or user participation quiz shows.

And much more, of course.

A network can be big or small: from a single piece of wire connecting two machines to the entirety of the Internet. And whenever you have more than one entity—be it computer or person—you have all the usual problems of communication: are they mutually comprehensible? Do they share a common world view? Is their means of communication efficient, or even suitable for the purpose?

"Networks" is a huge subject. There is masses of intricate detail, some of which is very subtle and hard to understand. On the other hand the rewards of understanding even a small part of the subject can be substantial, both intellectually and financially.

Networks are big money at the moment—just look at the `.com` frenzy—but most people don't realize they have been around for a long time in many guises. Mention "networks" and most only think of the Internet.

- The telephone system. Ancient technology, and a huge investment of money in systems and burying copper wire. Major problem to solve is how to make a connection from A to B. Now caught up in the Internet boom and modernising rapidly, with much investment in optical fibre.
- The mobile phone system. Newer and still developing (the next generation of mobile phones is about to arrive). Investment in transmitter stations and radio wavelengths. Now A and B are moving about.
- TV and radio. A one-many system, mostly. Investment in content, transmitters and relayers (e.g., satellites).
- Cable networks. TV again, but also telephone and data.

- Data Networks. Private company nets. Dial-up systems. Each its own protocol, both hardware (voltages, number of wires, etc.) and proprietary software. Examples: DECNet, Microsoft, Novell IPX, AppleTalk.
- The Internet. Often confused with the World Wide Web which is just one thing that the Internet serves. Actually email has been most important in the development of the Internet. Also serves file transfer, remote access, conference video, etc. the "Internet" is actually a collection of smaller networks all connected together using a widely agreed protocol: the Internet Protocol (IP). The smaller networks are owned by companies or governments or individuals, and may be themselves composed of smaller networks. There is a strong hierarchical shape to the Internet, but there is no-one in overall charge. Each group owns its own part of the Internet, and they all agree on how to connect to the other parts: the Internet is a great collaborative effort. This is in contrast to the above proprietary systems.

  The success of the Internet over private, proprietary systems is due to the Internet being public, open, and that it uses standards from the hardware level on up.

  There are technical groups to oversee the growth and development of the Internet, but these are generally non-profit.

Networks are often classified by size.

- LAN. Local Area Network. A network in a building or organisation. Requirements for speed and responsiveness.
- MAN. Metropolitan Area Network. A city-wide network. Problems of who pays for what. Bath is connected to the Bristol and West of England MAN (BWE MAN).
- WAN. Wide Area Network. Long haul, e.g., country-wide. Problems of long trip delays, protocol conversions. The UK academic community uses the *Joint Academic Network*, or JANET.

There is much overlap between these, but different technologies can be targeted at the problems of a particular size of network.

This course will concentrate mostly on the Internet, and in particular the protocols that are used on it.

Some links are available at `http://www.bath.ac.uk/~masrjb/CourseNotes/math0078.html`

Occasionally there are snippets of text like this one. These are bits and pieces that are not part of the main thrust of the text, or things that may only make sense later. Ignore at the first reading, if you wish.

1.2 Books

Primary: Stevens "TCP/IP Illustrated, Volume 1". There are many other books about, though beware of the "IP for Windows" kind of books. They just tell you what buttons to press in which windows, and give no understanding of what's really happening.

Background reading: Tanenbaum "Computer Networks" (Third Edition).

Due to the rapid change in Internet technology both these books are a trifle out of date in places: but the majority of the content is still absolutely relevant.

1.3 How big is 1MB?

There are several ways to measure things in the computer world, and some people use the same measures to mean different things.

For example, when describing memory, 1MB generally means 1 megabyte, which is $2^{20} = 1048576$ bytes. On the other hand, disk manufacturers usually use 1MB to mean $10^6 = 1000000$ bytes. Thus you can't fit a megabyte of memory on a megabyte disk! And worse, sometimes the two systems are mixed: the 1.44MB floppy disk uses a megabyte of 1024000 bytes.

To try to disambiguate the confusion, there is an official International Electrotechnical Commission (IEC) standard that defines a megabyte as definitely 10^6 bytes, and introduces a new unit, the *mebibyte*, that is definitely 2^{20} bytes. This takes the first two letters of the existing name and adds "bi" for binary. Unfortunately, not many people are yet aware of this system.

Traditional name	K kilo	M mega	G giga	T tera	P peta	E exa
binary	$2^{10} = 1024$	$2^{20} \approx 1.04 \times 10^6$	$2^{30} \approx 1.07 \times 10^9$	$2^{40} \approx 1.10 \times 10^{12}$	$2^{50} \approx 1.12 \times 10^{15}$	$2^{60} \approx 1.15 \times 10^{18}$
decimal	1000	10^6	10^9	10^{12}	10^{15}	10^{18}
IEC name	Ki kibi	Mi mebi	Gi gibi	Ti tebi	Pi pebi	Ei exbi
	2^{10}	2^{20}	2^{30}	2^{40}	2^{50}	2^{60}

We shall be using the traditional binary

b	bit	B	byte
M	mega	G	giga
K	kilo	s	per second

so that 10Mb means 10 megabits, and 10KB/s means 10 kilobytes per second, though sometimes when talking about data rates we shall be lazy and use Mb to mean Mb/s. For example, "10Mb Ethernet" should be "10Mb/s Ethernet," but the former is common usage.

2 History

This is a brief overview of the Internet only.

2.1 Past

This is a *very* sketchy history of the Internet. Much is omitted and much is simplified.

In the mid 1960s the Advanced Research Projects Agency (ARPA) wanted a system to allow researchers to use each other's (expensive) computers. Designed to be non-centralised to avoid single points of failure, specifically nuclear attacks (this was the height of the Cold War).

Simple telephone links were too vulnerable, as chopping one would split the network. They moved to the idea of *packet switched* networks.

The telephone system is based on *circuit switching*. This means that their objective is to provide an (electrical) circuit from A to B over which the conversation will be carried. This is like reserving the whole of the east coast railway line

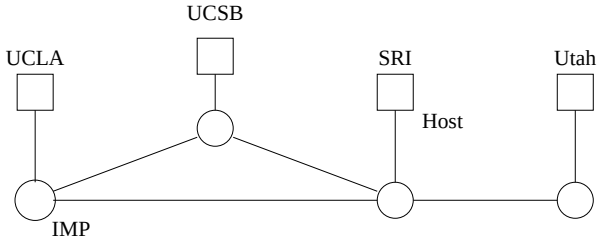


Figure 1: The original ARPANET

to allow a single train to go from London to Edinburgh. A second train cannot use the line until the first has reached its destination.

The alternative is *packet!switching*. The train is broken up into carriages, and each is sent singly down the track. The big advantage is that several trains can share the same line: the carriages can be interleaved. Furthermore separate carriages can actually take different routes, as long as we reassemble them in the correct order at the destination. This gives us better use of the track bandwidth, and resilience against leaves on the line.

In terms of data, packet switching is just this: chop the data up into manageable chunks or *packets*, and route each packet individually. Compare this with circuit switching, where a dedicated line is set up for the transaction. We shall compare the pros and cons later.

The first ARPA net consisted of Interface Message Processors (IMPs) connected by transmission lines. These were multiply connected together in a redundant fashion for reliability. The IMPs used *store and forward*, that is they read an entire packet into their memory before sending it on. These were 24KB minicomputers connected by 56Kb telephone lines.

Note that, as is still true today, it was common for the Internet to use the existing telephone system to carry the signals.

In 1969 the network went live with 4 nodes: Stanford Research Institute, UCLA, UC Santa Barbara, and the University of Utah. They specifically connected incompatible host computers to demonstrate the machine independence of their system. The protocol the network used was called Network Control Protocol (NCP). Very soon it was found that remote access of computers was not the main use of the system, but email and discussion groups. The social side of the Internet was starting to be recognised.

By the end of 1972 there were 30 or so hosts connected across the width of the USA. In 1973 University College London joined up. The protocols the network used were under continuous development, and by 1974 the Transmission Control Protocol/Internet Protocol (TCP/IP) emerged to replace NCP. As the operating system of choice at that time (Unix) had TCP/IP built in, it was easy for Universities to join the ARPANET.

Any many did. 1979 saw the advent of USENET news: a logical progression from telephone dial-up bulletin boards and the discussion groups.

By the early 1980s there were hundreds of machines connected. It was becoming a little difficult to manage all the names for the machines so new protocols were developed to collect machines into domains and have a non-centralised method of naming. This was the Domain Naming System (DNS): the `.com` was born. In 1982 the word “internet” was first used to describe a network of networks.

In the mid 1980s a high speed successor to ARPANET was developed. The National Science Foundation (NSF) created the NFSNET *backbone* set up between the six NSF supercomputer sites and this provided major trunking between regional networks. This started at 56Kb telephone lines, but was soon upgraded to 448Kb fibre optic lines, and then 1.5Mb lines in 1990. By the end of the ’80s, there are hundreds of thousands of hosts on the Internet.

In 1989/1990, the ARPANET is decommissioned.

Big business started to be interested, and they provided commercial IP networks, and the backbone was replaced by a collection of commercially driven infrastructure.

This growth was fuelled by the uses people made of the networks. Mostly email, but other things, too.

Use of a general protocol to connect machines. Non-proprietary and open so anyone could adopt it and implement it. Many other standards, e.g., OSF in the UK, IBM's mainframe network, BITNET, HEPNET (high energy physics), SPAN (NASA), and so on. But the only protocol allowed on the Internet is IP, and this ensured that an IBM machine could talk to a DEC machine regardless of their internal workings.

Continued growth. General use in Universities and a few companies, mainly email. Ethernet at 10Mb as a general local network connect. The decline of the other protocols: everybody starts using the IP in their systems in preference to their own or bought in protocols.

Invention of Gopher in 1991. The University of Minnesota invented a system to simplify the fetching of files with the "go for" system. This presents the user with a list of files and directories, and these can be links to other gopher systems anywhere else in the world. Gopher was popular for a while being text based, and thus suitable for the majority of terminals in use at the time. (Gopher is still supported in the major Web browsers.)

Invention of the World Wide Web (WWW) in 1991. Tim Berners-Lee at CERN (European centre for nuclear research) needed a way to control their huge amounts of data (report, pictures, programs, etc.) that were spread across the many participating countries. He invented the World Wide Web. Similar to gopher, but with a graphical point-and-click interface and the ability to display pictures (and later, sound and video). He and Marc Andreessen developed the Mosaic browser (1993), later to become Netscape.

This was a big breakthrough: point and click interfaces allows use by non technical people.

Sudden massive growth as the Internet is recognised to have commercial value for delivering content via the WWW, and the general public at home can use browsers to access it via modems. After several false starts (they tried to market their own proprietary system) Microsoft falls into line and the Internet takes off.

Huge growth in Internet Service Providers (ISPs), companies that connect you to the Internet, e.g., AOL) and companies selling over the WWW: billions of dollars are spent on and over the Internet. Growth in infrastructure involving advances in optical fibre technology and processor power.

In the UK "free" ISPs arrive and these boost the growth here.

Internet companies go public and reap billions. Entertainment companies (generally TV, film, and publishing) start taking an interest.

2.2 Present

The PC revolution allows the general public great computing power. Network cards and modems cheap through economies of scale. Small businesses can set up networks very easily, and huge numbers of commercial ISPs exist to connect them to the Internet.

More bandwidth. "Fast" modems at 56Kb. Integrated Services Digital Network (ISDN) at 128Kb. Fast Ethernet at 100Mb. Asynchronous Transfer Mode (ATM) networks at 155Mb and 622Mb (from the telecoms area: circuit based, but can run IP packets over it). Gigabit Ethernet at 1Gb, then 10Gb. Internet2.

Teleconferencing, Internet telephony, teleworking, telemedecine.

Continued growth. Continued business interest. Internet market in turmoil. Internet companies making a big splash on the money markets, but not actually making that much money, if any. Ecommerce. Companies still looking for the right approach. Company mergers: AOL and Time-Warner.

Big companies worried about intellectual property (e.g., DVD, MP3), and urge governments to pass laws (e.g., proposed copyright law in USA where you are bound by a contract you have not read).

Governments getting worried about something they don't control, particularly regarding taxes. Laws framed and passed, e.g., Australia, that don't appear to understand the way the Internet works. Questions of territoriality. The four horsemen of the Infocalypse: terrorists, paedophiles, money launderers, drug dealers.

Disputes over who owns what, e.g., domain names. Disputes over control of standards, e.g., Netscape vs. Microsoft for HTML, and Sun vs. Microsoft for Java.

Questions of security and authentication arise. Try to ensure security of, e.g., credit card numbers. Also: is this site I am buying from *really* who they say they are? Encryption is a difficult issue: governments want to read your email.

2.3 Future

Who knows? Here's some guesses.

More bandwidth. Terabit networks: fibre optic cables with hundreds of fibres each with thousands of colours each carrying gigabits. Broadband to the home: Asymmetric Digital Subscriber Line (ADSL) over existing copper wire (about 1.5Mb downlink, and perhaps 16Kb upstream, plus enough left for a normal telephone channel), and cable modems (500Kb to 2Mb) over fibre optic or coax.

New ways of supplying Internet connectivity: over radio; over the electricity grid; over TV and satellite. Use of Internet exceeds use of telephones.

The squeeze on IP addresses. The slow conversion to IPv6: enough addresses to have billions per square foot of the whole planet. Internet2 in Universities spreads.

More big business try to direct the way the Internet works. Some large companies try to enforce software patents to try to grab control of parts of the Internet, e.g., Amazon and "one-click shopping". Growth in Internet businesses like books, CDs, groceries. Virtual businesses: different fronts to the same company or unified fronts to several companies.

Governments get more worried about the empowerment of the individual: some group or individual does some terrorist act on or using the Internet, this is used as an excuse to try to pass restrictive laws (c.f., "the thieves used a getaway car, thus we should ban the use of cars").

Growth of Internet in Third World countries, e.g., Africa, and similarly in India, China, et al. There is a battle between the USA and the rest of the world over "cultural imperialism".

The entertainment industry tries to take over; delivery of TV and videos over the Internet. Interactive services. Telephony over the Internet (VoIP) becomes common. Banking over the Internet.

The Internet moves into smaller and smaller devices. Networked homes: connect the refrigerator (or coke machine) to the Internet. "Thin clients" where you have an Internet-enabled device that connects to a word processor on a chunky machine somewhere else. Mobile Internet: on your telephone, in your car.

Ubiquity. Convergence. TV equals computer equals games console equals telephone equals refrigerator.

3 The Seven Layer Model

Building a network is a very complicated problem. There are many things to be addressed:

- What hardware do we use? This includes things like the design of plugs and sockets.
- How do we encode bits on the hardware? What voltages, what speed.
- What standard of service do we wish to provide? Reliable, connectionless, stream oriented, packet switched. Flow control.

- What interface to the computer do we want? How do programmers actually use the network?
- What protocols should we to connect applications? E.g., the WWW.

The thing to note is that we have to have a standard all the way from the lowest part of the hardware right up to the highest level of the software if two random machines in the world are to communicate.

One way to approach this is to have one huge standard that fixes everything at every level. But this is not very flexible. Maybe we want to change the hardware: do we have to rewrite our browser to accommodate the new standard?

Fortunately, the problem nicely decomposes into several smaller problems, and in 1983 a *layered* standard was proposed. Or, to be more precise, a *reference model* was proposed, the International Standards Organisation (ISO) Open
OSI 7498 Systems Interconnection (OSI) Reference Model. This is commonly known as the *OSI Seven Layer Model*. It defines a way you should think about presenting a standard for a network definition. It doesn't actually give a standard for a network itself (though there was one directly based on it as a separate standard).

The principles involved were

- a layer should be created where a different level of abstraction is needed
- each layer should perform a well defined function
- the function of each layer should be chosen with an eye toward defining internationally standardised protocols
- the layer boundaries should be chosen to minimise the information flow across the interfaces
- the number of layers should be large enough that distinct functions need not be thrown together out necessity, and small enough that the architecture does not become unwieldy.

Here are the seven layers with their classical properties: note that not everyone sticks hard and fast to this kind of division of behaviours.

3.1 The Physical Layer

This is the hardware layer and deals with the transmission of bits over a channel. Typical problems are what voltages (or change of voltages) should be used to signify a 1 and a 0; how long should a bit be; how many wires to use in the cable; what each wire is for. This is an electrical and mechanical specification that transmits a continuous stream of bits. Note that this layer might be radio rather than copper wire.

3.2 The Data Link Layer

This layer takes the physical medium and decides how to use it to provide a channel where there are no undetected errors of transmission. Note that we might allow a physical layer that is prone to errors (e.g., radio) as long as we can detect those errors.

Typically this is achieved by breaking the input data into *data frames*, and transmitting each frame in sequence. A frame might be tens or thousands of bytes long. Perhaps *acknowledgement frames* are returned from the receiver. If a frame is corrupted (lost, damaged, or duplicated), the data link layer can retransmit.

Another problem that can be addressed at this layer is *flow control*. Perhaps the sender is pumping out data faster than the receiver can currently cope with: some means of telling the sender to slow down must be employed. Similarly, when the receiver has caught up, it can inform the sender to speed up again.

3.3 The Network Layer

This is concerned with controlling the operation of the subnet, including the question of how to route a packet from source to destination. This might include the problem of congestion control: if too many packets trying to use use one line we might reroute some, or use flow control to slow some sources down.

We can also deal with inter-network problems at this layer. Perhaps a packet is routed from one network to another that has a smaller frame size.

The network layer also deals with things like *accounting*: counting the number of bits sent by a user so we can bill them later.

3.4 The Transport Layer

This layer accepts data from the next, the session layer, and chops it into packets suitable for the network layer. Similarly, it receives packets from the network layer and reassembles them in the correct order for the session layer.

This layer can manage network connections, maybe sending one data stream out over several connections to improve throughput; or multiplexing several data streams over one connection to save money.

This layer provides the type of service available to the user: examples are reliable (error-free), order preserving, connection oriented (it appears that we have a direct line to the destination, cf. a telephone call), connectionless (each packet is treated individually).

3.5 The Session Layer

This allows the user to create a *session* between the source and destination. One example is a remote login session: you make the session by using `telnet` or whatever, and this session persists until you logout, when the session is taken down. Sometimes a session can be very short, e.g., just long enough for an email or Web page to be transmitted.

This layer takes care of things like synchronisation: if you have a large file to transmit that takes 2 hours, and the network or the remote machine crashes after 1 hour, the session layer can reestablish the connection at the point it left off rather than starting again. The session persists while the transport can disappear for a while.

3.6 The Presentation Layer

The presentation layer is getting very close to the end user. It provides things that are commonly needed so we don't have to reimplement them in every application. This includes stuff like standard encodings for letters (ASCII), integers (two complement big endian), and floating point (IEEE), so that machines at either end can agree on what this stream of bits actually means.

3.7 The Application Layer

This is the top layer in this model. It contains the protocols that end users applications need, like `telnet`, SMTP for email, http for the WWW, and so on.

Beyond the application layer are the programs that the user sees: a browser that uses `http` or a emailer that uses SMTP to send email.

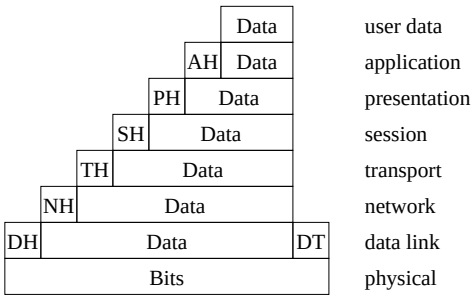


Figure 2: OSI encapsulation

3.8 How the layers fit together

In a pure implementation of the model each layer only has contact with the layers immediately above and below it.

Each layer is passed to the next via *encapsulation*. This is just wrapping up the data in a way that the layer below can cope with it. This starts when the user data is passed to the application layer. This might add some stuff, e.g., a standard email header.

This is passed to the presentation layer. As far as this layer is concerned, it just gets a bunch of bits. It doesn't (or shouldn't) know that the first few bits are an application header. This layer may transform the data in some way (e.g., convert numbers to a particular format), and may prepend a header that contains useful information for the person who eventually comes to unpack the data.

And so on down through the model. Each layer may perform any transform on the data and may prepend headers. Or a layer may do nothing at all, and have a null header. It all depends what you need to do for the job in hand.

Often the data link layer has a header and a trailer: this is to separate out the packets on the wire.

At the other end the receiving stack unwraps and untransforms each layer appropriately. (Sometimes the untransform is not successful: e.g., between different character sets in the presentation layer.)

3.9 Why Layers and Encapsulation?

The use of encapsulation seems wasteful: if the original data are small, then the packet on the wire could be mostly headers from the various layers. This is overhead that reduces the effective bandwidth of the transmission. Surely it is better to just put the data directly into the link layer?

The idea of using layers is for flexibility. Suppose we have a 10Mb hardware card in our machine, and someone comes up with an improved 100Mb card. Because the physical layer is (almost) totally separate from the datalink layer, we can just write a new standard for a 100Mb physical layer and slot it in where the old one used to be. The upper layers do not even need to know the hardware has changed. This is why we need to separate functionality carefully: the network layer and above should certainly know nothing about what hardware you are using.

In fact, this has happened several times: the Internet runs over (amongst many others), 10Mb Ethernet, 100Mb Ethernet, 1Gb Ethernet, 10Gb Ethernet, telephone lines (SLIP and PPP), radio. The user at their terminal has no idea of what is going on beneath them.

RFC1149 In principle you could use carrier pigeons as the physical layer and your browser should work unchanged, apart from a slow-down, maybe.

Someone did actually implement this RFC, with real carrier pigeons!

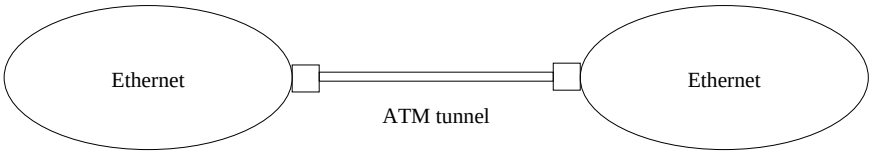


Figure 3: Tunnelling

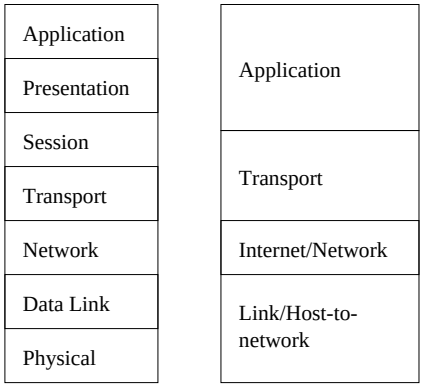


Figure 4: OSI vs. TCP/IP

Indeed, encapsulation may not stop even at this, the physical layer. For example, there are physical limits on the size of an Ethernet (speed of light problems!), so how can we connect up an Ethernet that spans the Atlantic? One way we might do this is to *tunnel* the Ethernet traffic inside some other kind of network, ATM or SMDS, for example. These protocols can work over long distances.

We simply stuff an Ethernet packet into the ATM network, and it pops out the other end to continue in its Ethernet world. In practice, things are more complicated, of course, and we tend to tunnel at the network layer level.

Tanenbaum uses this analogy: suppose you wish to drive your car from London to Paris. There is no road across the Channel, but there is the Shuttle. Your car moves under its own power on the road network from London to Dover, but then your car is encapsulated in the train carriage for the journey under the Channel. At the other end, the car moves under its own power again on the roads to Paris.

Someone once wrote software to tunnel TCP/IP over email. This allowed TCP connections through a firewall—but very slowly!

RFC3093 RFC1149

There is also a standard for tunnelling TCP/IP over HTTP, and, of course, RFC1149 for IP over avian carriers.

---

4 The Internet Model

The OSI model was very successful at getting people to concentrate on the specifics of a network implementation. However implementations based directly on it were not popular, principally since they were complex and quite slow. By sticking rigidly to the layers and following the principle of insulation between the layers it is difficult to get any real speed from an implementation.

The *TCP/IP Reference Model*, also called the *Internet Reference Model* was developed with the principles of the internet in mind: resilience to damage, and flexibility of application.

This is a four layer model.

4.1 The Link Layer

Also known as the *host-to-network* layer, *data link* layer, or *network access* layer.

This covers both the hardware of the OSI physical layer and the software in the OSI data link layer. The TCP/IP model does not say much about this layer as it recognises that there can be many different types of hardware to send your packets across. This layer only has to be capable of sending and receiving IP packets.

4.2 The Network Layer

Also known as the *Internet* layer.

This handles the movement of packets about the network, including routing. This layer defines a specific packet format and a protocol, the Internet Protocol (IP) to manipulate those packets.

4.3 The Transport Layer

Also known as the *host-to-host* layer.

This is analogous to the OSI transport layer. It provides for a flow of data between source and destination. Two protocols are defined at this level, TCP, and UDP.

The Transmission Control Protocol (TCP) is a reliable connection oriented protocol that delivers a stream of bytes from source to destination. It chops the incoming byte stream into packets and sends them to the Internet layer. It copes with acknowledgement packets, and resends packets if it thinks they have been lost. Going the other way, it receives packets and reassembles them into a continuous byte stream, sending acknowledgements for successfully received packets. Flow control is also handled here.

The User Datagram Protocol (UDP) is an unreliable, connectionless protocol for those cases where you do not want or need TCP's overhead. Used for situations where fast delivery is preferred to accurate delivery, e.g., sound or video.

The world "unreliable" is being used in a technical sense here as meaning "not guaranteed reliable". Typical unreliable networks are actually pretty reliable these days.

Theoretically, TCP and UDP do not have to be layered on top of IP, but it is unheard of not to do so.

4.4 The Application Layer

This model does not have session or application layers, as they were perceived as not necessary in this context.

The application layer provides the top-level layer protocols like SMTP, FTP and telnet.

Applications have to cope with presentation issues themselves, e.g., by using libraries like XDR to convert data to a machine-independent form. Though if you stick to the ASCII character set, there is no real problem. Occasional glitches do occur, such as Microsoft tool generated web pages that contain ? instead of '. This is due to Microsoft not following accepted standards.

The Internet model is somewhat more flexible than the OSI. Applications can (in rare cases) use the network layer directly (IP and ICMP) rather than going through TCP or UDP. This appears to contradict the point of using layers, but a) it is convenient, and b) since we are talking about IP we already know what the lower layers look like, and they are unlikely to change often. We will have to pay the price if there *is* a change, e.g., to IPv6. For the overwhelming majority of cases applications do use TCP or UDP.

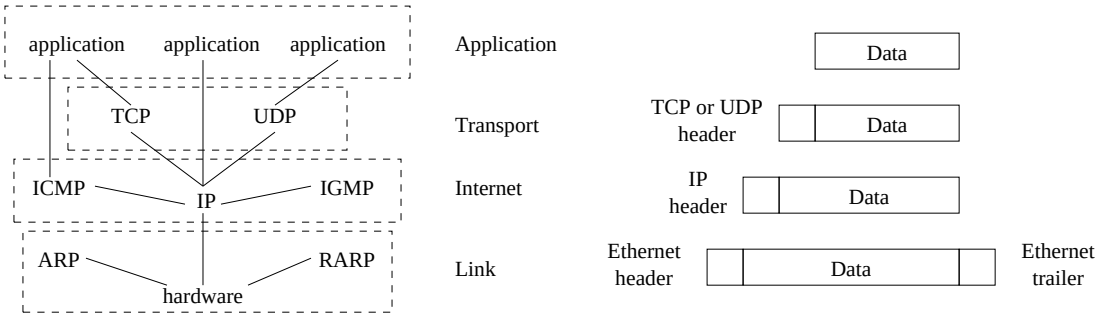


Figure 5: Internet Protocol

4.5 Models and Implementations

It is easy to confuse the OSI and Internet *models* with the OSI and Internet *protocols*. A model is a set of guidelines on how one should go about designing a network protocol. For example, it can say "use a physical layer which will deal with voltages, frequencies, etc." The model does *not* say "use copper wire, and voltages of 5V representing a 1 bit." That is a specific protocol implementation.

A model can have many implementations that fit it. For example consider the following network: two plastic cups joined by a piece of string. The physical layer is the cups and string; the network layer is empty; the transport layer is saying "over" at the end of each voice packet; the application layer is whatever we are talking about. This is a network implementation that fits the Internet Model.

4.6 Comparing OSI and Internet Models

There is a rough correspondence between the two models, apart from the missing and merged layers. Though there are big differences.

The OSI model was developed *before* an implementation, whereas the Internet model was developed *after* TCP/IP was implemented, and is rather a description of what happened. OSI makes a clear distinction between the model and and implementation, Internet is more fuzzy.

OSI is very general, whereas Internet is very specific. OSI is more flexible in that is it not tied to a specific protocol, and is better able to adapt to changes in technology. On the other hand, the OSI model had many problems when it came to an implementation when it was found that the layers provided did not correspond well to reality. Extra sublayers were developed, and the simplicity of the OSI model was lost.

As it turns out TCP/IP has been widely successful, while the OSI is relegated to books on networking. Many reasons have been given.

- Bad timing. The OSI people took too long to publish, and by the time they got there, TCP/IP was already widely used.
- Bad technology. The design of the layers is flawed, and does not work well in reality. Seven is not a magic number, and other proposals had more layers (splitting up several layers into smaller, easier ones), or fewer, e.g., the Internet model. It appears that seven was chosen as IBM already had a seven layer protocol (Systems Network Architecture, SNA).
- Bad implementations. The OSI standard was hard to follow, and only poor implementations were made. They were complex and slow, and OSI got the image of poor quality even though implementations improved later.

  TCP/IP, though, was free and fast.

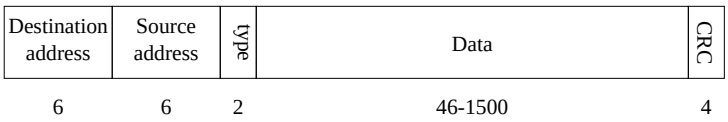


Figure 6: Ethernet Frame

- Bad politics. The Internet model was developed in academia as part of Unix. OSI was (viewed as being) developed by governmental bureaucrats across the world.

The TCP/IP model is not all-singing all-dancing either, it does have problems. The specification is confused with the implementation; it is only good for describing TCP/IP and no other protocol stack; the physical and data link layers are merged, making it hard to talk about (say) copper wire vs. fibre.

The OSI model is widely used; the OSI protocols are virtually never used. The Internet model is virtually never used; the Internet protocols are extremely widespread. A compromise is used by Tanenbaum: split the link layer of the Internet model into a physical and data link layer:

1. application
2. transport
3. network
4. data link
5. physical

This appears to be a good compromise in that it matches well with reality (i.e., TCP/IP) and is somewhat more flexible than the Internet model.

5 The Link Layer

We shall now look at each layer in turn, starting at the bottom: the link layer, including the physical layer. The link layer carries IP packets, and *Address Resolution Protocol* (ARP) packets. ARP is considered as part of the link layer, while IP is above in the Internet layer.

There are several popular link layers, including Ethernet, Token Ring, Fibre Distributed Data Interface (FDDI), and SLIP and PPP for serial interfaces (e.g., dial-up modems). Of these, Ethernet and (lately) PPP are the most popular and are widely deployed.

5.1 Ethernet

RFC894

The Ethernet standard was made in 1982 by DEC, Intel and Xerox. It uses a method called *carrier sense, multiple access with collision detection*, or CSMA/CD, and runs at 10Mb. Hosts have 48 bit addresses.

RFC1042
IEEE 802.3

A couple of years later the IEEE published another standard, 802.3, which is almost but not quite the same as Ethernet. It is sufficiently different that they do not interoperate, though you can have packets from both standards on the same wire without interference. Ethernet is by far more popular.

This being the link layer, we must define how bits are laid out on the wire.

An Ethernet *frame* (aka packet) starts with two 6 byte fields containing 48 bit *hardware addresses* (also known as a Media Access Control or MAC address). First destination, then source. Every Ethernet chip in the world has its own

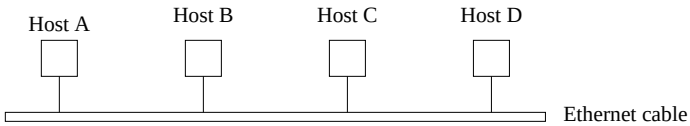


Figure 7: Ethernet

unique hardware address burned in at manufacture time. For example, a typical address could be 0:20:48:40:2e:4d, written as a sequence of 6 hex numbers.

The top 22 bits of this 48 bit address identify the vendor of the Ethernet chip, while 24 bits is a serial number set by the vendor. One bit is used to indicate a broadcast (or multicast) address, and the last bit is used to indicate a "locally administered address", where the address has been reassigned to fit some local policy.

Next is the 2 byte *type* field. This is a number that indicates what kind of data follows: (hex) 0800 indicates an IP
RFC1700 packet, while 0806 indicates an ARP packet. These numbers are defined in RFC1700 et seq. This allow the system to pass the data quickly to the relevant program.

Then comes the actual data. This can be up to 1500 bytes. Curiously, there is also a minimum size of 46 bytes. The reason for this will be explained shortly. If the data section would be less than 46 bytes, it is padded out with zero bytes. Somehow the data field must encode how long the real data part is itself.

Finally there is a 4 byte *checksum* (aka *cyclic redundancy check*) (CRC). This is a simple function of all the bytes in the frame, and is intended to catch hop-by-hop errors in transmission. It computed by the source just before sending the frame. On receipt, the destination recomputes the checksum on what it got. If an error occurs in the transmission of the frame this should show up as a difference in the values of the received and computed values of the checksum. If this should happen the packet is assumed corrupted, and the frame is dropped. It is up to a higher layer to realise this has happened and to take corrective action.

One Ethernet address is special: all ones, or ff:ff:ff:ff:ff:ff. This is the broadcast address: the packet goes to all machines on the (local) network. One or more machines may choose to reply. This is useful when bootstrapping other parts of the IP.

5.2 CSMA/CD

IEEE 802.3

The limitations on packet size are imposed on Ethernet because of physical considerations of the hardware. Ethernet is a shared medium (*multiple access*), that is several machines are connected by a single piece of wire that the bits flow through.

If host A wishes to send a frame to host B, but C is already sending to D, then A must wait, since the cable is already occupied. Thus A listens on the wire (*carrier sense*) until C has finished, and then attempts to send. However, B, say, might be doing the same to send to D, say. So A and B will start sending at the same time. Both A and B hear the frames clash (*collision detection*), and both stop transmitting immediately. They each wait for a small random time, and then start listening again. As the delay is random, one of the two will start before the other, thus resolving the clash.

This is why a packet has a minimum size: the speed of light in the wire. A packet has to be sufficiently large that all hosts on the network can see it before they decide to transmit themselves. If small packets are allowed, there can be collisions *after* a packet is transmitted on a seemingly free wire.

The minimum frame size is 64 bytes (46 bytes plus 18 bytes headers). At 10Mb this is about 50μsec.

Signals propagate in copper at around 200m/μsec, so 64 bytes is about 10km.

This method is pretty good for low usage of the shared medium, but performs poorly for high usage due to many collisions: more time is spent backing off from collisions than sending packets. If a host has to *back off* more than 16 times, the packet is simply dropped. The network is so heavily loaded that there is no point in trying any more. If collisions hit less than 5% of packets, the Ethernet is considered to be working normally. If more than 50%, though, you should consider breaking the network up into smaller pieces.

5.3 Ethernet Hardware

The Ethernet standard includes the kind of cables needed. Originally, a thick coaxial cable was specified, but over the years there have been many updates.

A selection of Ethernet cable types.

Name	Cable	Max segment	
10Base5	thick coax	500m	“thicknet”, chunky yellow, vampire taps
10Base2	thin coax	200m	“cheapernet” “thin ethernet”, “thinnet”, actually 185m
10BaseT	twisted pair	100m	telephone wire, Cat 3 cable. 150m on Cat 5
10BaseF	fibre optic	2000m	long distance
100BaseT4	twisted pair	100m	100Mb Ethernet on two telephone cables (Cat 3)
100BaseTX	twisted pair	100m	100Mb Ethernet, Cat 5 cable, aka 100BaseT, “fast Ethernet”
100BaseFX	fibre optic	2000m	100Mb Ethernet
1000BaseT	twisted pair	100m	Gigabit Ethernet, Cat 5 or Enhanced Cat 5

There are several others, including fibre Gigabit.

The most important aspects are the increase of speeds in the standards. The 100Mb and 1000Mb (or Gigabit) Ethernet standards are more-or-less the same as the original standard (in terms of frames layout, CSMA/CD, etc.) but simply faster. There are one or two wrinkles, such as minimum frame size, and the amount of time you listen for a collision, but mostly it just follows through.

10Mb is very common, 100Mb is common, 1000Mb is just starting to appear, and 10Gb is on the experimenter’s workbench.

10Base5, the original Ethernet, runs on a fat coaxial cable with *vampire taps* containing a *transceiver*. There is a minimum gap of 2.5m between taps. The transceiver contains electronics to do the collision detection. This supports a cable (up to 50m long) to the host containing 5 twisted pairs, with a large and unwieldy socket (AUI socket) on the end. 10Base2 and 10BaseT are much simpler.

A segment can be extended with up to a maximum of 4 *repeaters* (which simply amplify the electrical signal), therefore 5 segments (total length of 2460m) can be connected together. Of the 5 segments only 3 can have devices attached (100 per segment). A total of 300 devices can be attached on a thicknet broadcast domain. This is called the *5-4-3 rule*.

Thinnet, or 10Base2, uses simple coaxial cable, much like TV cable. It plugs in using simple passive BNC (*British Naval Connector*) connector. Much cheaper and easier to lay around the building and to connect up. The clever electronics is now on the card in the machine. Minimum gap between connections is 0.5m, and a segment can have up to 30 machines.

A segment can be extended with other segments using up to 4 repeaters, i.e. 5 segments in total. Two of these segments however, cannot be tapped, they can only be used for extending the length of the broadcast domain (to 925m). What this means is that 3 segments with a maximum of 30 stations on each can give you 90 devices on a thinnet broadcast domain.

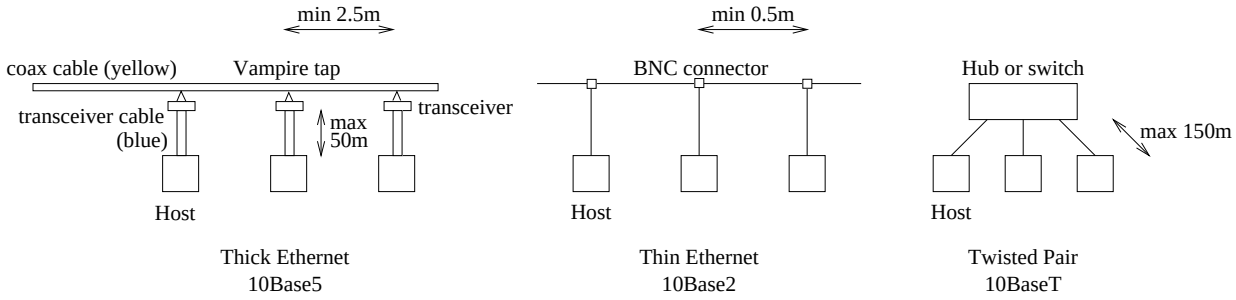


Figure 8: Wiring Ethernets

Twisted pair connects in a very different way. Sometimes known as *unshielded twisted pair* (UTP), this is literally a bunch (four pairs) of twisted wires. The twists help reduce electrical interference. This cable is much like telephone cable. On the ends are RJ45 plugs.

- Category 1: No performance criteria
- Category 2: Rated to 1 MHz (used for telephone wiring)
- Category 3: Rated to 16 MHz (used for Ethernet 10Base-T)
- Category 4: Rated to 20 MHz (used for Token-Ring, 10Base-T)
- Category 5: Rated to 100 MHz (used for 100Base-T, 10Base-T)

Enhanced Category 5 is Category 5 where you are really careful about making good clean connections in the plugs and sockets. Category 6 (250MHz) is just being ratified as a standard, and Category 7 (600MHz) is being discussed.

Instead of a single backbone cable that all the machines connect to, each machine is wired to a central box. This can be a *hub* or a *switch*.

A hub is ignorant of any protocol above the physical layer, and simply echos all packets on to all wires. This makes a hub connected system look electrically like a 10Base2 or 5 Ethernet. This is a single *collision domain* as a packet from any host has the potential to collide with packets from any other hosts, regardless of their destination. The collision domain shares the 10Mb or 100Mb available.

A switch understands a link layer protocol, e.g., Ethernet. It is able to direct a packet down that single wire that has the right machine on the end rather than copying it to all the output wires. This means now each output cable is a separate collision domain. There will only be a collision if two machines try to send to the same destination at the same instant: there will be no collision between machines sending packets to *different* machines at the same instant. Thus there is potentially 10/100Mb bandwidth available between each pair of machines simultaneously.

Generally switches are even more cunning than this: they *store and forward* packets. If it finds the output channel busy it can store a packet in internal memory, and forward it later when the channel is free. This means that each source-destination path is a separate collision domain. If this is happening we can do away with CSMA/CD completely, as there can be no collisions! If the output channel is exceptionally busy, the memory buffer can fill up and then the switch will just drop any more incoming packets (recall that Ethernet CSMA/CD will drop packets after 16 tries, so this is not so bad as it seems). It is up to a higher layer protocol to discover the loss.

Switches are more expensive, but give better bandwidth, and better efficiency under high load (no collisions).

Hubs and switches are also applicable to the other kinds of Ethernet.

If you have a switch, and your hardware supports it, twisted pair can run *full duplex*, meaning that there can be 10/100Mb *to* the machine simultaneously with 10/100Mb *from* the machine. A total of 20/200Mb in flight at a time. This is because there are no collisions between inward and outward traffic. The alternative is *half duplex*, where traffic

can only go one direction at a time. Typically, half duplex Ethernet uses CSMA/CD while full duplex is switched and does not.

You can buy boxes that connect together the different kinds of Ethernet cable, e.g., thinnet to twisted pair. Some network cards have more than one type of socket on them. Thicknet is virtually never seen these days, with thinnet still common and UTP being widely used, and Cat 5 (or better) UTP being used for all new installations.

If you need to connect just two machines, you can do this with UTP without a hub or switch. Just join then with a *crossed* cable: this has a pair of wires switched in the plug at one end. Much cheaper than buying a hub.

5.3.1 Faster and Faster

IEEE 802.3u Actually, 100Mb Ethernet only uses two of the four twisted pairs in a UTP cable: one pair for out and one pair for back. You are recommended not to use the other two pairs in a cable since the 100BaseTX standard does not take into account possible electrical interference from nearby twisted pairs.

IEEE 802.3z IEEE 802.3ab Gigabit Ethernet (802.3z) over copper (802.3ab) uses all four pairs with a different physical encoding of the bits (see section 5.4), and transmits and receives simultaneously on all pairs of wires.

Gigabit will run over standard Cat 5 if you are lucky, though you should use enhanced Cat 5 (or Cat 6 or more) to be sure it will support the more sophisticated encoding cleanly.

The minimum Ethernet frame size of 64 bytes is required in Ethernet for the CSMA/CD method to work. At Gigabit speeds using the original 10Mb Ethernet's minimum packet size would imply a maximum cable length of 10m: not very practical. A 512 byte limit would be more sensible. But Gigabit Ethernet does keep the 64 byte limit. Short packets are followed by a *carrier extension*, a special "hold the wire" signal to fill up to the end of a 512 byte slot. CSMA/CD regards this as part of the packet, and so holds off sending until we can be sure the wire is empty.

This means we might have 448 bytes of overhead on a small frame and so Gigabit Ethernet's performance on small packets is not much better than 100Mb Ethernet. The theoretical limit works out at about 86Mb/sec when sending 64 byte frames. As a partial fix for this, Gigabit Ethernet allows *packet bursting*, where several packets can be sent before relinquishing hold on the the wire. If a host has several packets to send, it sends the first one normally (possibly including carrier extension). If that gets through safely, the host still has control and can send more packets (without the need for carrier extension) up to a limit of 8192 bytes in total. It must then release the medium. This decreases the overhead if we want to send many small packets.

All this complication means that Gigabit Ethernet really ought to be used in full duplex mode on a switched network: then there is no need to worry about collisions and we can dispense with carrier extension and packet bursting. Notice, also, that a full duplex connection will have no inherent distance limit (apart from electrical considerations).

IEEE 802.3ae 10Gb Ethernet (expected by spring 2002) will use optical fibre, and is not currently expected to run over copper cable. It will be full duplex only as the standard Ethernet CSMA/CD does not scale this far. Just think on that: 10Gb Ethernet is 1000 times as fast as the original Ethernet standard over a timescale of less than 20 years!

Current (*dense wave division multiplexing*) (DWDM) fibre technology allows 10Gb per wavelength and about 160 wavelengths per fibre. A modest bundle of 10 fibres could carry 16Tb.

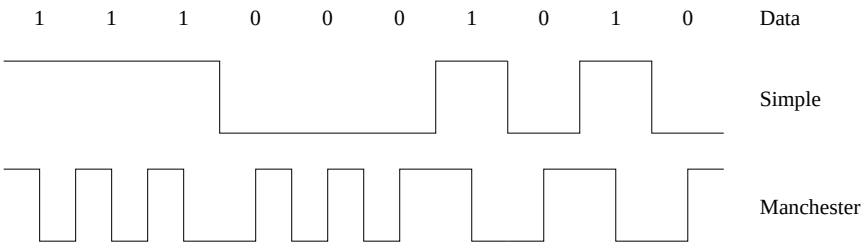


Figure 9: Manchester Encoding

5.4 Physical Encodings

We now look at the actual encodings of bits on the wire (or fibre). The simplest encoding would be to use 0V for a 0 and a positive voltage 1V, say, for a 1. This has a couple of problems:

- How can we distinguish between a free network (nobody transmitting) and somebody transmitting a stream of 0s? They look the same electrically.
- Sender and receiver need to be accurately synchronised to the starts and ends of bits, since with a long strings of 1s or 0s they could drift out of step. This means that, for example, the sender might send 1000 1s, but the receiver could think that it was only 999 1s.
- A long streams of 1s would be encoded as a steady 1V on the wire, namely a DC voltage. This is something that gives engineers great difficulties, as it is much easier to connect together equipment that has average voltage 0, e.g., an AC voltage.
- A similar problem happens with the lasers in an optical system: it is not too good to have continuous laser beams representing a stream of 1s.

So engineers prefer an encoding where the average number of 1s and 0s is about the same. Thus they use encodings of the data stream that approximate this even if the data is all 1s or all 0s.

Ethernet and 802.3 use a *Manchester encoding*. This chops the time interval for a bit into two halves: a 1 is represented by a high voltage in the first half, and a low voltage in the second. A 0 is represented by the reverse: a low voltage in the first half, and a high voltage in the second. The actual voltages used are $+0.85$V for high and -0.85V for low. On average, the voltage is 0. There is an extra advantage that this signal is easy to synchronise with: a transition through zero is always the middle of a bit.

Using 0.85V is also a compromise. Smaller voltages require less power, but are more prone to interference from noise from the surrounding electrical environment.

There is one major drawback. The bandwidth that the Manchester encoding requires is twice that of the simple encoding as the frequency of the signal is doubled. A 10Mb rate requires a 20MHz signal. This is not so bad, but it gets worse with 100Mb and higher rates.

Cat 5 cable is rated to 100MHz so we can't use Manchester encoding for 100Mb Ethernet. The encoding is more subtle. It uses a *4B/5B* system, meaning that 4 data bits are encoded into 5 physical bits. The encodings picked for the 16 values are such that each has at least two signal transitions which helps synchronisation, and they minimise DC current. A couple of 5B patterns are used for frame start and end, and one for "idle network" (again to maintain synchronisation).

So far this has only made it worse in terms of bandwidth. But now the encoding uses a three level (ternary) electrical encoding called *MLT-3*. MLT-3 is like Manchester in that it encodes 1 bits by transitions, but now the transitions are cyclically from 0 to 1, then 1 to 0, then 0 to -1, then -1 to 0, then 0 to 1, and so on. A 0 bit is encoded by no transition.

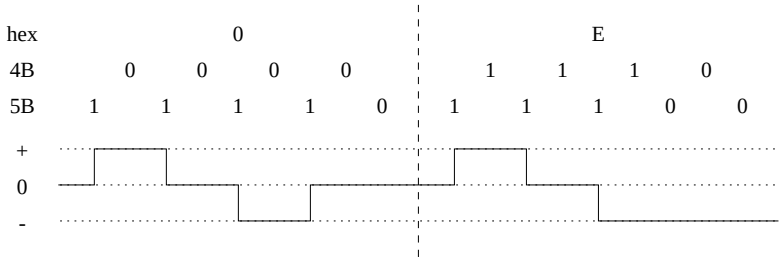


Figure 10: 4B/5b and MLT-3

An example, the byte value 15, or hexadecimal 0E. This is translated nibble by nibble by the 4B/5B encoding to binary 11110 and 11100.

Data Symbols		Control Symbols	
data	symbol	name	symbol
1111	11110	IDLE	11111
0001	01001	J	11000
0010	10100	K	10001
0011	10101	T	01101
0100	01010	R	00111
0101	01011	S	11001
0110	01110	QUIET	00000
0111	01111	HALT	00100
1000	10010		
1001	10011		
1010	10110		
1011	10111		
1100	11010		
1101	11011		
1110	11100		
1111	11101		

4B/5B Encoding

This runs at 31.25MHz giving a symbol rate of 125 MBaud. This is because the fastest electrical cycle happens with a stream of all 1 symbols (IDLE), and then we get a complete cycle every 4 transitions (0 to 1 to 0 to -1 to 0), giving us 4 symbols per cycle. Typically, though, we get frequencies of around 4/5 of this (25MHz) as data symbols have four 1 symbols or fewer.

A *baud* is the number of symbols per second, where a symbol is some chunk of information. A symbol might represent a bit, or 2 bits, or 2/3 of a bit, and so on.

100Mb Ethernet requires a symbol rate of 125 MBaud (efficiency 80%) due to 4B/5B encoding. Here one symbol encodes 0.8 bits.

The frequency a signal runs at has other implications, in particular the amount of electrical interference it produces. A system running at 25MHz is friendlier to the electrical environment than one running at 31.25MHz.

Gigabit Ethernet over copper uses a much more sophisticated 8B/10B encoding invented by IBM for Fibre Channel. It runs over 5 electrical levels (± 2, ± 1, 0) over all four pairs of wires in the cable in both directions simultaneously. The encoding, PAM-5, gives us 2 bits per symbol: 4 levels to encode 00, 01, 10 and 11, and one for error correction. This runs at 125 MBaud, as before, so we have $2\,\text{bits} \times 125\,\text{Mbaud} \times 4\,\text{pairs} = 1000\,\text{Mb/s}$ in both directions simultaneously.

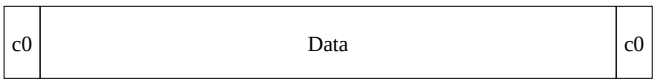


Figure 11: SLIP

Clever signal processing is required to disentangle the outward signal from the inward signal. The processors on a Gigabit Ethernet interface are about the complexity of a 486 processor.

As previously mentioned, 10Gb Ethernet is not expected to run over copper and will be exclusively optical. Optical systems only use binary encodings: either the laser is on or off. They don't have the option of multiple level encodings, so the encodings along fibres are different from those along copper.

5.5 SLIP and PPP

It is nice to have dedicated Ethernet to run our network, but not everybody has the right sort of cables running to their homes (or rather they might, but electrical limitations and absence of the right hardware at the end of the wire in the telecoms company prevents Ethernet being used). Thus a lot of people use modems to access the Internet through their telephone lines.

There are two common link layer standards for encapsulation of IP (and other) packets over telephone lines: SLIP and PPP. SLIP was common until PPP took over as the protocol of choice.

5.5.1 SLIP

RFC1055

Serial Line IP, or SLIP, is a simple encapsulation for IP over serial lines. It is a *point-to-point* protocol, meaning it is designed just to connect two machines together.

A frame is terminated by a byte containing the hex value c0, called END. Often, implementations will start the frame with c0, too, just to be sure.

This means we can't have bytes with the value c0 inside the frame as they would be interpreted as the end of a frame. To get around this SLIP uses byte stuffing. It we need to transmit a c0 it is replaced by two bytes db dc. The character db is called ESC. To send a value of db, replace this by two bytes db dd. This causes a minor expansion of data, but not much.

Thus the byte stream 00 c0 db 01 gets transmitted in a frame as

c0 00 db dc db dd 01 c0

When reading the frame at the destination it is a simple matter to replace any occurence of db dc by c0 and db dd by db to recover the original data.

There is no defined limit on the frame size, but it is suggested that you should be able to cope with frames at least 1006 bytes long, though many implementations use 296 bytes. This is because 1006 is too large a frame: at 9600bits/sec it would take about 1sec to transmit a full frame. Suppose we are copying a large file across the network: this means a continuous stream of full frames. If we want to have an interactive login session at the same time we will have to wait about 0.5sec on average for the current frame to finish before the packet containing our keystrokes can get through. Similarly for the response from the other end. An interactive response time of more than 100-200ms is felt by users to be too long. Reducing the frame to 296 bytes is a good compromise between interactive response and bulk data throughput. For modern machines with fast modems, 576 or more bytes is the size of choice.

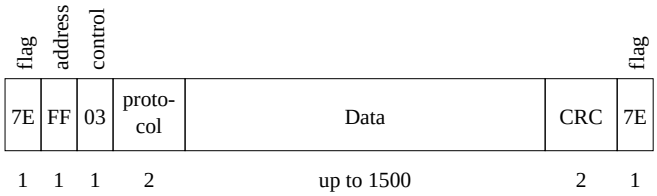


Figure 12: PPP

More importantly, the larger packet size reduces the fragmentation needed. Most DNS packets can fit inside 576 bytes.

SLIP works, but it has problems.

1. The ends must have pre-agreed on IP addresses: there is no mechanism for one end to tell the other its address.
2. No type field. SLIP only supports IP.
3. No checksum. Essential on noisy telephone lines.
4. No authentication. Needed on dial-up lines.

Telephone lines are slow (4KHz or 56Kb/s maximum), so any opportunities to improve throughput are welcome. TCP/IP has a large overhead of 40 bytes or more of header per packet (see later), and typically you send many packets with nearly identical headers from source to destination. A variant of SLIP, *compressed* SLIP (CSLIP) takes
RFC1144 advantage of this repetition. Instead of sending a full header, CSLIP sends the small differences, and this greatly improves throughput. Notice this is in complete disregard of layering, and ties CSLIP to TCP/IP.

5.5.2 PPP

The *Point-to-Point Protocol* was designed to fix the deficiencies of SLIP. There are three parts

1. a framing for packets on a serial link, including error correction
2. a *link control protocol* (LCP) for bringing up links, configuring them, and taking them down again
3. a set of *network control protocols* (NCPs) to negotiate network layer options that are specific to the network layer.

RFC1661 RFC1661 defines the encapsulation, and RFC1332 defines the NCP for IP.
RFC1332

A frame starts and ends with the byte 7e. The next two bytes are *address* (always ff) and *control* (always 03).

Next is the *protocol* field in 2 bytes. IP has value 0021. The NCP for IP has value 8021. Two bytes for a checksum of the frame, and 7e to end.

There are up to 1500 bytes in the data field, but this can be negotiated. As for SLIP, they must be escaped, using 7d.

- 7e $\rightarrow$ 7d 5e
- 7d $\rightarrow$ 7d 5d

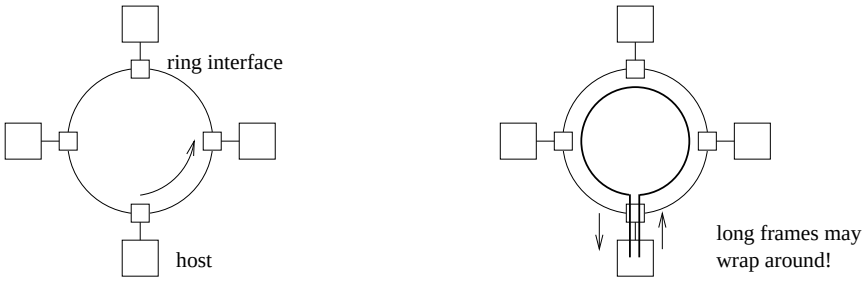


Figure 13: Token Ring

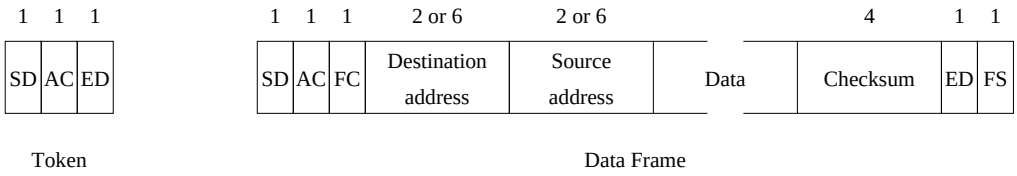


Figure 14: Token Ring

- x, where $x < 20$ (hex) $\rightarrow$ 7d $[x + 20]$. So 01 becomes 7d 21. This is done as some modems interpret values < 20 specially.

Connections can negotiate extra things via the NCP, like header compression (as in CSLIP). PPP is much better than SLIP, and should always be used in preference.

5.6 Token Ring

RFC1748
IEEE 802.5

Rather than have a shared segment of wire, we can have a circular network. These have been around for a long time (at least 1972) and are well-understood. They are really a collection of point-to-point links that happen to form a ring.

Packets circulate about the ring, being passed from one interface to the next until it reaches its destination. The ring is controlled by a special packet, called the *token*. When the ring is idle, this circulates. If a host wants to transmit a frame it must wait until the token arrives. It removes the token and replaces it with the packet, which continues round to the destination. The destination reads the packet. The packet continues around the ring until it reaches the source again, which removes the packet and replaces the token.

If many machines are waiting to send packets, the token will be grabbed in a round-robin style, giving natural bandwidth sharing.

Token Ring runs at 1 or 4Mb (and lately, 16Mb and more) over twisted pairs.

The token is 3 bytes long:

- Starting Delimiter. A special pattern of electrical signals that does not correspond to a data byte (an invalid Manchester encoding).
- Access Control. This byte contains the *token bit*, which is 0 for a token and 1 for a data frame.
- Ending Delimiter. Similar to the SD.

The data frame is arbitrarily long:

- Starting Delimiter.

- Access Control.
- Frame Control. This distinguishes a data frame from other kinds of frames which are used to control the network.
- Destination and Source addresses. Six byte addresses as for 802.3/Ethernet, but a variant allows two byte addresses.
- The data. This can be of any length up to a maximum time limit, which is generally 10ms, the *token-holding time*, which is 5000 bytes.
- A Checksum.
- Ending Delimiter.
- Frame Status. Contains various status bits, e.g., "destination present and frame read successfully". This information is is useful to the sender.

Packets also have *priorities*. The AC field contains a priority value, as does the token. That packet can only be sent if the token has priority not greater than the priority of the packet. Similarly, a host can raise the priority of the next token by raising the priority of the current data frame.

Rings have several advantages over Ethernet

1. good bounds for time to access the network: Ethernet is non-deterministic as it may have to repeatedly back off, and this is bad for real-time applications
2. smooth degradation under high load: Ethernet gets very inefficient at high loads
3. there is no practical limit on the size of the network
4. there is no minimum packet size, thus reducing overhead
5. point-to-point links are cheap and easy to build

On the other hand

1. a ring is difficult to extend: you have to take the entire network down to add a machine: Ethernet you just plug in
2. a broken interface is a big problem: Ethernet you probably wouldn't notice
3. there is a fixed large delay while you wait for the token: at low load the delay on Ethernet is virtually zero
4. the details of token ring are quite complicated, involving the priorities of packets, and means to monitor the liveness of the system (regenerating the token if it gets lost, etc.): Ethernet is relatively simple

On the whole, though, the two use roughly the same technology, and get roughly the same performance.

5.7 ATM

Asynchronous Transfer Mode is a relatively new link layer protocol that was designed to be fast and flexible. It is packet based, though packets are called *cells* in the ATM world and the technique is called *cell switching*. Cell delivery is not guaranteed, but order is. Thus, if packet 1 and 2 are sent in that order, then if both packets arrive, they will arrive in that order.

A cell is of fixed length: 53 bytes. Firstly, a fixed size was chosen as this makes hardware to manipulate ATM that much easier, and therefore faster. This length was chosen as a compromise between large packets for low overhead,

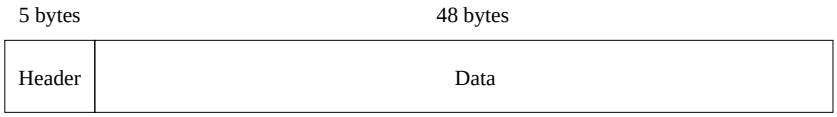


Figure 15: ATM Cell

and small packets for flexibility in carrying bursty traffic, like speech and video. Of the 53 bytes, 48 are data while 5 bytes are ATM header.

ATM runs at 155Mb and 655Mb, and is moving into the gigabit arena soon. It may be packet based, but it is connection-oriented. Making a connection involves setting up a path between source and destination. This same path is used for all the packets in this connection.

ATM makes big noises about *quality of service*. You can define certain levels of service, e.g., bandwidth or packet delay, and ATM can guarantee these levels will be maintained. Furthermore, different charges for the different levels of service can be made. ATM grew from the telecom organisations, so we can understand the preoccupation with connections, quality and charging.

ATM was designed to be used at all levels, from LAN to WAN, but only seems to be successful in the WAN sector. Some LANs run over ATM, but it is very expensive (cost of hardware), and 10Mb/100Mb Ethernet is dominant. This is because Ethernet is well understood and hardware and software are widely available, while the IP-to-ATM layer interface is quite complex and hard to implement.

On the other hand, ATM is very good long distance carrier, and it used extensively for large bandwidth long distance connections. For example, the link across the Atlantic used by JANET is ATM.

Standards for layering PPP over ATM are just starting to appear, which becomes more important when we start to consider ADSL.

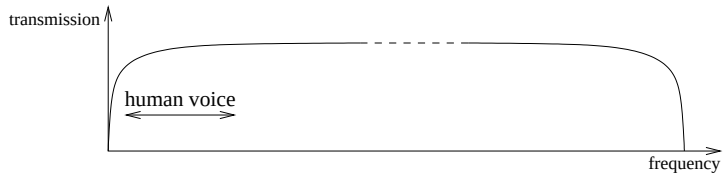
There are a few standards for how to split packets into ATM cells, the most useful being *ATM Adaption Layer 5* (AAL5). There is no header but a trailer instead. It pads the end of a packet to a length that is 47 mod 48 and a single byte information field (currently unused) finishes that cell. The next cell starts with a single byte to align the next two fields (recall the ATM header is 5 bytes long). These are a two byte length field and a 4 byte CRC. A bit in the ATM header indicates that this is the last cell for the packet, and the length field allows us to find the end of the packet within the previous cell. Notice that this relies on the in-order transmission of cells. The maximum payload is $2^{16} - 1$ bytes,
RFC2225 but RFC2225 recommends 9180 bytes as this matches the SMDS MTU (RFC1209).
RFC1209

5.8 ADSL

This is *Asymmetric Digital Subscriber Line*. We conclude with a brief encounter with ADSL: this should become widespread quite soon as a replacement for home connections to the Internet.

Current modems are limited to 56Kb: this number is the absolute maximum speed you can get from a standard telephone line. This is because the telephone system filters out all frequencies apart from a 3KHz chunk centred about the human voice, even though the the cable (often Cat 2) can support up to 1MHz. This is why telephone calls have that characteristic “telephone sound”. This number was chosen as it is just enough to get recognisable speech though it, and you can then pack several telephone calls together on a single cable. Thus a traditional modem, which converts the data into noises to be transmitted down the telephone line, is limited by this.

Actually 4KHz is reserved, to give a 0.5KHz safety area either side. Then calls are *frequency shifted* and *multiplexed* on to a single wire.



Frequency transmission in a standard telephone cable

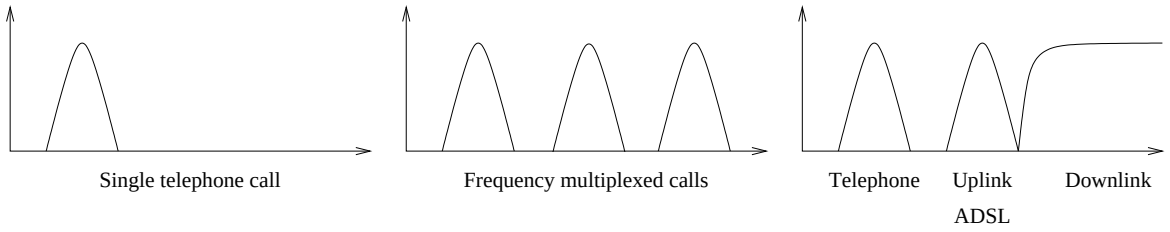


Figure 16: ADSL

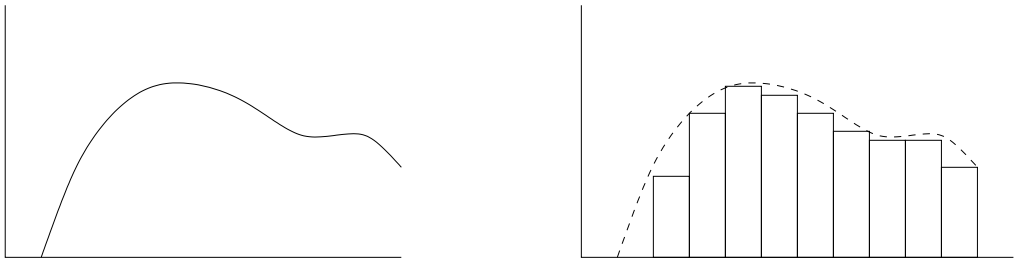


Figure 17: DMT

ADSL (one of a series of DSL standards) tries to use the maximum available bandwidth of a line: this means you can't use the current hardware in the telephone exchange, but must replace it with a specific ADSL box. Similarly, the user must have a specific ADSL modem.

A certain amount of bandwidth is available on the wire. This varies according to the quality of the wire, the length of the wire, the quality of the splices in the wire, local electrical interference, and many other things. The *asymmetric* refers to the non-symmetric splitting of this bandwidth into *uplink* (towards the user) and *downlink* (towards the Internet) parts. For example, you might get 256Kb downlink and 2Mb uplink speeds, though these numbers vary widely. And a 4KHz channel for your telephone on top of that. These are connected 24/7, and there is no need to dial up an ISP.

This asymmetry is not a real problem when you consider how most networks are used. A user types in a few characters or does a few clicks. This becomes a small chunk of data to be sent out along the slow link. The return data, e.g., a web page, is much larger and returns along the fast link. Users generate little data but expect big replies, and this matches the asymmetry.

Sometimes ADSL implementations overlap the upstream and downstream frequencies and rely on clever signal processing to separate the signals of the sending and receiving streams (called *echo cancellation*). This increases the bandwidth a little at the cost of extra complexity.

ADSL is often encoded using *Discrete Multi Tone* (DMT). This splits the available frequency range (26Hz to 1.1MHz) into 256 channels of 4kHz. Each channel encodes as many bits as noise on the line allows: this can be from 0 to maybe 15 bits/Hz. If a channel is particularly noisy (e.g., interference from fluorescent lighting) it can only transmit a few bits/Hz. Neighbouring channels that are outside the range of the interference can transmit at full capacity. In this way we make the most that the copper allows, working around the non-linear transmissions along the wire.

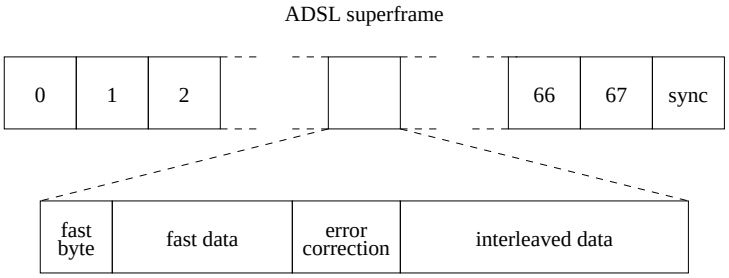


Figure 18: ADSL Frames

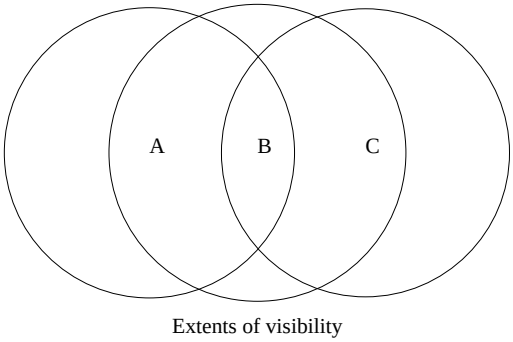


Figure 19: Visibility in Wireless Networks

Clever ADSL hardware (modems) is required both in the home and in the telephone exchange is required to get these speeds. This is currently relatively expensive, but costs should drop rapidly.

Data is transmitted in *superframes* containing 69 ADSL frames. Frame 69 is for synchronisation, the rest contain data. An ADSL frame contains two kinds of data: *fast data*, and *interleaved data*. The fast section, is used for time-sensitive data such as audio and can contain forward error correction bits. The interleaved section has purely user data. The *fast byte* has several uses: in frames 1, 34 and 35 it contains administrative flags; in frame 0 it contains a CRC; it can be used to signal to the other end that a CRC error occurred; and so on.

People are starting to use ADSL as a layer to support ATM: this will be important as we can then have IP over PPP over ATM over ADSL.

A popular method of delivery of ADSL to the home is to use a modem that plugs into the USB port on a computer. This means that an application, e.g., a web browser, will be implemented using HTML over HTTP over TCP over IP over PPP over AAL5 over ATM over ADSL over USB over copper!

Many other physical/link layers exist: e.g., wireless links are becoming increasingly popular.

5.9 Wireless

There are already many extensive wireless networks about, namely the cellular telephone networks. These were not designed with data transmission as a primary concern, so it is not surprising that that do not support data terribly well. The emergent 3G networks are supposed to address this problem, but they are having problems getting off the ground.

IEEE 802.11 We shall be concentrating on wireless LANS, in particular wireless Ethernet. The 802.11 group of standards deals with wireless Ethernet. In principle, wireless Ethernet is just the CSMA/CD protocol run over radio waves rather than copper, but naturally there are many problems unique to wireless networks.

These problems include:

- wireless networks generally have fairly high error rates due to electrically noisy environments, signal reflections and so on
- this means the bandwidth you get is heavily dependent on the circumstances of your environment and distance between hosts
- wireless networks generate electrical interference themselves
- if workstation A can "see" B (i.e., B's transmissions reach as far as A), and B can see C, then C can't necessarily see A, so A cannot be aware if there is a collision of its packets with packets from C
- as your packets are being broadcast, wireless networks are intrinsically insecure, so that extra care needs to be taken over security and authentication

---

A technique known as "war driving" involves driving about your neighbourhood with a wireless networked PC until you chance upon the signal from some unsecured system. Then you have free access to the Internet! Studies have shown that there are indeed many available networks out there. A similar trick works with cordless telephones.

The name "war driving" is derived from the older "war dialling" where you try every phone number in a company's telephone directory until you hit upon a modem. This modem often had unrestricted access to the company's computer system.

The are several parts to the standard: 802.11, 802.11a, 802.11b and 802.11g. 802.11 also contains a specification for infra-red wireless transmissions, but this doesn't seem to have been taken up by manufacturers with such great enthusiasm.

5.9.1 802.11b

IEEE 802.11
IEEE 802.11b

The 802.11 and 802.11b standards were designed for LANs and operate in the 2.4GHz waveband. The original 802.11 runs at 1Mb/s or 2Mb/s (i.e., a maximum of 1Mb/s and 2Mb/s, respectively), while 802.11b upgrades this to 5.5Mb/s and 11Mb/s. Machines can be up to 100m (300 feet) apart indoors, and up to 300m (1000 feet) outdoors. You might hear 802.11b being called "Wi-Fi" or even "AirPort".

Wi-Fi is actually a certificate of interopability given to manufacturers whose equipment demonstrably works with other manufacturers'.

5.9.2 Spread Spectrum

The bits are not transmitted over a simple single frequency as if it were a radio station. This is because there is a great deal of electrical noise in the 2.4GHz frequency band that must be overcome, in particular microwave ovens radiate in this band. Instead, the signal is spread over many frequencies with the range using a variety of techniques called *spread spectrum* transmission. Also called *wide band*, in contrast to *narrow band*.

Two popular ways of doing this are *Frequency Hopping Spread Spectrum* (FHSS) and *Direct Sequence Spread Spectrum* (DSSS). These provide some protection against interference (and jamming!), but also some security against interception: for example, in FHSS the hops are in a pseudo-random sequence that the sender and receiver both know, and an eavesdropper (presumably) doesn't. This spread spectrum transmission is much harder to tap than a fixed frequency transmission (but not impossible).

FHSS was developed about 50 years ago as by the military for secure wireless communication. Another advantage is that SS spreads the transmission power thinly over a large frequency range rather than having a big spike of power at one frequency. This is then harder to detect, and itself causes less interference to other electrical systems.

Some modern SS techniques use picosecond timing, and produce a signal that is indistinguishable from the background noise more than a few metres away from the transmitter, but nevertheless can be picked up and decoded several kilometres away!

FHSS in 802.11b uses 75 or more 1MHz channels, and it hops every 400ms or less (that is, 2.5 hops per second). A hopping pattern covers all the available channels before repeating itself, and there are 22 hop patterns to choose from. This provides up to about a 2Mb/s data rate.

An additional benefit is that multiple machines on the same network on the same channel can each have (just about) the full 11Mb/s bandwidth: as one transmission hops about its sub-frequencies it is quite unlikely to clash with a second transmission also hopping about within the same channel.

DSSS is used in 802.11b as it gives a higher data rate (up to 11Mb/s). This encodes a single bit as a collection of 10 to 20 *chips* in a *chipping code*. A chip is also a 1 or a 0, thus a 1 might be encoded as 00010011100. What is the point of doing this, as we are now sending 11 chips where we were previously sending just one bit? Well, the chipping codes are chosen so that a 1 bit from one source is clearly distinguishable from a 1 bit from another source, meaning two sources can be sending on the same frequencies and we can still pick apart the data streams. Furthermore, a few chips can be corrupted through interference and we still recognise the original bit sent.

The number of chips used in the code is called the . The 802.11b standard uses CCK, or *Complementary Code Keying* modulation with a processing gain of 11.

The available frequency range is split into up to 14 channels centred about specified frequencies: in most of Europe all channels are available, and each channel provides the full 11Mb/s. In other countries, such as France and Japan, fewer channels are available due to licensing restrictions. The multiplicity of channels allows wireless networks to exist side-by-side and they do not interfere as long as they are separated 30MHz or more. This means you can have a maximum of three neighbouring networks.

Channel	Frequency (GHz)	Channel	Frequency (GHz)
1	2.412	8	2.447
2	2.417	9	2.452
3	2.422	10	2.457
4	2.427	11	2.462
5	2.432	12	2.467
6	2.437	13	2.472
7	2.442	14	2.484

The 802.11b protocol employs *carrier sense, multiple access, collision avoidance* (CSMA/CA), which is similar to CSMA/CD, but more complicated. Before sending a packet the collision avoidance algorithm sends a *request to send* (RTS) frame. If the destination is happy with this (it is not already busy receiving data from some other machine), it replies with a *clear to send* (CTS) frame. On getting this, the source sends its packet. A third machine nearby will also see the CTS and know it cannot transmit yet. Furthermore, the RTS and CTS contain the length of the desired transmission, so the third machine knows how long it has to wait before transmitting.

Modern 802.11 network cards implement a microwave oven mode where they try to work around the bursts of radiation that leak from the typical microwave oven. Typically this can involve fragmenting packets into smaller parts so that only small chunks of data get blasted by microwaves.

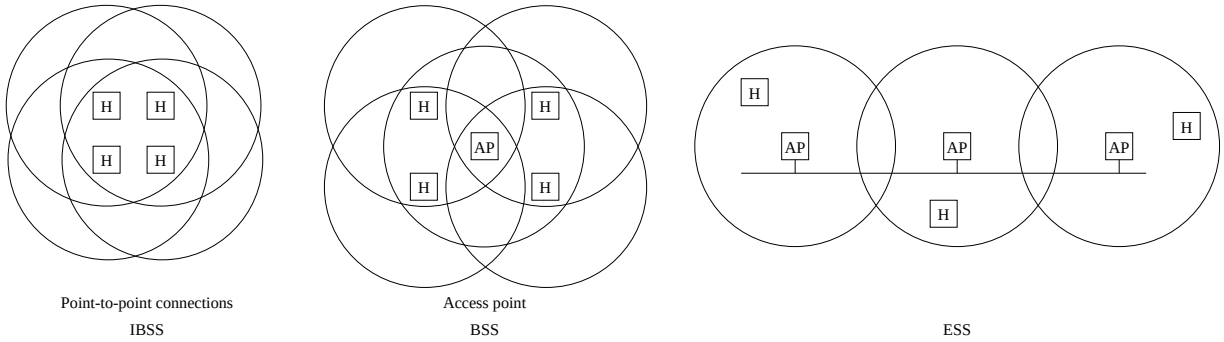


Figure 20: Wireless Networks

5.9.3 802.11a and 802.11g

IEEE 802.11a The 802.11a standard is designed for the user with high bandwidth demands, and operates in the newly allocated 5GHz arena. It uses a different encoding (*orthogonal frequency division multiplexing*, OFDM) to gain the extra bandwidth. It runs at up to 54Mb/s but you will often find it at 6Mb/s, 12Mb/s and 24Mb/s (amongst others). This standard works over a shorter range than 802.11b, but for a given power and distance between machines it allows a higher bandwidth.

IEEE 802.11g The 802.11g standard is backwards compatible with 802.11b (it uses the 2.4GHz band), but provides up to 54Mb/s over the same distances as 802.11b by using OFDM.

Why should anyone bother with 802.11a when 802.11g has the same bandwidth but a larger range? The principal reaon is that 802.11a has a specific set of frequencies reserved for it, while 802.11g uses the free-for-all 2.4GHz band and so is much more likely to get interference. Furthermore, you can only have a limited number of 802.11g networks physically close to each other (perhaps just three) but 802.11a allows for a dozen or so to operate together without mutual interference.

5.9.4 Wireless Networks

Networks can be arranged in a point-to-point (aka *ad-hoc* and *Independent Basic Service Set* (IBSS) mode) configuration, where each machine can see each other, and they contact their peer directly. Alternatively, they can be in a hub configuration where a central machine routes traffic (also called *infrastructure* or *Basic Service Set* (BSS) mode). This can work over a physically wider area than point-to-point, but requires a machine to be the hub or *access point*.

There is also *Extended Service Set* (ESS) mode where you have multiple access points connected by (say) a wired Ethernet. In this setup you can *roam* between access points. This is just like in cellular telephone networks which can handoff between transmitters as you physically move about.

For security, 802.11 employs *Wired Equivalent Privacy* (WEP) which uses the RC4 encryption algorithm with 40 or 128 bit keys. Both ends of a transmission share a secret key which is used to encrypt the traffic before it goes out on the airwaves.

The choice of key sizes was due to the restrictions at that time on exporting strong encryption systems from the USA. The 40 bit version could be exported, while the 128 bit version was for domestic USA use.

Note that 40 bit security is better than no security, but not by much, as it is not beyond the bounds of feasibility to go though all possible keys one by one. Furthermore, as people often use keys that are 5 character ASCII strings, this reduces the search space to about 30 bits.

It brute force is too much effort for you, it appears that the WEP algorithm itself is breakable with a bit of ingenuity: after collecting about a day's worth of traffic a computation will reveal the secret key. Even worse, WEP is typically off by default!

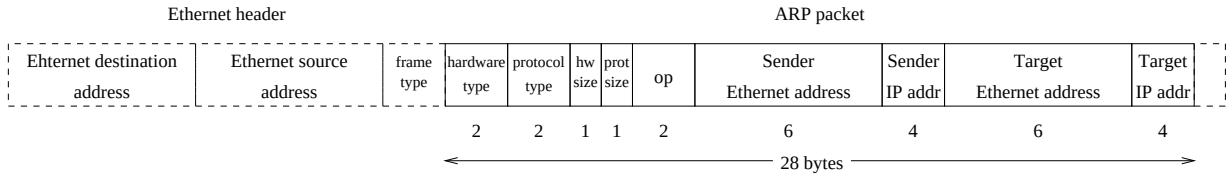


Figure 21: ARP

To overcome this, you should not rely on the link level encryption, but always encrypt traffic yourself (e.g., using `ssh`).

5.9.5 Other Wireless

Bluetooth is a another wireless technology rather akin to 802.11b, but is designed for short-range point-to-point connections between appliances like TVs, fridges and telephones. It has a range of 30 feet, but uses less power than 802.11b. It's better not to think of Bluetooth as a wireless LAN, but rather as a wireless cable.

Similarly there is HomeRF, though this appears to be lagging behind Bluetooth in popularity. All of 802.11b, Bluetooth and HomeRF use the 2.4GHz band, meaning there is potential for contention and interference in these networks. The performance of a network will suffer if there is one of the others nearby.

Other (wide area) systems include code division multiple access (CDMA) and the emerging 3G mobile telephone system.

It is possible that wireless networks may assist in the development of MANs since some people are looking at using wireless links to connect street-wide and larger collection of machines together in a *Community Area Network* (CAN).

5.10 ARP

RFC826

We now consider the *Address Resolution Protocol* here, as it is often considered to be part of the link layer. It forms part of the Internet Protocol, and exists to solve one very specific problem: the gap between physical layer addresses (e.g., Ethernet 6 byte) addresses, and network layer (e.g., IP 4 byte) addresses. This problem does not exist if we are using, say PPP, as this physical layer has no addresses.

Every machine on the Internet has at least one 32 bit (4 byte) IP address. It is unique to that machine (modulo some considerations that will become clear later). There is a convention in writing IP addresses intended for human consumption. The four bytes are written as a *dotted quad* of decimal values. Thus the address 00000001000000100000001100000100 is written as 1.2.3.4. For example, mary has IP address 138.38.32.14 (and Ethernet address 0:20:48:40:2e:4d). The Internet layer uses these addresses exclusively, as it knows nothing about the link layer. IP addresses are separate for precisely this reason: so that IP can run over many different link layers.

The difficulty lies when the IP layer wants to send a packet over Ethernet. It knows the destination's IP address, whereas to build the Ethernet frame we require the destination's Ethernet address. Thus we need to find which Ethernet address corresponds to which IP address. This is done by ARP.

To determine the Ether address for an IP address, the source machine broadcasts an ARP packet using Ethernet address ff:ff:ff:ff:ff:ff. All machines on the (local) network look at the packet, and the target machine recognises that the packet is for it, and responds with an ARP reply containing its hardware address. The source machine reads this and extracts the Ethernet address.

This packet is embedded in an Ethernet frame, of course. The frame type for ARP packets is 0806.

The packet in more detail:

- Hardware type. This is 1 for an Ethernet address.
- Protocol type. This is 0800 for an IP address.
- Hardware size, Protocol size. The number of bytes each of these kinds of addresses occupy. This is 6 for Ethernet and 4 for IP.
- Op. This is 1 for an ARP request, and 2 for an ARP reply.
- Address fields. The hardware and protocol addresses for the source and target.

In an ARP request the target hardware address is not filled in; after all, that is what we are trying to find out. In an ARP reply, the target is the original sender, and the sender hardware address field contains the information we are interested in.

Of course it would be stupid and a waste of network bandwidth to have an ARP for every packet sent: instead a cache of mappings from IP to Ethernet addresses is kept. Entries in this cache time out and are removed after 20 minutes, typically.

If no machine on the network has this hardware address, or that machine is down, no reply will be forthcoming. If an ARP reply is not received after a couple of seconds, an error message will be sent to the application trying to make the IP connection.

Use the Unix command
`/usr/sbin/arp -a`
to look at the cache.

The ARP can be used in situations other than Ethernet and IP as it has parameterised fields and so can be used to associate pairs of any types of addresses, but it is by far most associated with Ethernet.

5.11 Reverse ARP

RFC903

Reverse ARP, or RARP, solves the opposite problem: given a hardware address, find the IP address. This might happen when a diskless machine is booting and needs to find its IP address and only has its Ethernet address to go by. Alternatively, a refrigerator may make a RARP request when it boots to find its IP address on the home network.

RARP is almost identical to ARP, but has frame type 8035, and op numbers 3 for a RARP request, and 4 for a RARP reply. Some machine on the local network assigned to deal with the RARP service will read these packets and reply.

RARP is simple, but is limited by the fact that a hardware broadcast only goes to the local network: it is not passed between networks. The reason for this limitation on hardware broadcast is that you can't pass on broadcasts arbitrarily, or else every broadcast will spread across the entire Internet! So some machine on the local network must be prepared to reply to a RARP, which may be inconvenient for large multiply subnetted networks. More general solutions to the hardware to software mapping problem exist: see BOOTP and DHCP.

6 The Internet/Network Layer: IP

RFC791

We now turn to look at the next layer in the Internet Protocol: the Internet layer.

This the basis the the Internet is built upon: a reasonably simple protocol that provides a foundation for the more complicated higher layers.

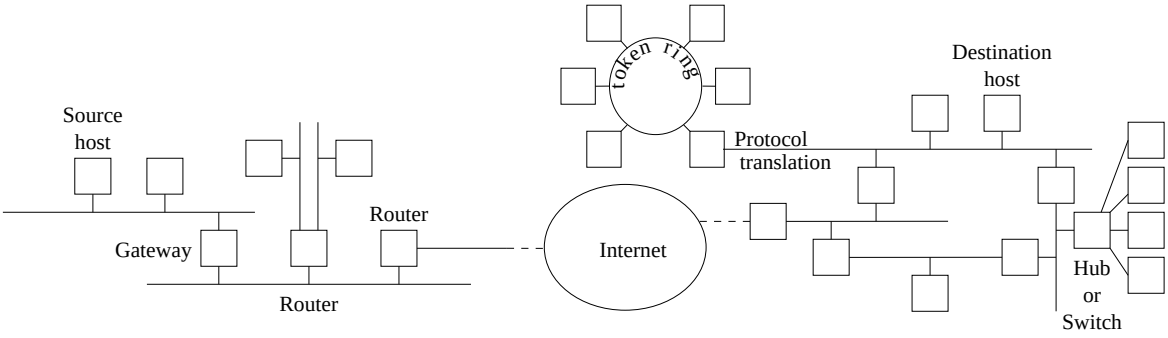


Figure 22: Internet

We shall describe the *Internet Protocol* (IP). Actually, this is the fourth incarnation: IP version 4 (IPv4). We shall talk about IPv6 much later (see section 6.17).

The IP provides a best-effort, connectionless, unreliable packet protocol. This combination was chosen as it represents a lowest common denominator of network properties, and it relies the least on functionality of the link layer. Thus IP can run on top of several link layers, be they reliable or not.

IP is a cooperative system. For a packet to get to its destination it is handed on from one machine to the next, step by step.

The nodes in the system have various names according to their function.

- Host. A computer that you actually use to do some work.
- Gateway. A machine that connects two networks.
- Router. A machine whose primary function is to decide where a packet goes next.

These are by no means mutually exclusive, and you use a term to indicate how you are thinking of a component at the time. Gateways and routers are often hosts, though dedicated routing hardware is popular. Gateways do routing. An individual packet does not know how to get from source to destinations: the routers figure this out, and it can be a very complicated business. See later.

The IP layer breaks the data stream into packets, also called *datagrams* in the context of IP, and prepends a header. A packet (including header) can be as large as 64KB, but is usually around 1500 bytes (this is only because much IP runs over Ethernet, and Ethernet has this limit).

> We have the three words *frame*, *packet* and *datagram* all meaning roughly the same thing: a chunk of data. Later we shall add *segment*. The only difference between them is the emphasis we want to put on the chunk: for example, frames are typically at the link level.

Immediately we are faced with the problem of different link layers having different properties such as maximum packet size. What happens when a big packet hits a network with a smaller maximum limit? The IP deals with this by *fragmentation*: a single packet is subdivided by a router into several smaller packets. See sections 6.5 and following, below. It is the destination's problem to glue the fragments back together.

We shall now deal with each field in the header in turn. The header and data are transmitted in *big endian* format, that is left to right, top to bottom in the diagram.

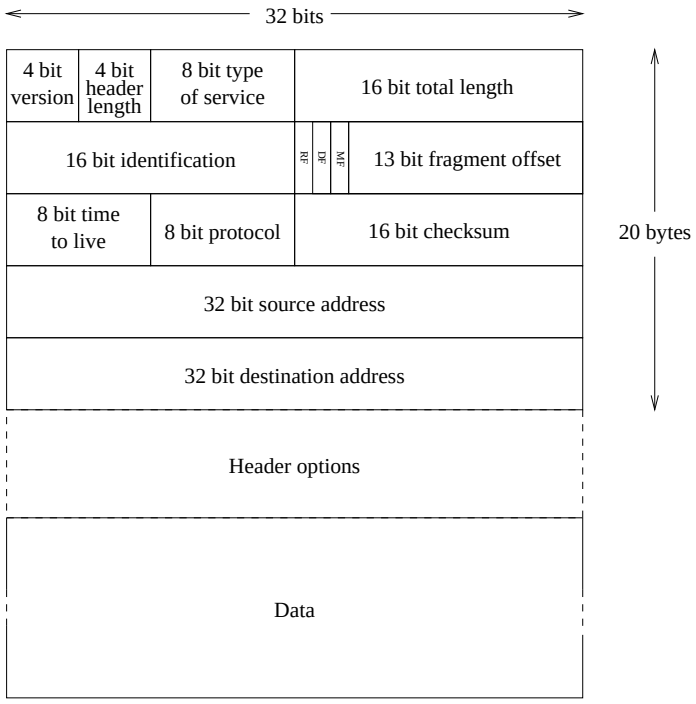


Figure 23: IP Datagram

6.1 Version

This four bit field contains the version number, 4. The latest version of IP, which is being introduced very slowly, has version number 6. This field is used to aid such a transition.

6.2 Header Length

The header can vary in length: there are some optional fields later. So we need to be able to find where there header ends and the data starts: how can we distinguish between a header field and data that happens to look like a header field? An easy way is to supply the header length. This is given as the number of 32 bit words, and is generally 5 which is the minimum length of the header (no optional fields).

This 4 bit field has maximum value 15, or 60 bytes, so we can have up to 40 bytes of options.

6.3 Type of Service

This field allows us to indicate how the datagram should be treated in terms of speed and reliability. For some data streams speed is more important than 100% reliability. Voice is a good example of this. On the other hand, when we are transmitting data files, when accuracy is paramount, we are willing to sacrifice a little speed for improved reliability.

Not all implementations of IP take notice of this field, but it is becoming increasingly important as networks providers want to charge different rates for different levels of service.

RFC2481 Recently, bits 6 and 7 of the TOS field are being used in an *explicit congestion notification* (ECN) mechanism. See section 14.5.3. Otherwise these bits should be set to zero.

Application	Minimise delay	Maximise throughput	Maximise reliability	Minimise cost	Hex value
Telnet/Rlogin	1	0	0	0	0x10
FTP					
control	1	0	0	0	0x10
data	0	1	0	0	0x08
Bulk data	0	1	0	0	0x08
ICMP errors	0	0	0	0	0x00
SNMP (network config)	0	0	1	0	0x04
NNTP (news)	0	0	0	1	0x02

Figure 24: Some of the recommended values for TOS field

6.4 Total Length

This is the total datagram length, including the header, in bytes. A 16 bit field gives us a maximum size of 65535 bytes. This is tolerable to most people at the moment, but is way too small for those who like to push about big data, and will be way too small in the near future with the arrival of gigabit networks.

Datagram size is important as we have overheads in using packets:

- time overhead spent in splitting data into packets, adding headers, and then removing headers and reassembling the datastream at the other end
- bandwidth overhead in that we are using 20 bytes for the header and not for data.

A bigger packet means better amortisation of overheads. In the limit, with an infinite packet size, more familiar as a connection oriented system, the overhead is minimised. But then we have the problems of a connection oriented system.

6.5 Identification

This is an integer that is unique to each IP datagram, often incrementing by 1 for each successive datagram sent. It is used to reassemble a datagram if it gets fragmented. When a datagram is fragmented, it is split up into several smaller datagrams, each with copies of the original IP header (a few fields are changed, e.g., the total length). The identification field serves to group together fragments that came from a single datagram. A fragment is a datagram in its own right, and can be fragmented itself.

6.6 Flags

Three bits are used as flags. Or rather, two are used and one is reserved.

1. RF. Reserved for later use. Unlikely ever to be used since the advent of IPv6.
2. DF. *Don't Fragment*. If the destination is incapable of reassembling fragments this bit is set to inform the routers on the path to the destination not to fragment. This might involve the routers choosing a slower path that does not fragment, or if this is impossible they might drop the datagram and return an error message back to the sender. All machines are required to be capable of accepting datagrams of size 576 bytes.

RFC791

In detail: all machines are required to be capable of *accepting* datagrams of size 576 bytes, either whole or in fragments. They must be able to forward datagrams of size 68 bytes without fragmentation (maximum 60 bytes for an IP header plus 8 bytes of a fragment).

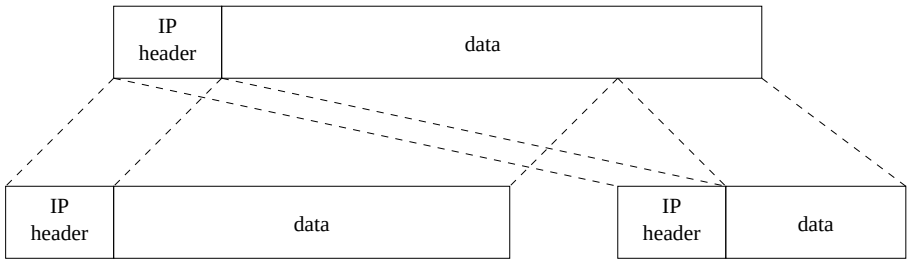


Figure 25: IP fragmentation

3. MF. *More Fragments*. All fragments except the last have this set.

6.7 Fragment Offset

This is an integer that locates the current fragment within the current datagram. Notice that the individual fragments may arrive at the destination out of order as they might take different routes. This value is a byte offset, shifted down by three bits. This means two things: (a) 13 bits is enough to cover the 16 bit range of offsets, and (b) each fragment (other than the last) must be a multiple of 8 bytes long.

Each fragment is a chunk of the original datagram data with a copy of the original IP header and the various fragmentation fields set appropriately. When the fragment with MF not set is received, the destination can use the fragment offset plus fragment length to determine the size of the original datagram.

Fragmentation is a difficult problem, and this is not the best solution as it causes a drop in efficiency:

- fragmentation in a router slows down the processing of a datagram immensely: unfragmented segments can be sent on much more quickly,
- extra overhead as more headers are being used for a given amount of data,
- extra overhead as more segments are traversing the network,
- if a single fragment is lost, the entire datagram must be retransmitted, which is a huge waste of bandwidth,
- the greater the number of fragments flying about the greater probability one will be lost.

IP does not do retransmission itself: it it up to a higher layer to do this if required.

IP implementations clear out fragments for an incompleted datagram after a suitable time, e.g., 30 seconds after the arrival of the first fragment (that is, the fragment that *arrived* first). See section 17.2.3.

Well behaved implementations return an ICMP error “timeout during fragment reassembly”. A further wrinkle is that the rules say that if the initial fragment is lost (the fragment with offset 0) an ICMP need not be generated. This is because only that fragment contains the information (e.g., a UDP or TCP header) that will allow the source to identify the sending process and tell it something has gone awry.

Better is the approach of IPv6, where a datagram is never fragmented en route. In IPv6, if a gateway realises a datagram is too large for the next hop, it drops the datagram and sends an ICMP error message back to the source. The source can then resend several smaller datagrams. This reduces the complexity of routers considerably, and consequently speeds up routing. Just as importantly, the loss of a single segment will be covered by the retransmission of that single segment.

Fortunately, even with IPv4, most of the time fragmentation is not required, and furthermore hosts are able to negotiate packet sizes by *MTU discovery*. The *Maximum Transmission Unit* is the largest packet size a host or network can cope

with. The *path* MTU is the smallest MTU on the path from source to destination. The path MTU can be determined by sending various size packets with the DF flag set, and watching for errors returned. When an unfragmented packet gets to the destination we have a lower bound for the MTU. Note that this is only an approximation, as we are trying to measure a dynamic system: in each probe the packet may take a different route that has a different path MTU! See section 14.9.

Find the MTU of an interface by
`/usr/sbin/ifconfig -a`
or `netstat -i`

6.8 Time to Live

This is a counter that is used to limit the lifetime of a datagram. If the routers get confused or are badly configured, a datagram may bounce back and forward or around in circles indefinitely. Eventually the network will be filled with lost datagrams, and the real ones will not be able to get through.

The TTL is designed to stop this. The TTL starts off at some value, say 64 or 32. Each time the datagram goes through a router its TTL is decremented. If the value ever gets to zero, the router must drop the datagrams and send an error back to the source.

The 8 bit field limits us to a maximum TTL of 255, but currently the Internet is not nearly that wide.

The width of the Internet goes up and down all the time. It goes up when new networks are attached, but it goes down when extra links are installed or traffic is tunnelled over ATM, say. The IP regards a tunnel as a single hop, no matter how long or complex the ATM route might happen to be.

Originally this field was supposed to measure time in seconds: a router was supposed to decrement the field for each second it was queued in the router. In practice no-one did this, but just decremented the TTL on each hop.

The Internet is like this. What happens out there in the real world is important, not what the standards say. Though, on the whole, most try to follow the standards.

6.9 Protocol

RFC1700 et seq

This connects the IP layer to the transport layer. This field contains a number that tells us what transport system to pass the datagram on to. There are numbers defined for TCP and UDP amongst others.

6.10 Header Checksum

As in the Ethernet layer, this is a simple function of all the bytes in the header. If the checksum is bad, the datagram is silently dropped. It is up to a higher layer to detect a missing datagram and request retransmission: recall that IP is unreliable.

This is not a perfect system as an error could arise in the checksum and nowhere else, meaning an otherwise good datagram is dropped; or several errors in the header could combine and cancel themselves out; and so on. More sophisticated checksums could be more robust, but would be more time consuming to compute. Notice that the TTL changes in each router, so the router must recompute and update the checksum on each hop.

6.11 Source and Destination Address

These are 32 bit numbers that uniquely determine the source and destination machines on the Internet. That is, each machine on the Internet has a different IP address. This gives us a maximum of 4,294,967,296 machines on the Internet: actually fewer are available since some addresses are reserved for special purposes. This is not really enough. We discuss IP addresses below.

6.12 Optional fields

This is a variable length list (usually absent) of optional bits and pieces, originally included in case the designers thought of something new they wanted to add to IP. Also for rarely used stuff, so we don't clutter the header with mostly unused fields.

Options include

RFC1108

- Security (encryption) and authentication
- Record route: each router records its address in the datagram as it passes by
- Timestamp: each router records its address and current time in the datagram as it passes by
- Strict source routing: a list of addresses of routers that give the entire path from source to destination
- Loose source routing: a list of addresses of routers that must be included in the path from source to destination

All except the first are for debugging the network. The limitation of the options to 40 bytes means that only 9 routers can squeeze their addresses in. Not enough for these days when routes can easily be over 30 hops.

6.13 IP Addresses and Routing Tables

We now need to look at those 32 bit addresses. As mentioned, every machine on the Internet has its own unique address. This is so every machine can contact every other machine. These numbers are strictly controlled by the Internet Assigned Number Authority (IANA), and more about this later. They are not dished out at random, but are carefully allocated to make routing between networks that much easier.

Consider the complexity of the problem of finding a route between two arbitrary machines. If there was no structure on IP addresses, this would be impossible without gigantic tables that detail each and every machine.

Instead, remember that the Internet is actually a collection of networks. The IP address is split into two parts: firstly the network number, and secondly the host number. The host number defines the host uniquely on the network, and the network number defines the network uniquely on the Internet.

To an end host, routing is trivial: if the destination is on the local network, simply put the packet out on the network. If the destination is not local, simply send the packet to the network gateway and let it deal with the problem.

> If we are running over Ethernet, the Ethernet hardware address on the packet will be that of the gateway, not of the destination machine. Recall that the link layer can't know about IP addresses, all it knows is this packet must go to the gateway. The gateway will send on the packet with a hardware address appropriate for the next hop, and so on. In terms of ARPing for this packet, we make a request for the gateway IP address.

So, to a first approximation, the routing problem is that of directing packets between networks. Once a packet has arrived at the destination network the gateway simply sends to packet to the destination host. This is much better, as there are significantly fewer networks than hosts.

Routers contain tables of (network layer, IP) host and network addresses, together with information about what to do with packets with these addresses.

1. Destination address. This can be the address of a specific machine, or the address of a network.
2. Address of the *next hop* router, i.e., the address of where to send this packet next. This is the address of the immediate next router that is directly connected to the current router.
3. Which *interface* to send the packet out on to get to the next router. A router has, of course, several interfaces, i.e., network connections, so we need to know which connection goes to the next router.

When a packet arrives at a router it checks its table.

1. If the packet destination address matches a host address in the table send the packet out to the appropriate next router on the appropriate interface.
2. Else if the packet destination address matches a network address in the table send the packet out to the appropriate next router on the appropriate interface
3. Else find an entry in the table marked *default*, and send the packet out to the appropriate next router on the appropriate interface

If none of the above works, drop the packet, and send back an error message “host unreachable” or “network unreachable”

We return to the network routing problem later, in particular how the information gets into the tables, but for now regard routers as machines with big tables that tell them where to send packets.

Use `netstat -r` to see the router table on a Unix machine.

```
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
138.38.96.0     0.0.0.0         255.255.248.0   U         0 0          0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U         0 0          0 lo
default         138.38.103.254  0.0.0.0         UG        0 0          0 eth0
```

Every host has a routing table, and generally this says to send packets destined for the local network out on the network interface, and send other packets to the default gateway. The above table indicates that packets on the local network (138.38.96.0) should be sent out on interface `eth0`, and the default is to send to the gateway 138.38.103.254, which is also attached to interface `eth0`.

There is also an entry for a *loopback network*, a virtual network internal to the machine, found on the (virtual) interface `lo`. This network connects a machine to itself, and useful for testing network programs.

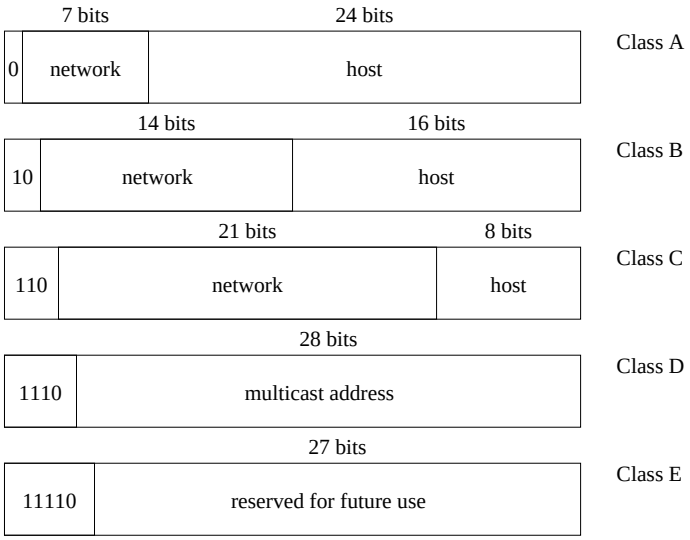


Figure 26: IP addresses

6.14 Networks and IP addresses

RFC943

So how are the addresses split into network plus host parts? If we give, say, 8 bits to represent the network and the rest to represent the hosts on the network, we will have $2^8 = 256$ networks each with $2^{24} = 16777216$ hosts. This isn't really enough networks, and not many people require that many hosts. A few do, though. Splitting it up the other way we could have 16777216 networks each with only 256 hosts. This is too small a network for many people, e.g., large corporations, but fine for small companies. Cutting it down the middle we can have $2^{16} = 65536$ networks with 65536 hosts. Still not really enough networks, and too many or too few hosts per network according to taste.

As different people have different requirements, a compromise is used: we split the address in (essentially) three ways!

- *Class A* addresses, from 1.0.0.0 to 127.255.255.255 have 7 bits for the network and 24 bits for the host: this is 126 networks containing 16777214 hosts. The address x.y.z.w has x as network, y.z.w as host.
- *Class B* addresses, from 128.0.0.0 to 191.255.255.255 have 14 bits for the network and 16 bits for the host: 16382 networks, 65534 hosts. The address x.y.z.w has x.y as network, z.w as host.
- *Class C* addresses, from 192.0.0.0 to 223.255.255.255 have 21 bits for the network and 8 bits for the host: 2097152 networks, 254 hosts. The address x.y.z.w has x.y.z as network, w as host.

There are also

- *Class D* addresses, from 224.0.0.0 to 239.255.255.255 are *multicast* addresses. *Multicast* is sending a single packet to multiple hosts. This is in contrast to *broadcast*, which is sending to *all* hosts on a network.
- *Class E* addresses, from 240.0.0.0 to 247.255.255.255 are reserved for experimental and future use. This might include the transition to IPv6.

Various values are reserved for special purposes:

- host number 0: "this host". Sometimes mistakenly used as a broadcast address
- host number all all 255s: broadcast address to network

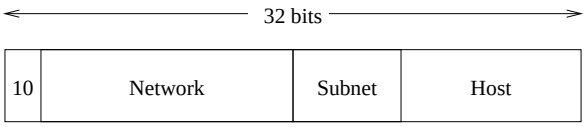


Figure 27: Subnetting

- network number 0: “this network”
- network 127.0.0.0: the loopback network.

The idea behind this scheme is that the IANA can allocate big networks to those who want a lot of hosts, and small networks to those who don’t need so many. Thus we don’t lose large chunks of the address space to Two Men and a Dog Enterprises.

The University of Bath has been allocated addresses in the network 138.38. This is a class B address, and so we have 65534 possible hosts. Stanford University has 36, a class A address. The class C address 193.0.0 is allocated to *Réseaux IP Européens* (RIPE), the European Internet Registry (responsible for IP addresses in Europe).

This kind of information can be found using `nslookup` and PTR queries, and using `whois` to find who owns a name.

6.15 Subnetting

RFC950

Suppose you have your IP network address, a class B address, say, and you are building your network. You have 64 thousand host addresses to play with. Having a single network with 64K addresses on is not a good idea: it is very hard to administer for a start, even before we get into technical reasons. The solution is to use *subnetting*. This allows us to split our network into smaller independent subnetworks. Each subnet can be administered independently, e.g., by different Departments.

A *subnet mask* is used. This tells routers which part of the address is the subnet address, and which is the host address. For example, the Department of Maths has a subnet that contains addresses from 138.38.96.0 to 138.38.103.255, or 4096 values. In binary

network address	138.38.96.0	10001010 00100110 01100000 00000000
broadcast address	138.38.103.255	10001010 00100110 01100111 11111111
mask	255.255.248.0	11111111 11111111 11111000 00000000

A machine can tell if an address is on the local network if the host address ANDed with the mask gives us the network address. Thus the address 138.38.100.20 is on the Maths subnet since

host address	138.38.100.20	10001010 00100110 01100100 00010100
mask	255.255.248.0	11111111 11111111 11111000 00000000
AND	138.38.96.0	10001010 00100110 01100000 00000000

but 138.38.104.20 is not, since

host address	138.38.104.20	10001010 00100110 01101000 00010100
mask	255.255.248.0	11111111 11111111 11111000 00000000
AND	138.38.104.0	10001010 00100110 01101000 00000000

Outside the network 138.38, subnetting is not visible, so no applications to IANA or changes to routing tables are required if we move things about locally. A subnet can itself be subnetted.

Use the Unix command
`/usr/sbin/ifconfig -a`
to see the way a network is configured. The format varies from machine to machine, but this will tell you

- `inet` the machine's IP address
- `netmask` the subnet mask

amongst other interesting stuff.

The Maths subnet is can be described as "138.38.96.0, subnet mask 255.255.255.248.0", or more succinctly as "138.38.96.0/21", where 21 is the number of 1 bits in the mask. The subnetting standard does not actually require subnet masks to use the top n bits, and we could have the Maths network as 138.38.0.96, subnet mask 255.255.0.248" if we really wanted. However, it is overwhelmingly the case that people follow the former format, particularly when
RFC1219 *classless networks* are involved. (The short $/n$ format only applies to the top-n-bits style of mask.)

6.16 Classless Networks

Everybody wants a class B network, since a class C is too small, and a class A is too large (you pay per address).

This is called the *Three Bears Problem*.

Thus there are very few class B addresses left. One thing you might do is take several class C networks and link them together. This is not trivial, as you cannot have two network addresses on the same physical network. Thus you must have two networks, probably joined by a gateway. And every time the network grows, you have to apply for a new class C address and join it up to your existing system.

Additionally, many small networks is a problem for routing: many networks means large tables in routers. Again, this is not good.

Due to the massive growth of the Internet, some way of managing the IP address space had to be invented. There are a few ways we can go:

1. change the way classes of networks are defined
2. use private networks with *network address translation*
3. increase the number of addresses available by changing IP

The first two ways are backwardly compatible with the existing system. The third is a radical change.

6.16.1 CIDR

RFC1519

This is *Classless InterDomain Routing*.

There are many class C network addresses left. CIDR is a way of packaging them up that (a) allows networks of larger than 254 hosts, and (b) simplifies routing.

Firstly, blocks of class C networks are allocated

- 194.0.0.0–195.255.255.255 Europe
- 198.0.0.0–199.255.255.255 North America
- 200.0.0.0–201.255.255.255 Central and South America
- 202.0.0.0–203.255.255.255 Asia and the Pacific

This gives each region about 32 million addresses. Another 320 million addresses from 204.0.0.0 to 223.255.255.255 are waiting for later allocation. Now routing between such addresses is easy: anything that starts 194 or 195 is routed to Europe. This is a single entry in a router table, rather than an entry for each network.

Within each region, the same idea is repeated. Contiguous blocks of class C addresses allocated to ISPs or end users. Say 192.24.0 to 192.24.7. This is described as network 192.24.0.0/255.255.255.248, or more commonly, 192.24.0.0/21. The 21 is the number of 1 bits required to mask off the network address part from the whole address.

192.24.0.0	11000000 00011000 00000000 00000000
192.24.7.0	11000000 00011000 00000111 00000000
255.255.255.248	11111111 11111111 11111000 00000000

And we now know that any packet with address that has addr $AND\ 255.255.248 = 192.24.0$ should be routed to that ISP or user.

Example: 193.0.0.0/22 is allocated to RIPE.

This is a very clever update to the original class system, as the end machines do not need to know about CIDR. Once a packet has reached the destination network, it can be treated identically to a classed network. Only the routers need to know anything special, and only those external routers that connect your network to the rest of the Internet. A classless network can used just as a subnetted classed network, and may even be subnetted further.

This has been a valuable addition to IP, and has allowed the Internet to grow much further than once imagined. Extensions to CIDR to use class A addresses are also available.

Compare CIDR with subnetting: CIDR merges small networks into a larger one, while subnetting divides a large network into smaller ones. In fact, CIDR is sometimes called *supernetting*.

> It was reported at the end of 1999 that some routers were close to having 100,000 entries in their route table. This, while large, is still less than the 2,000,000+ entries that classic IP without CIDR would have required.

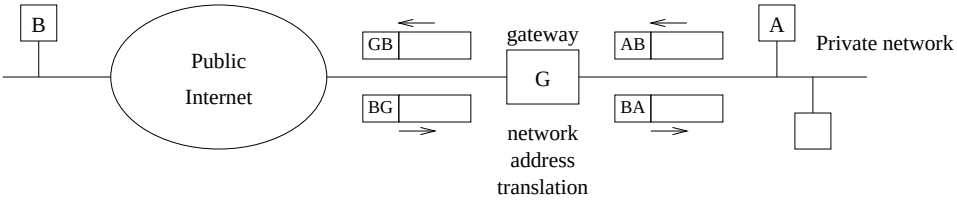


Figure 28: Network Address Translation

6.16.2 Network Address Translation

RFC1918 Some IP addresses are reserved for private networks.

- Class A: 10.0.0.0–10.255.255.255
- Class B: 172.16.0.0–172.31.255.255
- Class C: 192.168.0.0–192.168.255.255

These are one class A network, 16 class B networks and 256 class C networks, though with CIDR, you can supernet the B and C networks.

These addresses are guaranteed never to be allocated for use in the public Internet. Thus if you have a private IP network that is not connected to the Internet, these are safe addresses to use. How does this help if we *do* wish to be connected? There is a process known as *masquerading* or *Network Address Translation* (NAT) that converts an outward packet with a private address to one with a public address, and an inward packet with a public address back to the private address.

A packet going from the private network network to the Internet might have source address 10.0.1.1, and destination address 138.38.32.14. Let us suppose the gateway has address 138.38.32.252. The action of NAT is to make the Internet believe that this packet came from 138.38.32.252. We can do this by updating the source address in the IP header to be 138.38.32.252. This is done in the gateway. The gateway must remember this has been done, so when a reply comes back (with source 138.38.32.14, destination 138.38.32.252), it can re-edit the header to replace the destination with 10.0.1.1, and pass the packet on to the private network.

Thus it appears to hosts on the private network that they are connected to the Internet, while it appears to the Internet that a lot of traffic is originating at the gateway. This has the added property that machines on the Internet cannot refer to machines on the private network, as the addresses 10.0.0.0 can never be used on the Internet. This may seem a problem, but is actually an advantage to most people: it prevents external users hacking into the private network as they have no way of referring to the private machines. Only packets in reply to ones originating inside can travel inwards. See also section 18.1.

There are problems with this technique: sometimes IP addresses are passed in the data part of the packet (e.g., FTP, Quake). This is often done to set up other connections to various places. In order for the NAT to work in this case, the masquerading code has to be taught about such examples, and where to look in the data for these addresses. This is technically quite difficult, but can be done, and is very effective.

This technique is particularly good for home networks, where you can share one dial-up line between several hosts.

6.17 IPv6

RFC2460

Here is a brief introduction to the next version of IP. Variously called IPv6 and IPng (*next generation*) it takes IPv4 in the light of modern experience and reworks things to be simpler and more efficient. It has has 128 bit addresses and
RFC2373 uses a CIDR-style geographical allocation (amongst other types of allocation), which is what IPv4 really should have

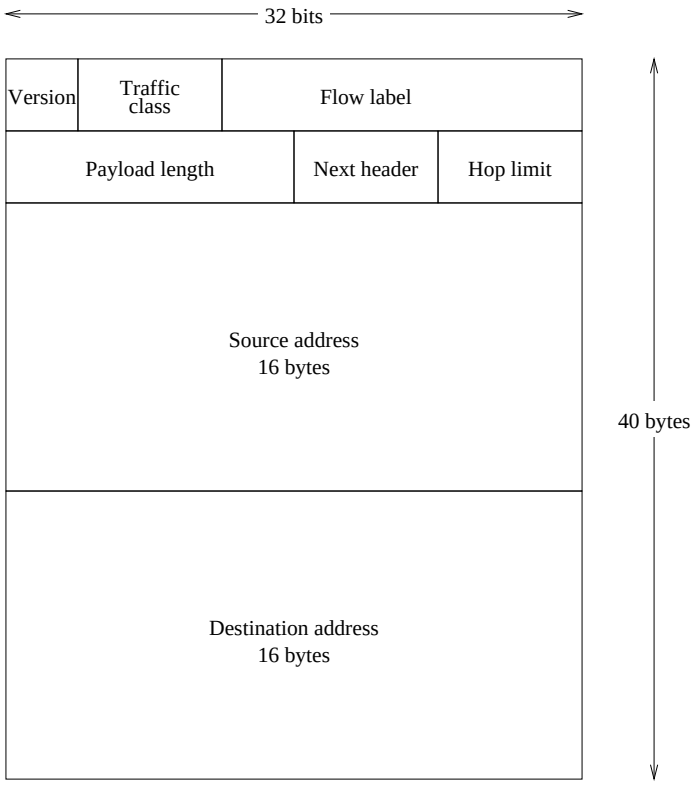


Figure 29: IPv6 Header

used in the first place. IPv6 is gradually being introduced, but it will be a long time before it replaces IPv4 everywhere.

IPv6 addresses various shortcomings of IPv4:

1. larger address space
2. reduce size of router tables
3. simplify the protocol so routers can process packets faster
4. provide security and authentication
5. pay attention to type of service
6. have better multicasting support
7. have mobile hosts with fixed IP addresses
8. room for evolution of the protocol
9. permit IPv4 and IPv6 to coexist during the transition.

One thing the designers of IPv6 did was to make the names of the fields more clear and more relevant to their actual use.

- Version, 4 bits. The number 6. This field is identical in position to the field in IPv4, and can be used to distinguish packets in mixed-version environments.

- Traffic class, 8 bits. Like TOS in IPv4. This can be used to differentiate different kinds of traffic, e.g., video want constant bit rate but can drop the occasional packet; FTP allows variable bit rate but must be loss-free.
- Flow label, 20 bits. This allows routers to recognise packets as belonging to a single *flow*, that is a stream of packets from a certain host to a certain destination. Given this, the routers can endeavour to give special treatment to these packets, such as reserved bandwidth or minimal delay. This effectively sets up a virtual circuit for the flow that can guarantee, say, sufficient bandwidth for video stream. This field is set to 0 is flows are not being used.
- Payload length, 16 bits. The number of bytes that follow this fixed 40 byte header. The IPv4 count included the header.
- Next header, 8 bits. This plays the role of the protocol field in IPv4, but also allows for IPv6 options. This field indicates which type of optional header (if any) follows the fixed header. If there are no optional headers (as is usually the case), this contains the transport protocol, such as TCP or UDP.
- Hop limit, 8 bits. This is the TTL field, but renamed to make it clear what is actually happening.
- Source and Destination addresses, 128 bits each. The biggest change: addresses are four times as long.

The use of 128 bit addresses gives us a potential $2^{128} \approx 3 \times 10^{38}$ addresses. Initial allocations of addresses are following the lead of CIDR, and are mostly geographical. But there is plenty of room for flexibility. This number of addresses is enough for 7×10^{23} addresses per square metre of the Earth's surface, or roughly enough to give an IP address for every molecule on the surface of the Earth.

RFC2373 Some addresses are unicast, some multicast. IPv6 adds *anycast* addresses: an anycast address identifies many machines, just as multicast, but instead of sending the packets to all of them, IPv6 picks one (often the "closest") and sends the packet to this one. This can be used for load sharing. The BBC might have an anycast server in the UK, and another in the USA. USA clients will contact the USA server, UK clients the UK one. This would save much traffic over the Atlantic link.

IPv6 has no fragmentation field: instead of routers large fragmenting packets, IPv6 just drops them and sends an ICMP error message back to the source. The source can then resend smaller packets. An optional fragmentation header is available for the unlikely situation when the source cannot re-packet the data into smaller chunks: it proceeds pretty much like fragmentation in IPv4, but is in the source and destination only. A router need never be concerned with fragments which is a huge simplification over IPv4, and means that packets can be routed that much faster. IPv6 requires all links in the network to have MTUs of 1280 bytes or larger. If any link can't do this, a layer *below* IP must do fragmentation.

IPv6 has no header length field. This is because the header is of fixed length. The *next header* field deals with options, more properly called *extension headers*.

IPv6 has no checksum field. Since most modern networks are pretty reliable, a checksum is not needed as much: and the transport layers (TCP and UDP) have checksums themselves anyway, so what's the point of duplicating work? One problem with IPv4 checksums is that the IP header changes in every router as the TTL decreases, meaning the header checksum must be recomputed every hop. In IPv6 we avoid this cost, and so packets are routed faster.

IPv4 has 13 fields in the fixed part of the header, while IPv6 has 8. IPv6 has addresses four times the size of IPv4, but the header is only twice the size, and is much simpler.

The next header field daisy-chains extension headers together until we reach the end. The last next header field contains the protocol of the data in the packet.

Extension headers are either of fixed length or have Type, Length, and Data fields. The top two bits of the Type indicate what to do if the header is not recognised:

- 00. Skip this option

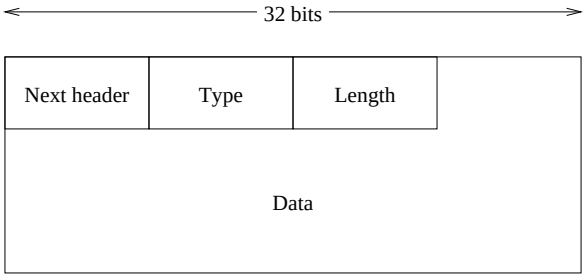


Figure 30: IPv6 Extension Headers

- 01. Discard this packet
- 10. Discard this packet, and send an ICMP error back to the source
- 11. Discard this packet, and send an ICMP error back to the source only if the destination was not a multicast address (so one bad packet will not produce millions of ICMP errors)

Extension headers include, amongst others: routing options (c.f. source routing in IPv4); fragmentation management; authentication; security; jumbograms (very large packets, bigger than the $2^{16} = 64$KB limit given by the payload
RFC2675 field: a single jumbogram packet can be up to 4GB in length!).

UDP and TCP layer just as before over IPv6, though small changes are needed in the way their checksums are computed. DNS adds a new RR type for an IPv6 address, namely AAAA.

6.18 IPsec

The IP was designed in an academic environment where everybody trusted everybody else. The contents of packets are readable by anyone that has access to the physical layer. Since packets are routed by third parties as they progress from source to destination any router on the path can peek at the data contained therein. This is not so good as these days IP is used for all kinds of commercial and confidential data traffic and the intermediate routers are owned by many different third parties.

RFC2406 IPsec addresses the problems of secrecy and authentication. Secrecy is implemented by *encapsulating security payload*
RFC2402 (ESP), while authentication is achieved by *authentication header* (AH). Keys are managed by *internet key exchange*
RFC2409 (IKE) that runs over UDP.

AH authenticates connections, not users. You don't use AH to login, but rather to ensure that the remote host is really the CD shop it claims to be before you send your credit card details (encrypted by ESP), rather than just someone pretending to be that CD shop.

IPsec is not very widely employed. Difficulties include

1. key management: getting the public key from the destination, and ensuring it *is* the public key for the destination. DNSsec should help with this
2. routing: some routers make decisions based on the type of traffic, e.g., video or ftp. Encryption hides this, and so makes efficient routing harder
3. governmental controls over encryption technologies

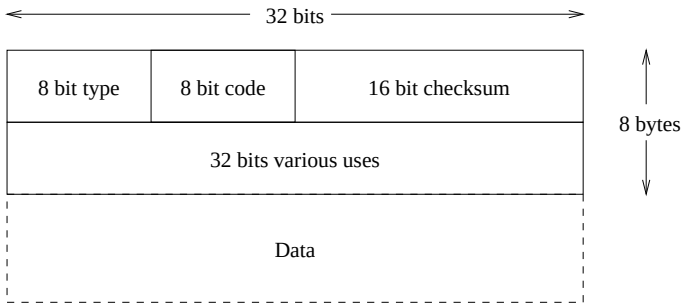


Figure 31: ICMP

Ether header	IP header	ICMP header	ICMP data	Ether trailer

Figure 32: ICMP in IP

7 The Internet Layer: ICMP

RFC792

Several places we have said things like “drop a packet and return an error message to the source”. How is this done? Well, the only mechanism we have for communication is to send packets, so the error message must itself be a packet. This is a normal IP packet, but has special contents. This kind of packet is called an *Internet Control Message Protocol* packet (ICMP).

The ICMP is used for general control of the network as well as for indicating errors. It is layered on top of IP, but is considered to be part of the Internet layer.

Recall that this is enclosed in an IP packet, enclosed in (say) an Ethernet frame.

The fields of an ICMP packet are

- Type. TTL expired; echo request; destination unreachable; etc.
- Code. Host unreachable; port unreachable; etc.
- Checksum.
- This field does different things for different ICMP types. This includes a 32 bit gateway address; a 16 bit identifier and a 16 bit sequence number for echo request and reply; etc.

ICMP packets are IP packets, and so are subject to the general foibles of IP, like being lost or duplicated.

ICMP messages are classified as either *query*, or *error*. For example, an ICMP echo request is a query, while a TTL expired is an error. ICMP errors are never generated in response to

1. an ICMP error message, e.g., if the TTL expires on a ICMP error packet
2. a datagram whose destination is a broadcast (or multicast) address
3. a datagram whose source is a broadcast (or multicast) address
4. a datagram whose link layer address is a broadcast address
5. any fragment other than the first.

This is to prevent *broadcast storms* of ICMP packets, where a single error can be multiplied into many ICMP packets.

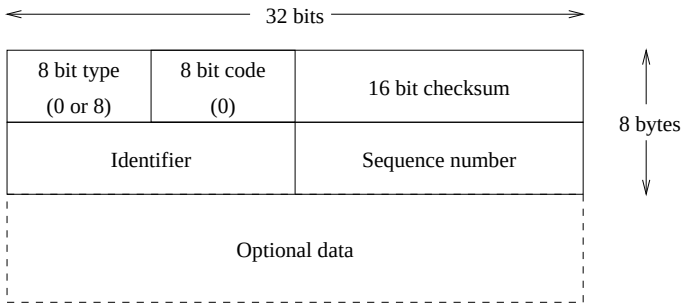


Figure 33: ICMP echo request and reply

7.1 Ping

There are a couple of clever ways we can exploit ICMP messages. Firstly to discover whether a machine is up, and secondly to determine the route a packet took to get to a machine.

Ping is a simple way to check whether a machine is alive or has crashed.

`/usr/sbin/ping -s www.yahoo.co.uk`
Hit `^C` to terminate. DON'T LEAVE PING RUNNING. This uses bandwidth and annoys a lot of people.

This sends an ICMP *echo request* packet. This is ICMP type 0, code 0. The required response from a host is to copy back the packet and its data in an ICMP *echo reply* packet (type 8, code 0).

The identifier field is some random number (often the Unix process number) so that the operating system can distinguish packets when multiple pings are running on the same machine. The sequence number starts at 0 and is incremented by 1 for each packet sent. This allows ping to determine if any packets were lost, reordered or duplicated.

When a ping echo response is received, the sequence number is printed, along with other useful information, such as the *round trip time* (RTT) of the packet.

```
mary % /usr/sbin/ping -s www.yahoo.co.uk
PING homerc.europe.yahoo.com: 56 data bytes
64 bytes from rc3.europe.yahoo.com (194.237.109.72): icmp_seq=0. time=160. ms
64 bytes from rc3.europe.yahoo.com (194.237.109.72): icmp_seq=1. time=154. ms
64 bytes from rc3.europe.yahoo.com (194.237.109.72): icmp_seq=2. time=176. ms
64 bytes from rc3.europe.yahoo.com (194.237.109.72): icmp_seq=3. time=159. ms
64 bytes from rc3.europe.yahoo.com (194.237.109.72): icmp_seq=4. time=161. ms
^C
----homerc.europe.yahoo.com PING Statistics----
5 packets transmitted, 5 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 154/162/176
```

There is variation in the RTT: this variation increases with the distance the packet travels. WANs give us a lot of variance.

There is an IP option *record route*. This attempts to trace the path of a packet through the Internet by saving the IP addresses of machines as the packet passes by.

All IP options have a *code* to tell us what kind of option we have, and most have a *length* field. The code for record route is 7, and the length is dependent on the number of IP addresses recorded. The *pointer* field tells us what offset to write the next IP address (starting at 4, it goes 4, 8, 12, etc.).

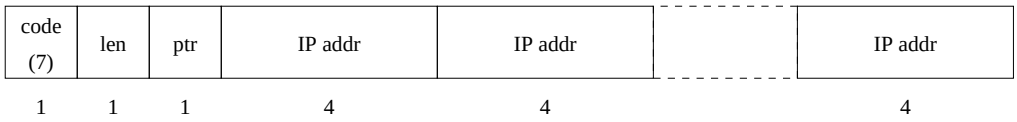


Figure 34: IP record route

The options field in an IP packet can be up to 60 bytes (using a 4 bit field to denote 4 byte words). Thus we can pack up to 9 addresses (taking the code and other fields into account). This is not very much. In the early ARPANET this was ample, but no longer. If the packet requires more than 9 hops, only the first 9 are recorded.

```
mary:3 % /usr/sbin/ping -s -R -v www.yahoo.co.uk
PING homerc.europe.yahoo.com: 56 data bytes
64 bytes from rc3.europe.yahoo.com (194.237.109.72): icmp_seq=0. time=168. ms
  IP options:  <record route> 138.38.29.254, bath-gw-1.bwe.net.uk
 (194.82.125.198), man-gw-2.bwe.net.uk (194.82.125.210), bristol.
bweman.site.ja.net (146.97.252.102), south-east-gw.bristol-core.j
a.net (146.97.252.62), south-east-gw.ja.net (193.63.94.50), 212.1
.192.150, se-uk.uk.ten-155.net (212.1.192.110), se-aucs.se.ten-155
.net (212.1.194.25)
64 bytes from rc3.europe.yahoo.com (194.237.109.72): icmp_seq=1. time=165. ms
  IP options:  <record route> 138.38.29.254, bath-gw-1.bwe.net.uk
 (194.82.125.198), man-gw-2.bwe.net.uk (194.82.125.210), bristol.
bweman.site.ja.net (146.97.252.102), south-east-gw.bristol-core.j
a.net (146.97.252.62), south-east-gw.ja.net (128.86.1.50), 212.1.
192.150, se-uk.uk.ten-155.net (212.1.192.110), se-aucs.se.ten-155
.net (212.1.194.25)
64 bytes from rc3.europe.yahoo.com (194.237.109.72): icmp_seq=2. time=167. ms
  IP options:  <record route> 138.38.29.254, bath-gw-1.bwe.net.uk
 (194.82.125.198), man-gw-2.bwe.net.uk (194.82.125.210), bristol.
bweman.site.ja.net (146.97.252.102), south-east-gw.bristol-core.j
a.net (146.97.252.62), south-east-gw.ja.net (128.86.1.50), 212.1.
192.150, se-uk.uk.ten-155.net (212.1.192.110), se-aucs.se.ten-15
5.net (212.1.194.25)
^C
----homerc.europe.yahoo.com PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 165/166/168
```

Record route can be used on any IP packet, but is is mostly useful when pinging.

Some people dislike being pinged as they believe it could be a security hole, and shut off the normal ICMP response. Thus it could appear that a machine is not working due to failed pings, but it it actually up and running.

7.2 Traceroute

The IP option to record route has limited utility, since it can save only 9 hosts, and shows nothing if the destination machine is down. Traceroute is a clever means to discover the route a packet is taking that solves both of these problems. Traceroute works by deliberately generating errors and examining the ICMP packets returned.

Traceroute sends UDP packets (transport layer packets, see later) to the selected destination that have a artificially small small time-to-live. When the TTL drops to zero, and ICMP *TTL exceeded* packet is generated and returned to the source.

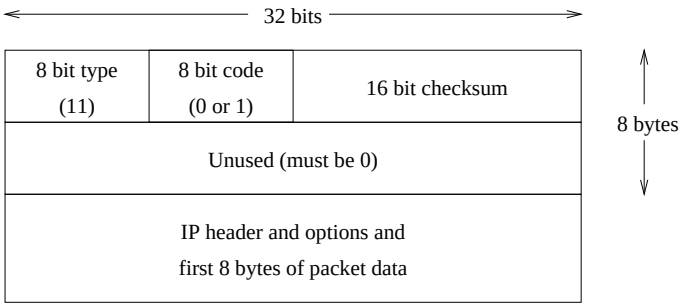


Figure 35: ICMP TTL exceeded

1. Send a packet with TTL set to 1, but with the destination address of the machine we wish to probe.
2. This packet reaches the first gateway/router and the TTL is decremented to 0. The router drops the packet, returns an ICMP TTL exceeded.
3. This reaches the source, that notes from where the packet originated, viz., the router.
4. Send a packet with TTL set to 2. This gets to the next router before the TTL gets to 0, and the ICMP response identifies the second router.
5. Repeat with TTL 3, 4, and so on, until the packet manages to reach the destination. At each state we get an ICMP error telling us of the routers the packets passed through.

```
wiz % traceroute mary.bath.ac.uk
traceroute to mary.bath.ac.uk (138.38.32.14), 30 hops max, 46 byte packets
 1  136.159.7.1 (136.159.7.1)  0.779 ms  1.131 ms  0.642 ms
 2  136.159.28.1 (136.159.28.1)  1.369 ms  0.910 ms  1.489 ms
 3  136.159.30.1 (136.159.30.1)  2.339 ms  1.937 ms  0.988 ms
 4  136.159.251.2 (136.159.251.2)  1.458 ms  1.071 ms  1.831 ms
 5  192.168.47.1 (192.168.47.1)  1.434 ms  1.554 ms  1.008 ms
 6  192.168.3.25 (192.168.3.25)  29.192 ms  30.094 ms  25.374 ms
 7  REGIONAL2.tac.net (205.233.111.67)  25.413 ms  33.002 ms  32.677 ms
 8  * * *
 9  * 117.ATM3-0.XR2.CHI6.ALTER.NET (146.188.209.182)  82.403 ms  58.747 ms
10  190.ATM11-0-0.GW4.CHI6.ALTER.NET (146.188.209.149)  56.376 ms  67.898 ms  73.462 ms
11  if-4-0-1-1.bb1.Chicago2.Teleglobe.net (207.45.193.9)  66.853 ms  46.089 ms  44.670 m
12  if-0-0.core1.Chicago3.Teleglobe.net (207.45.222.213)  48.817 ms *  75.093 ms
13  if-8-1.core1.NewYork.Teleglobe.net (207.45.222.209)  106.198 ms  94.249 ms  73.375 m
14  ix-5-3.core1.NewYork.Teleglobe.net (207.45.202.30)  75.286 ms  89.873 ms  98.789 ms
15  us-gw.ja.net (193.62.157.13)  143.686 ms  159.212 ms  166.020 ms
16  external-gw.ja.net (193.63.94.40)  172.803 ms  189.216 ms  191.260 ms
17  external-gw.bristol-core.ja.net (146.97.252.58)  206.403 ms  185.438 ms  192.989 ms
18  bristol.bweman.site.ja.net (146.97.252.102)  196.685 ms  206.221 ms  183.763 ms
19  man-gw-2.bwe.net.uk (194.82.125.210)  197.968 ms *  174.809 ms
20  bath-gw-1.bwe.net.uk (194.82.125.198)  209.307 ms  221.512 ms  199.168 ms
21  * * *
22  mary.bath.ac.uk (138.38.32.14)  250.670 ms *  186.400 ms
```

Some versions of `ping` support setting the TTL, e.g., Solaris uses `-t`. This can be used to mimic `traceroute` by hand.

ICMP errors often contain the IP header and 8 bytes of data of the datagram that caused the problem. This is so the source machine can match up the ICMP datagram with the one that caused it. Eight bytes is just enough to contain the interesting parts of the header of the next layer (TCP or UDP).

RFC1812 In fact, it is now recommended that the ICMP error contains "as many bytes of the original datagram as possible without the length of the ICMP datagram exceeding 576 bytes". (Recall that 576 is the minimum segment maximum.)

There are many things that can happen in a traceroute:

- three `*`s in line 8: some routers return an ICMP error with a TTL that was whatever was left in the original datagram. This is guaranteed not to reach us
- if the last half of the routers are `*`s, this means that the destination has this bug: the TTL is ramped up until it is double the hop length of the route and then the ICMP reply can reach us. The destination is really only half the advertised number of hops away
- another possibility (on longer routes) is that the router is setting a TTL too small to reach us
- a third possibility is simply that the router refuses to send ICMP errors for TTL exceeded
- a `*` before the machine name in line 9: the DNS name lookup did not return a name before the ICMP error came back
- sometimes the same line is repeated twice: this is because some routers forward datagrams with TTL of 0. This is a bug.

There are many bugs out there in real routers!

8 Routing IP

We have already alluded to routing tables. Small tables can be set up by hand, and most host's tables only contain two route of interest: to the local network for local traffic, and to the gateway for non-local traffic. We now look a little more closely at routing tables.

A *static route* is one added by hand, typically by means of the `route` command. For example,

```
route add default gw 138.38.103.254
```

to add a default route to a gateway. Different operating systems have different arguments to `route`, but the principles are similar. The command `netstat -r -n` displays the routing table.

```
Destination     Gateway         Flags  Refcnt Use    Interface
140.252.13.65   140.252.13.35   UGH    0      171    le0
127.0.0.1       127.0.0.1       UH     1      766    lo0
140.252.1.183   140.252.1.29    UH     0      0      sl0
default         140.252.1.183   UG     1      2955   sl0
140.252.13.32   140.252.13.33   U      8      99551  le0
```

The flags are

- U. The route is up (i.e., working)
- G. The route is to a gateway/router. Without a G the route is directly connected to the network on the interface.

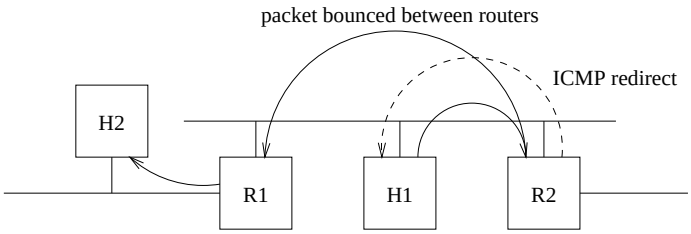


Figure 36: ICMP Redirect

- H. The route is to a host. The Destination is a host address, not a network address.
- D. The route was created by ICMP *redirect*.
- M. The route was modified by ICMP *redirect*.

The `Refcnt` indicates the number of (TCP) connections currently using this route. The `Use` is the number of packets that have passed via this route.

8.1 ICMP Redirect

Sometimes the routing tables are not perfectly set up.

Suppose host H1 wants to send to host H2, but H1's route table directs all packets to router R2. When the packets reaches R2, R2 looks at its table and realises the packet should be forwarded back through the interface it came in on. This is an indication to R2 that H1's table needs improving. R2 forwards the packet to R1, but also sends an ICMP *redirect* message to H1. H1 uses the information in the message to update its table (which is marked by the D or M flag). Next time H1 wants to send to H2 it will send the packet directly to R1.

This allows small improvements in routing to accumulate over time.

RFC1256 Another mechanism of generating router tables involves *router advertisement* using ICMP *Router Discovery* messages. A host can broadcast a *router solicitation* message, and one or more routers can respond with messages containing routes via those routers.

8.2 Dynamic Routing Protocols

Dynamic routing is the passing of routing information between routers. ICMP redirects are a limited form of dynamic routing, but they are generally classed as static routes.

Routers swap routes between themselves using some protocol. There are several protocols, including the *Routing Information Protocol* (RIP), the *Open Shortest Path First* (OSPF) protocol, and the *Border Gateway Protocol* (BGP).

The Internet uses many routing protocols. It is composed of a collection of *Autonomous systems* (ASs), each of which is administered by a single entity, e.g., a university or a company. Each AS chooses its own routing protocol to direct packets within the AS. This is an *interior gateway protocol* (IGP) (or *intradomain routing protocol*), for example RIP or OSPF.

Between ASs run *exterior gateway protocols* (EGP) (or *interdomain routing protocols*), for example BGP.

Typically a router will run a *router daemon*, a program whose sole purpose is to exchange routing information and update the routing table. For example, `routed` talks RIP, and `gated` talks RIP, OSPF and BGP.

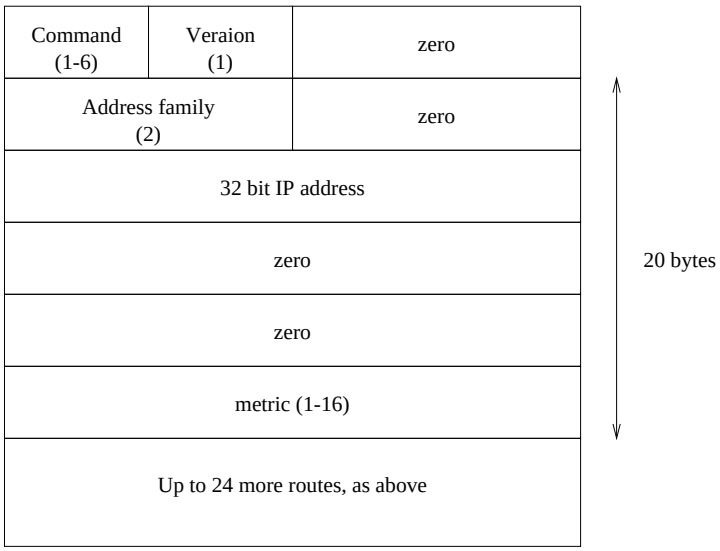


Figure 37: RIP packets

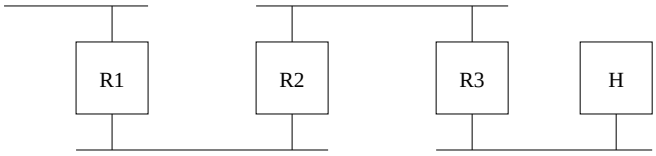


Figure 38: RIP routing

8.2.1 RIP

RFC1058

RIP is one of the most widely used protocols on small to mid-sized networks.

A RIP message is encapsulated within a UDP packet. A *command* of 1 is a request, while 2 is a reply. Other values are not generally used. The *version* is generally 1, though an newer version of RIP has 2.

The next 20 bytes specify a route: the *address family* is 2 for IP addresses; an IP address; and a *metric*. The limit of 25 is to keep the total packet size less than 512 bytes, a size that should never need to be fragmented.

The metric is the number of hops to the specified address.

When `routed` starts, it broadcasts a request (command value 1) out on all interfaces with address family 0 and metric 16. This is a "send me all your routes" message. When a router receives such a RIP request, it replies with all its table's entries in one or more RIP replies.

Otherwise a RIP request is a request for a route to a specific address (or several addresses). A response to this is our metric for our route, or 16 to signify infinity or "no route".

When a response is received, we can update our routing table appropriately. Our metric is the received metric plus 1 for the hop to the router that replied. If a new route arrives with a smaller metric than an existing route we can replace the old route with the new one: this happens if a new path arises that is shorter than the old one. A shorter path is deemed better.

RIP also sends a chunk of the current routing table every 30 seconds to all its neighbour routers. Routes are timed out if they haven't been reconfirmed for 3 minutes (six updates). The metric is set to 16, but the route is not deleted for 60 seconds. This ensures the invalidation is propagated.

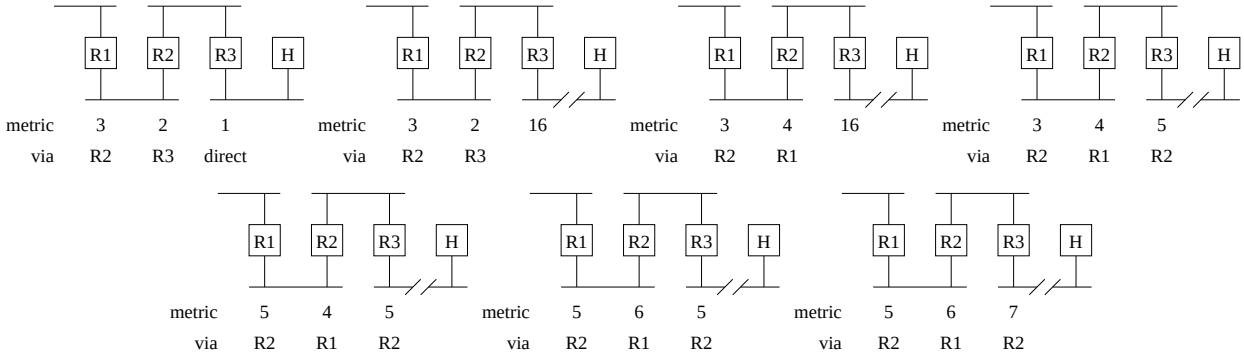


Figure 39: Slow Convergence

R3 has a route to H with metric 1. A RIP message between R2 and R3 allows R2 to learn there is a route to H via R3 with metric 2. Then a RIP message between R1 and R2 allows R1 to learn there is a route to H via R2 with metric 3. Notice that R2 gets a message from R1 saying it knows route of metric 3 to R3, but it ignores this as R2 already know a better route.

There are a few problems with RIP. Firstly it is not suitable for use between ASs as it passes on too much internal network information, and it is ignorant of subnetting. Furthermore, the limit of 15 on the metric is too small for the Internet.

Another problem is the time RIP takes to settle down after a change, for example when a new router is added, or a router crashes: this can be several minutes. This is called *slow convergence*. Suppose that the link from R3 to H breaks. R3 starts sending RIP messages with metric 16 (infinity) for the route to H. R2 picks this up. But now R2 gets a message from R1 with a metric of 3. So R2 replaces its route to H to go via R1 with metric 4. R1 now sees that R2 has a route to H metric 4, and so must updates its metric to 5 in its table. This bounces back and forth between R1 and R2 until eventually they both reach infinity at 16. This takes about 4 minutes. Meanwhile, real data packets are also being bounced between R1 and R2 adding to the confusion. The problem is that R2 does not know that R1's advertised route is actually via R2.

The use of a hop count as a metric is a bit simplistic: it is not always the case that the fewest hops is the best route to take. Other considerations like network speed, network bandwidth, and cost should be taken into account.

RFC1388 Nevertheless, RIP is quite suitable for small to medium networks. RIP version 2 addresses the questions of EGPs and subnets, but OSPF is now more popular.

8.2.2 OSPF

RFC2328

OSPF is a newer routing protocol that fixes the problems of RIP. OSPF is a *link-state* protocol, in contrast to RIP which is a *distance-vector* protocol. In essence, RIP measures simple hop counts while OSPF tests the state of neighbouring routers and passes this information through the AS. Each router takes this state information and builds its own router table. This allows OSPF to *converge* faster than RIP in the case of a change in the network.

OSPF is layered directly on top of IP (RIP is on UDP on IP), and has many advantages over RIP.

1. OSPF can have different routes for different IP types of service.
2. An interface is assigned a *cost*. This is a number that is computed from anything relevant, e.g., reliability, throughput, round-trip-time.
3. If more than one equal cost route exists, OSPF shares traffic equally between them (*load balancing*).
4. OSPF understands subnets.

5. A simple authentication scheme can be used to prevent spoofing of routes.
6. OSPF uses multicast rather than broadcast, so only routers that are interested have to listen to OSPF traffic.

8.2.3 BGP

RFC1467

BGP is an EGP, and so is used for routing between ASs. An EGP has a different problem to an IGP: now the problem is to route between ASs rather than networks or hosts, and politics becomes a dominant factor.

In BGP ASs are classified into three types:

1. A *stub* AS. This has only one connection to any other AS, and only carries local traffic. Bath University is such.
2. A *multihomed* AS. This has more than one connection, but refuses to carry anyone else's traffic.
3. A *transit* AS. This has more than one connection, and will carry traffic from one AS to another (usually with certain policy restrictions).

BGP allows *policy based routing*, and such polices are determined by the administrator of an AS, and they can be political, economic or anything else.

For example, there is a Canadian law to the effect that all traffic beginning and ending in Canada must not leave Canada at any point. So no transit AS can pass traffic out of the country under such circumstances.

BGP is layered on top of TCP, and is a distance-vector protocol like RIP, but rather than giving simple hop counts BGP passes on the actual AS-to-AS routes. This fixes some of the problems associated with distance-vector protocols. On the other hand, ASs do not change very much, so there is not a big problem with slow convergence.

9 Broadcasting and Multicasting

There are three types of IP addresses: *unicast*, *broadcast* and *multicast*. Unicast we are familiar with: an address specifies a single machine. Broadcast is similar to broadcast at the link layer: a single packet goes to every machine on the network, such as is used by ARP. Multicast is something different: a single packet goes to several selected machines (more than one, and probably less than all).

Notice that not all network layers support broadcast or multicast. For example, broadcast on a PPP link is not terribly useful. Use `ifconfig` to see if your network supports multicast.

9.1 Broadcast

Broadcasting is conceptually simple: have a single packet that is read by all machines. This is better than sending unicast packets individually to all machines. But what do we mean by "all"? Clearly some limitation on broadcast packets is needed, else the entire Internet would be permenantly flooded.

- Limited Broadcast. The address 255.255.255.255 send to all hosts on the local network. A packet with such a destination is never forwarded by a router.
- Net-Directed Broadcast. When the host part of the network address is all ones. E.g., n.255.255.255 for a class A, or $n.m$.255.255 for a class B. Routers forward such packets if they are connecting two subnets in the same network.

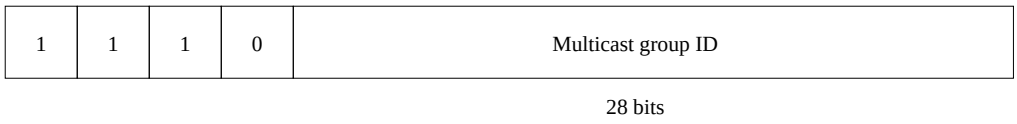


Figure 40: Class D Multicast Addresses

- Subnet-Directed Broadcast. Now we broadcast to all hosts on the local subnet. Address has a valid subnet part with the host part all ones. This is the most common form of broadcast.
- All-Subnets-Directed Broadcast. No longer used, and replaced by multicast. Sends to a collection of subnets.

In fact, these broadcasts (apart from subnet-directed) are essentially obsolete due to CIDR.

If you want to annoy a lot of people, try `ping` to a broadcast address.

9.2 Multicast

RFC1112

Multicast is used when we want to send the same packet to several hosts. For example, when streaming audio over the Internet we could unicast 100 packets to 100 people listening. Better, as there is less network traffic, we could broadcast one packet that is read by all hosts. Unfortunately, this must be read and processed by *every* host whether that host wants the audio stream or not. Better still, we can *multicast* a single packet that is read just by those 100 machines, leaving all other hosts alone.

For example, multicast is used by OSPF to communicate routing data: a single packet informs all routers. Broadcast would inflict this data on *all* machines, not just the routers.

RFC2908 Special multicast addresses are used, and *multicast groups* are formed of those machines that are interested in receiving packets from a given source. E.g., a group to listen to Radio 1.

A *multicast group id* is a 28 bit number (nearly 270 million groups), with no further structure. Multicast addresses fall in the range 244.0.0.0 to 239.255.255.255. The set of hosts listening to a particular IP multicast address are called a *host group*. A host group can span many networks, and there is no limit on the number of members.

Some group addresses are assigned as well-known addresses by IANA: these are the *permenant host groups*. For example, 224.0.0.5 for OSPF routers and 224.0.0.1 for "all multicast aware hosts on the subnet".

RFC2236 The process of joining and leaving host groups is governed by the *Internet Group Management Protocol* (IGMP). Multicast over LANs is reasonably simple, while the use of multicasting over WAN is still a subject of experimentation (a multicast packet must be copied and sent to each network that has joined the host group). The *multicast backbone* (MBONE) is an step in this direction.

9.2.1 Multicast and Ethernet Addresses

Many Ethernet cards support multicasting in hardware. What this means is that the card can be aware of which multicast groups the host requires, and can filter out those multicast packets not is those groups. This hardware assist means less work for the IP layer to do. To support this, special Ethernet hardware addresses are used for multicast packets.

Addresses in the range 01:00:5e:00:00:00 to 01:00:5e:7f:ff:ff are reserved for multicasting. This is 23 bits of address space. The lowest 23 bits of an IP multicast address is mapped directly into this space. There are 5 bits of the IP address left over, so this means that 32 different IP multicast addresses map to the same Ethernet multicast address. This means that the IP layer will have to do some filtering after all: it can't all be done by the hardware. On the

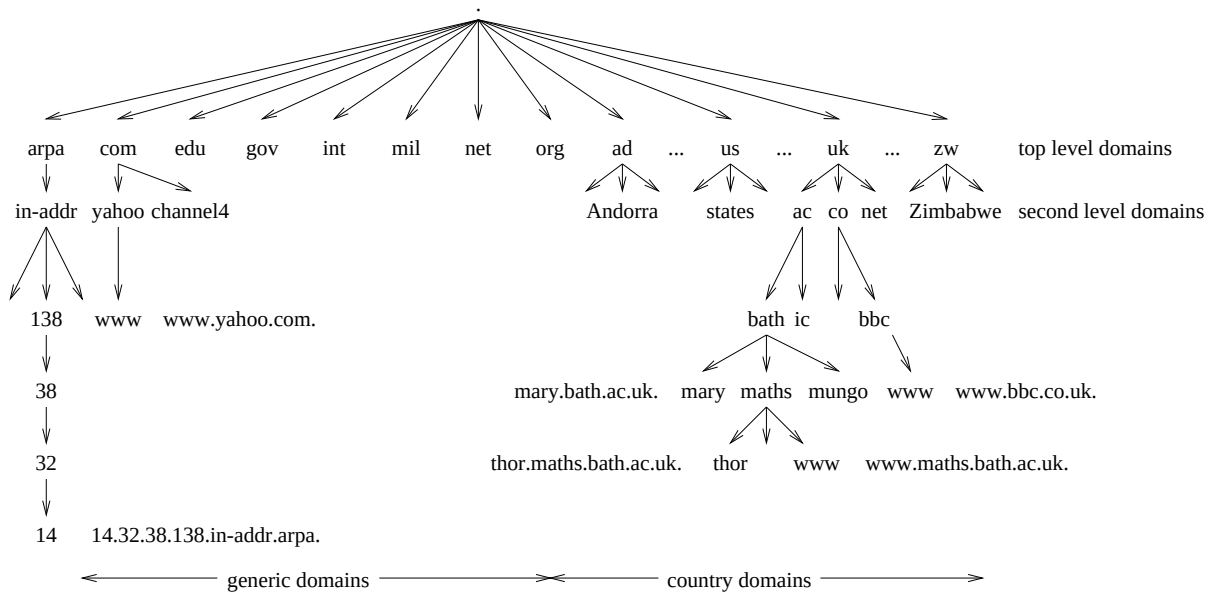


Figure 41: DNS hierarchy

other hand, it will be rare that there will be a clash. Despite this problem, this combination of hardware plus software filtering of multicasting addresses is still better than broadcasts.

10 The Domain Name System

RFC1034
RFC1035

DNS is the means by which we can convert names like `mary.bath.ac.uk` to IP addresses like 138.38.32.14. The benefit of having human comprehensible names rather than a jumble of numbers is clear. We *could* learn numbers to refer to machines (we do learn telephone numbers), but names tend to stick much better in the mind.

In the early days of the Internet, all the machines could keep a table of every name and IP address that existed. See `/etc/hosts`. Soon, though, this became untenable as the Internet grew. The DNS was developed as a hierarchical system to *resolve* names, and is *distributed*: that is, no one machine on the Internet knows all the names of all the machines, but the mapping is spread about over very many machines.

10.1 The Hierarchy

The DNS hierarchy is a tree with root at the top. The root has name "." (dot). Other nodes in the tree have *labels* of up to 63 characters. A *fully qualified domain name* (FQDN) is a sequence of labels terminated by a dot, e.g., `mary.bath.ac.uk.`. A name without a terminating dot is assumed incomplete, and may be completed by the DNS software, e.g., `mary` is completed on BUCS hosts to be `mary.bath.ac.uk.`. Furthermore the incomplete name `mary.bath.ac.uk` is completed to the same.

On the Maths network, `mary` would be expanded as `mary.maths.bath.ac.uk.` then, when it found that no such name exists, `mary.bath.ac.uk.`. The software tries appending each trailing substring of more than two labels until it finds a name that exists. When BUCS is given `mary.bath.ac.uk` it first tries to see if `mary.bath.ac.uk.bath.ac.uk.` is a name, then `mary.bath.ac.uk.`. Alternate strings can be given in `/etc/resolv.conf`.

The `arpa` domain is used for getting names from numbers. For a long while there were just seven *generic domains*, each three characters long. These include labels like `com`, `org` and `edu`. These mostly refer to machines in the USA (because they forgot about the rest of the world when inventing DNS), but some, e.g., `com` are used worldwide. More recently, new labels like `biz` and `info` have been added.

The two character names are *country domains*, and refer to the appropriate country. These names come from ISO 3166, the list of official country name abbreviations, except that the UK uses `uk` instead of the ISO `gb`.

Each level of the tree represents different management and responsibility for the names. The *top level domains* (TLDs) are managed by IANA, who has delegated responsibility to The Internet Corporation for Assigned Names and Numbers (ICANN). To get a new name at this level, you would have to apply to ICANN—but they wouldn't give you one, as this level is essentially fixed politically.

Every other layer is managed by some other entity. For example labels under `uk` are managed by the UK Network Information Centre (NIC), run by a company called Nominet. Again, this level is fixed, and it is hard to get a name at this level.

The beauty of the DNS is the shared responsibility. The labels under `ac.uk` are managed by the United Kingdom Education and Research Networking Association (UKERNA). You have some chance of getting a name from these people as long as you are connected with the UK academic community. Labels under `co.uk` happen also to be managed by Nominet: you can readily get a label here for the right price. Labels under `bath.ac.uk` are administered by BUCS.

Another thing to note that names administered by one authority can refer to machines anywhere in the world: `bill.acme.com` might be in Rangoon, while `ben.acme.com` might be in Tunbridge Wells.

`.co.uk`	for commercial enterprises
`.org.uk`	for organisations
`.ltd.uk`	for UK limited companies
`.plc.uk`	for UK public limited companies
`.net.uk`	for Internet service providers
`.sch.uk`	for UK schools
`.ac.uk`	for academic establishments
`.gov.uk`	for government bodies
`.nhs.uk`	for NHS organisations
`.police.uk`	for UK police forces
`.mod.uk`	for Ministry of Defence establishments

Labels under `bath.ac.uk` are managed by BUCS, while labels under `cs.bath.ac.uk` are managed by the Department of Computer Science. Other countries manage their parts of the tree differently. Germany does not have an `ac.uk` equivalent, but rather we get names like `uni-paderborn.de`. This is a shame because they lose the ability to distribute responsibility for names.

A *zone* is a subtree that is administered separately, `bath.ac.uk`, for example. A zone can have sub-zones (`cs.bath.ac.uk`).

The authority for a zone must set up some *name servers*. A name server is (a program on) a machine that has a database of the labels for that zone. Names are added or deleted at this level by changing this database. To provide resiliency, there must be a *primary name server*, and one or more *secondary name servers* in case the primary goes down. The primary gets its information from the database, while the secondaries get theirs from the primary using *zone transfers*. This is just the copying of zone information. The secondary queries the primary every 3 hours typically.

So if a machine requests a name lookup from the name server, it can hand it an *authoritative* response. The server `tamarin.bath.ac.uk` is primary for the `bath` zone. See `/etc/resolv.conf`.

10.2 Recursive Lookup

If the request is for a name outside the zone, the name server must work a bit harder. It contacts a *root name server*. This is one of (currently) 13 machine dotted about the world that are responsible for the TLDs (the root zone). Our name server must have the IP addresses of these in a file somewhere. Current name servers are named `a.root-servers.net` to `m.root-servers.net`.

```
A.ROOT-SERVERS.NET      internet address = 198.41.0.4
B.ROOT-SERVERS.NET      internet address = 128.9.0.107
C.ROOT-SERVERS.NET      internet address = 192.33.4.12
D.ROOT-SERVERS.NET      internet address = 128.8.10.90
E.ROOT-SERVERS.NET      internet address = 192.203.230.10
F.ROOT-SERVERS.NET      internet address = 192.5.5.241
G.ROOT-SERVERS.NET      internet address = 192.112.36.4
H.ROOT-SERVERS.NET      internet address = 128.63.2.53
I.ROOT-SERVERS.NET      internet address = 192.36.148.17
J.ROOT-SERVERS.NET      internet address = 198.41.0.10
K.ROOT-SERVERS.NET      internet address = 193.0.14.129
L.ROOT-SERVERS.NET      internet address = 198.32.64.12
M.ROOT-SERVERS.NET      internet address = 202.12.27.33
```

Suppose we want to find the IP address of `news.bbc.co.uk`. Our name server `tamarin` doesn't have responsibility for the `bbc` zone. So `tamarin` contacts a root server with the question "who is responsible for the `uk` domain?" The root server answers with `ns1.nic.uk` or somesuch.

Now `tamarin` asks `ns1.nic.uk` "who is responsible for the `co.uk` domain?" and gets `ns1.nic.uk` (again). Next, it asks `ns1.nic.net` for `bbc.co.uk`, and gets `ns.bbc.co.uk`. Finally, `tamarin` asks `ns.bbc.co.uk` for `news.bbc.co.uk` and gets the IP address 194.130.56.40. At last `tamarin` can hand the IP address back to us.

Of course, these responses are cached by `tamarin` so it doesn't have to go through a complete lookup every time. Each response has a *time to live* attached that indicates how long the server should keep the information before asking again.

```
mary % nslookup
Default Server:  tamarin.bath.ac.uk
Address:  138.38.32.3

> news.bbc.co.uk
Server:  tamarin.bath.ac.uk
Address:  138.38.32.3

Non-authoritative answer:
Name:    newswww.bbc.net.uk
Address:  194.130.56.40
Aliases:  news.bbc.co.uk
```

The "Non-authoritative" answer in the above is an indication that in this instance `tamarin` didn't do a full lookup, but re-used a result that it had discovered earlier. It is not authoritative since the zone authority may have changed the IP address for that name since we last looked, but this is generally unlikely. The first time a machine looks up a name you will get an authoritative answer.

More than one name can map to the same IP address: `newswww.bbc.net.uk` is an alternate name for `news.bbc.co.uk`. In fact, `newswww.bbc.net.uk` is the *canonical name* or CNAME for this machine, while `news.bbc.co.uk`

is an *alias*. Aliases are useful to give mnemonic names to machines, e.g., `www.bath.ac.uk` is an alias for `jess.bath.ac.uk`. If we decide to run the Web server on a different machine, we can transfer the alias to the new machine, and nobody else needs to be aware.

Conversely, one name can map to more than one IP address: `www.yahoo.com` has

```
Name:    www.yahoo.com
Addresses:  216.32.74.50, 216.32.74.55, 204.71.200.74, 204.71.200.67
          204.71.202.160, 216.32.74.52, 216.32.74.54, 204.71.200.68,
          216.32.74.51, 216.32.74.53, 204.71.200.75
```

which is 11 different machines.

When this happens, the IP address to use is taken in a round-robin fashion. This is usually done to balance load: different people will be trying to get the same web page from different machines. Usually, all the machines are configured identically so it doesn't matter which machine we actually contact. Notice that these machines can be entirely independent and distributed throughout the world!

To find the authority for a domain, use `nslookup` and use `set q=soa` (start of authority). Using `set q=ns` to find an authoritative name server.

10.3 Reverse Lookup

There is another branch of the DNS tree with TLD `arpa`. This branch allows us to do the reverse lookup of IP address to DNS name. This is very useful for determining the source of a packet when you only know its IP address.

A DNS name has the most significant part last, e.g., the `uk` in `mary.bath.ac.uk`, and we delegate downwards from that end. An IP address has its most significant part first, e.g., the 138 in 138.38.32.14, and the values are delegated from that end.

In the early days the UK academic community used the order `uk.ac.bath.mary` to be consistent. But the rest of the world disagreed, and eventually we changed.

Looking up the address `14.32.38.138.in-addr.arpa.` will reveal that the IP address 138.38.32.14 belongs to `mary.bath.ac.uk`. Again, `tamarin` is the authority for the `38.138.in-addr.arpa` zone.

RFC2050 CIDR causes complications here as networks are no longer necessarily divided on byte boundaries.

Setting up a name server authority properly requires the management of two databases: one from name to number, and the other the reverse. Some people forget to set up the number to name map, or change the name to number map and forget to update the reverse, etc. This causes all sort of problems (e.g., sometimes reverse lookup is used in authentication of connections).

```
> set q=ptr
> 14.32.38.138.in-addr.arpa.
Server:  tamarin.bath.ac.uk
Address:  138.38.32.3

14.32.38.138.in-addr.arpa       name = mary.bath.ac.uk
```

Actually a name request for 138.38.32.14 is switched by `nslookup` to a PTR request for `14.32.38.138.in-addr.arpa`.

10.4 Other Data

The DNS can give you more than just IP addresses. Several types of *record* have been defined.

Name	Type	
A	1	IP address
AAAA	28	IPv6 address (128 bit)
NS	2	Authoritative name server
CNAME	5	Canonical name
SOA	6	Start of Authority
PTR	12	pointer (IP address to name lookup)
HINFO	13	host info
MX	15	email exchange
AXFR	252	zone transfer
ANY	255	all records

There are many others: about 50 in total. For example, `set q=ptr` in `nslookup` to fetch names from the `in-addr.arpa` domain.

HINFO is some snippet of information about the machine, e.g., operating system. Not many people publish this kind of information.

Mail exchange, MX, gives the IP address of one of our machines that will accept email.

```
> set q=mx
> maths.bath.ac.uk
Server:  tamarin.bath.ac.uk
Address:  138.38.32.3

maths.bath.ac.uk        preference = 10, mail exchanger = pat.bath.ac.uk
maths.bath.ac.uk        preference = 10, mail exchanger = mercury.bath.ac.uk
maths.bath.ac.uk        preference = 5, mail exchanger = mailrouter.maths.bath.ac.uk
```

One of the hosts with the smallest preference is contacted first. If that host is down, try the next. Notice that no machine with the name `maths.bath.ac.uk` actually exists, but can still send email to `rjb@maths.bath.ac.uk`. This allows us to dedicate a single mail server to serve a large number of people using a variety of different machines.

NS gives an authoritative name server for the domain.

AXFR is for a zone transfer, i.e., transfer of a zone's records, probably from a primary name server to a secondary.

ANY fetches all available records associated with a name.

10.5 Packet Format

The message format for DNS has a fixed 12 byte header, followed by four variable length fields.

The *identification* is a number selected by the client in the query message and is returned by the server in the reply. This allows the client to match up queries with responses if there are several in flight simultaneously.

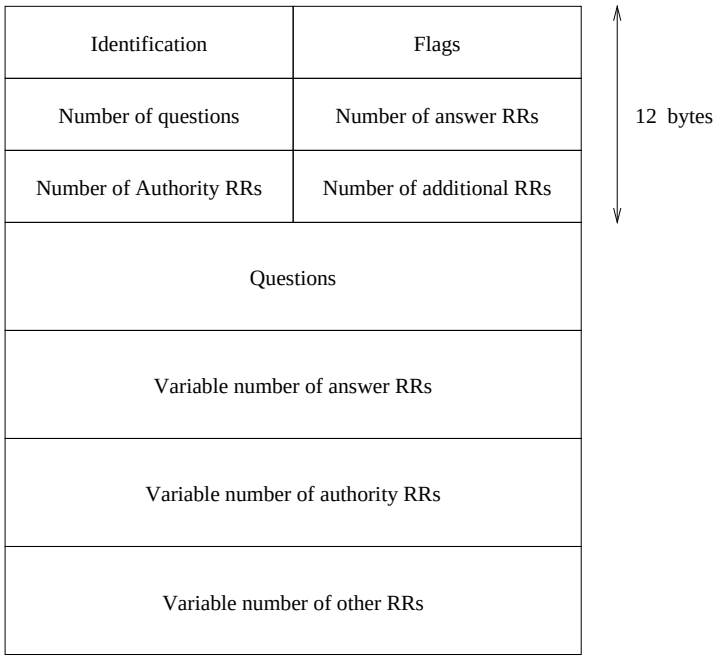


Figure 42: DNS Format

Figure 43: DNS Header Flags

The flags are

- QR. This bit is 1 for a query, 0 for a response
- Opcode. Usually 0 for a standard request, but can be other values
- AA. This bit is set on an authoritative answer
- TC. *Truncated*: couldn't fit the reply within 512 bytes (see later)
- RD. *Recursion desired*: the name server should do the recursive lookup. Otherwise, the name server returns a list of other name servers for the client to contact (an *iterative lookup*). This bit is normally set.
- RA. *Recursion available*: this name server can do a recursive lookup. Normally set.
- Rcode. A four bit return code. 0 is no error, while 3 is *name error*, a response from an authority saying the requested name does not exist.

The next four fields give the numbers of each type of *resource records* (RRs) that follow. Usually these are 1, 0, 0, 0 for a request and 1, 1, 0, 0 for a reply (the question is returned with the answer).

10.5.1 Query

The question format starts with the name we want to resolve. It is a sequence of one or more labels, where a label is stored as a single byte containing the number of characters in the label followed by those characters. The name is terminated by a 0 byte. Recall that the length of a label is not more than 63 characters.

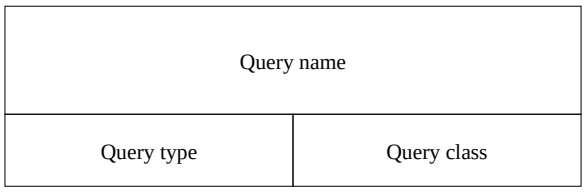


Figure 44: DNS Question Format

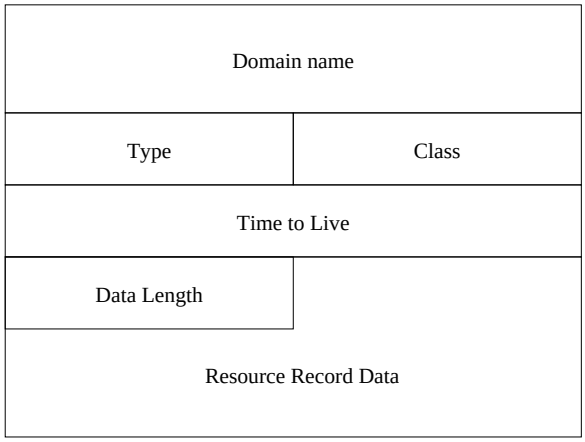


Figure 45: DNS RR Format

`4mary4bath2ac2uk0`

The *query type* is a number that specifies A, or AAAA, or CNAME, etc. The *query class* is normally 1, denoting IP address. A few others exist.

10.5.2 Response

- The domain name is the same as for the question field.
- Type and Class are as before.
- The *time to live* is a time, in seconds, that the name server should cache this data. A popular value is 2 days. When the TTL expires, the name server should re-query the authority for this data.
- The resource data is given as a length, followed by the data. The format of the data varies according to the type. For example, for an A reply this is just 4 bytes of IP address.

10.6 Other Stuff

DNS runs over both UDP and TCP. Typically UDP is used for speed, but there is a twist. To avoid possible fragmentation, a DNS server will never return an answer in a UDP datagram larger than 512 bytes. Instead, the response has the TC bit (*truncated*) set. Then the two machines start up a TCP connection and try again using multiple segments.

The DNS hierarchy allows machine names at any level: for example `channel4.com` is both a zone name, and a machine name. As a machine name it happens to resolve to the same IP address as `www.channel4.com`.

A simple form of compression can be used if the RRs contain repetitive data, e.g., the same root repeated many times.

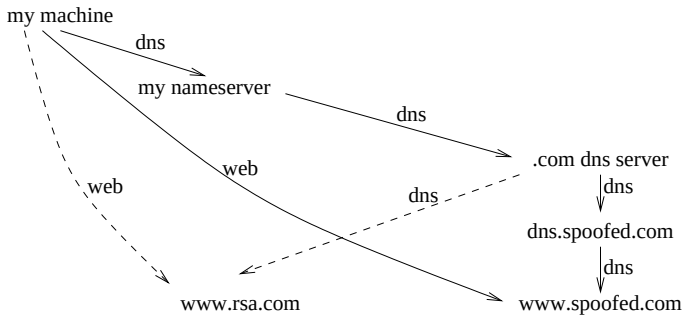


Figure 46: RSA Web Page Spoof

Also zone transfers are done using TCP as large amounts of data can be transferred.

To see the stages in a name lookup use `set debug` in `nslookup`.

DNS does have problems: in particular there is no authentication. If I get a message telling me that `www.bath.ac.uk` has IP address 138.38.32.14 can I be certain this is not the IP address of somebody else?

The Web server of the security company RSA was subverted by spoofing DNS. An authority for `.com` was convinced somehow that that the authority for `rsa.com` was a machine other than the real one (`dnsauth1.sys.gtei.net` is a real one). Let's say `dns.spoofed.com` was made authority. This was a cracked machine somewhere. Now `dns.spoofed.com` said that `www.rsa.com` resolved to some IP address, let's say for `www.spoofed.com`. A replacement web page for RSA was set up on `www.spoofed.com` When users tried to get the web page for `www.rsa.com` they actually got the web page from `www.spoofed.com`. Which said nasty things about RSA the company. None of RSA's machines were ever touched.

RFC2065 There is a solution to this in *Secure DNS* (DNSSec), which uses public key authentication, involving cryptographically secure authentication certificates. It hasn't really taken off, possibly due to unfamiliarity with the concepts. And it doesn't really make sense until many people use it, and nobody is going first.

The Unix `whois` can be used to find the name of the current owner of a domain name. You must first determine which *whois server* to query, e.g., `whois.networksolutions.com` for `.com` names. Then

```
whois -h whois.networksolutions.com yahoo.com
```

to find who has registered the name `yahoo.com`. The whois server is often `whois.`*`company`* where *company* is the SOA for that domain. For example, `whois.nic.uk` for `co.uk`, and `whois.ja.net` for `ac.uk`.

---

11 The Transport Layer

The IP has two protocols defined at the transport layer. They are complementary, one (UDP) being fast, unreliable, connectionless, and the other (TCP) being more sophisticated, reliable and connection-oriented.

Both use *ports*. At any time there may be many services available on the server machine that a client may want to use, e.g., web pages serving, email delivery, telnet login, FTP, and so on. How does a client indicate which service it requires? In IP it is done by the use of *ports*.

A port is simply a 16 bit integer (1–65535). Every transport layer connection (using UDP and TCP) has a source port and destination port. When a service starts (i.e., some program or other that will deal with some service) it *listens* on a port. Connections made to that port are passed to the service. Certain *well-known ports* are reserved for certain

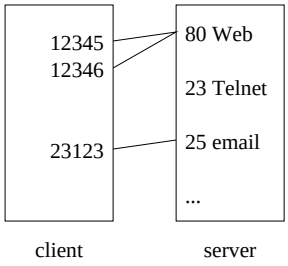


Figure 47: Ports

services, e.g., a web server on port 80; other ports are only available to privileged programs; most ports are available to any program that wants to use them.

One analogy is to think of a host as a block of flats. Inside the block of flats there are many occupants, doing many different things. The block has a single address. A letter to be delivered needs a flat number as well as the main address.

Also the use of ports solve another problem: that of several connections between two machines. There can be several users on the client machine all connecting to the same port on the server, e.g., several people retrieving web pages. The source port allows us to distinguish connections and hand the replies back to the correct user. Source port numbers are generally chosen afresh for each connection from the pool of currently unused numbers: these are called *ephemeral* ports, as they only live for the lifetime of the connection.

The quad

(source address, source port, destination address, destination port)

identifies a connection uniquely. The pair (address, port) is sometimes called a *socket*, while the full quad is sometimes called a *socket pair*.

RFC1700 et seq

See RFC1700 et seq (now kept on a Web site) for allocations of port numbers, or look at the file `/etc/services`.

Use the Unix command
`netstat -f inet`
to see the current Internet connections on a machine. The reply uses the format `machine.port` for source, then destination. `Port` is either a port number, or a name like `www` or `domain` (DNS) from `/etc/services`.

Both UDP and TCP have ports fields in their headers. In fact both have these fields at the very start of the headers, as this allows the port numbers to be included in the “IP header and 8 bytes of data” that an ICMP error message contains. This is so that the operating system can identify which program sent the original packet (from the source port) and can direct an error message back to it.

12 The Transport Layer: UDP

RFC768

The *User Datagram Protocol* is the transport layer for an unreliable, connectionless protocol.

12.1 Ports

Port numbers for the source and destination.

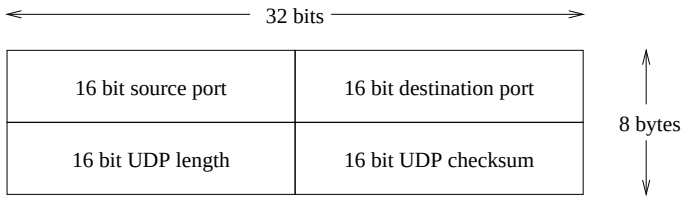


Figure 48: UDP header

12.2 Length

This 16 bit field contains the length of the UDP packet, including the header, which is always 8 bytes long. Not strictly necessary as this can be computed as the length of the IP packet (as given in the IP header) minus the length of the IP header.

12.3 Checksum

Checksum of the UDP header plus some of the fields from the IP header. This field is optional, but recommended. If you don't want to compute the checksum (presumably for extra speed), put 0 here.

12.4 General

UDP is a thin layer on top of IP, we only add the minimum needed for a transport layer. It is as reliable or unreliable as the IP implementation it is based upon, and just about as fast and efficient as IP, with only a small overhead.

UDP is widely used as is is good in two areas

1. One shot applications: where we have a single request and reply. Examples include the Domain Name Service (DNS) which looks up the IP address associated with a name.
2. When speed is more important than accuracy: such as Real Audio, where the occasional lost packet is not a problem, but a late packet is.

No provision is made for lost or duplicated packets in this protocol. If an application uses UDP then it must deal with these issues itself. For example, it can set a timer when it sends a request datagram. If the reply takes too long in coming, assume the datagram was lost, and resend. DNS over UDP is like this. Duplicated datagrams are not a problem to DNS, but they might be to other applications.

13 The Transport Layer: TCP

RFC793

The *Transmission Control Protocol* is the transport layer for a reliable, connection-oriented protocol. Often called *TCP/IP* to emphasise its layering on top of IP, it is hugely more complicated than UDP as it must create a reliable layer from an unreliable IP. Most of this complication is in the handling of error cases, though some is in the details to improve performance and flow control.

The basis of the reliability is by use of acknowledgements for every packet sent. So if A sends a packet to B, B must send a packet back to A purely to acknowledge the arrival of the packet. If A doesn't get an acknowledgement, then it resends the packet.

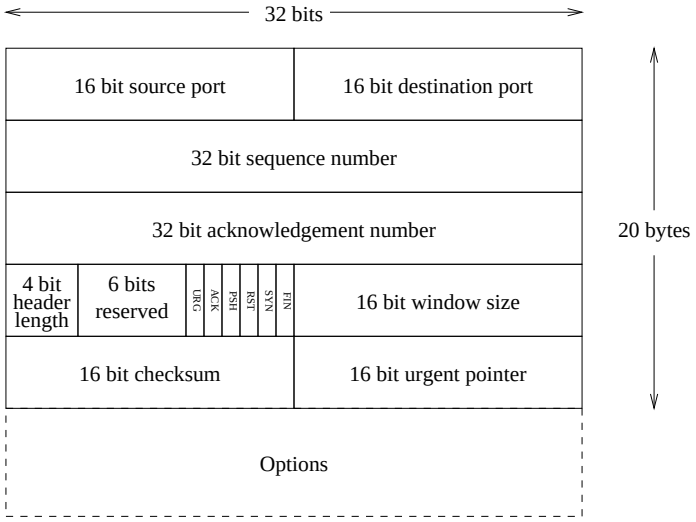


Figure 49: TCP header

The use of acknowledgements raises the question of the *Two Army Problem*. Two armies A and B wish to coordinate an attack on a third army C. So A sends a message to B, saying "Attack at dawn". How does A know that B got the message? A cannot safely attack until it is sure B got the message. So B sends an acknowledgement back to A. This may seem enough, but the acknowledgement may be intercepted, so A may not discover that its message got through, and A cannot yet attack. B realises this, and cannot attack. To fix this, A must send a second acknowledgement to B, to say it received B's acknowledgement. But this might not get through, so B must send a third acknowledgement back to A. And so on to infinity. We can never be sure that both parties have confirmed the attack if we don't have a reliable channel.

A starts a *retransmission timer* when TCP sends the packet: if the timer runs out before the ACK is received, A resends. We shall go into this in detail later, but consider the problems to be solved:

- how long to wait before a resend: this may be a slow but otherwise reliable link, and resending will just clog the system with extra packets
- how many times to resend before giving up: it may be that the destination has gone away completely
- how long to wait before sending an ACK: you can piggyback an ACK on a normal data packet, so it may be better to wait until some data is ready to return rather than sending an empty ACK as this will reduce the total number of packets sent
- how to maintain order: IP packets can arrive out of order, so we need some way to restore order when reassembling the data stream
- how to manage duplicates: IP packets can be duplicated, so we need some way to recognise and discard extra copies (a packet can be duplicated if someone thought it was lost when it wasn't and resends)
- flow control

Packets in a TCP connection are often called *segments*. The TCP header contains many fields.

13.1 Ports

Port numbers for the source and destination.

13.2 Sequence and Acknowledgement

These are at the heart of the reliability of TCP. Each byte in the data stream from source to destination is numbered. This 32 bit unsigned *sequence number* starts off at some random(ish) value at connection initialisation, and increases by 1 for each byte sent. The sequence number is the the number of the first byte in the segment's data.

The destination acknowledges those bytes it receives. Note that it may not get a whole segment due to fragmentation. The acknowledgement field is only active if the ACK flag (see below) is set. The reverse connection from destination to source has its own sequence number, as TCP is full duplex.

TCP is full duplex at the IP layer: it may or may not be full duplex at the link layer.

If the sequence number is 10000, and 10 bytes are received, the ACK is the sequence number of the next byte the destination expects to receive, i.e., 10011. Notice that you can use ACK in a normal data segment: this is called *piggybacking* the ACK, as the ACK gets a "free" ride on top of the returning data segment.

The sequence number wraps round at $2^{32} - 1$: this can cause problems with very high bandwidth connections. For example, this would wrap after about 8 seconds for a Gigabit Ethernet (actually longer, as some bandwidth is used by headers). It is easy to imagine a segment being delayed for this order of time. Thus, when the straggler does turn up, it might be confused with other segments with similar sequence numbers. For such cases, it might be wise to
RFC1323 use Protection Against Wrapped Sequence numbers, or PAWS. This co-opts the timestamp TCP option, and uses it to distinguish segments with the same sequence number that were sent at different times.

13.3 Header Length

The header has a variable length as we can have options. This 4 bit field gives the header length. The header is always at least 20 bytes, and can be up to 60 bytes.

13.4 Flags

- URG. Urgent data.
- ACK. The acknowledgement field is active.
- PSH. Push this data to the application as quickly as possible.
- RST. Reset (break) the connection.
- SYN. Synchronise a new connection.
- FIN. Finish a connection.

13.5 Window Size

This is used for flow control. The destination has only a limited amount of buffer memory it can store segments in. If the application is not reading the segments as fast as they are arriving, eventually the buffer will be filled up. The *window size* is the number of bytes that the destination is willing to accept, i.e., the amount of buffer it has left. If the window size is very small, the sender can slow down until the window increases again. The 16 bit field gives us a window of 65535 bytes, but there is an option to scale this to something larger.

Notice that the *advertised window size* is the amount of space free when the segment is *sent*. There may be more space freed up a little later when a reply is returned. Thus the window size is not a wholly accurate measurement, and it

gets more inaccurate as time passes. Nevertheless, it is safest to assume that the advertised window size is the largest amount of data that can be sent until we get a segment with a different window size. See section 14.1.

13.6 Checksum

A checksum of the TCP header plus some of the fields from the IP header.

13.7 Urgent Pointer

This is active if the URG flag is set. It is an offset into the TCP data stream indicating the end of the current urgent data block. Urgent data includes things like interrupts that need to be processed before any other data that is buffered (hitting control-C during a big FTP transfer).

13.8 Options

There are several of these, including the window scale option, and *maximum segment size* (MSS). More later.

13.9 Data

Finally, the data. This can be empty, and is often so while setting up or tearing down a connection, or for an ACK when there is no data to be returned.

13.10 TCP Acknowledgements

Reliable delivery is achieved through acknowledgements. These are sent back to the sender of a segment, and the acknowledgement field of the TCP header indicates which byte we are expecting next.

If, after a suitable period of time has elapsed (see later), no ACK is received by the sender, it realises that the original segment was lost, and resends it.

A is sending 10 bytes segments to B at regular intervals, and B ACKs them. But segment containing bytes 21–30 gets lost. When B gets a segment with bytes 31–40 it ACKs with value 31: it expected byte 31 next. While the ACK is travelling back to A, A is still sending. Each time, B ACKs with 31.

So eventually A receives *duplicate ACKs* for 31. If this happens, A can tell something is wrong. Eventually, A times out on the ACK for bytes 21–30, and resends the segment with bytes 21–30. B gets this, and is able to ACK all the way up to byte 60.

13.11 Connection Setup and Tear Down

Setting up a connection is a complicated business. There is connection state to be initialised (e.g., sequence numbers) that will be used throughout the connection. TCP is a *connection-oriented* protocol. UDP, on the other hand *connectionless*, and is *stateless*, as each segment is independent of all others.

At the other end, tearing down a connection is not trivial either, as we need to make sure that all segments in flight have been received safely: you can't just drop segments that arrive since the other end is awaiting ACKs.

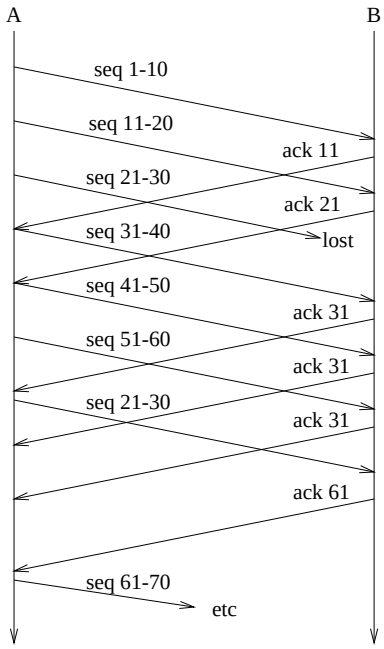


Figure 50: TCP Acknowledgements

13.11.1 Connection Establishment Protocol

Three segments are used to make a connection.

1. The initiator (normally called the *client*) sends a SYN segment, i.e., a segment with the SYN flag set, containing the *initial sequence number* (ISN), n, say.
2. The server replies with another SYN segment containing its own ISN, m say. It also ACKs the client's segment, i.e., sends a segment with the SYN and ACK flags set, and the ACK field set to $n + 1$. The SYN flag consumes one byte. (Consider why this is: so we can ACK the SYN independently of the first data byte. These initial segments can be lost just as much as any other!)
3. The client ACKs the server's SYN with $m + 1$.

In all three segments the data field is empty: these segment are overhead in setting up the connection.

This is called the *three way handshake*. The initiator is said to do an *active open*, while the server does a *passive*
RFC793 *open*. The ISN is a number that should change over time. RFC793 suggests that it should be incremented ever 4 microseconds. The reason to change the ISN is so that slow segments from an earlier connection to the same machine and port cannot be confused with the current connection.

RFC1948 These days, it is better to choose random ISNs to avoid IP attacks which start with a malicious person guessing the ISN for someone else's connection and inserting their own segments into the connection.

It is possible, but hard, to do a *simultaneous open*. This is when both ends send a SYN, and the segments cross in flight. This is defined to result in one connection, not two.

The TCP protocol handles this eventuality, too.

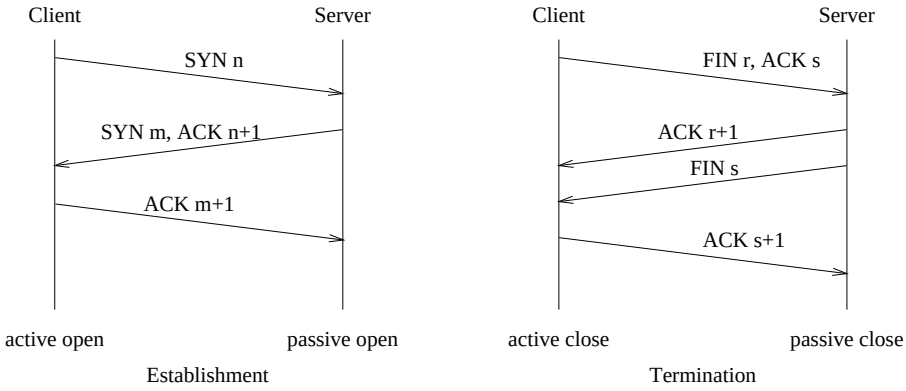


Figure 51: TCP setup and tear down

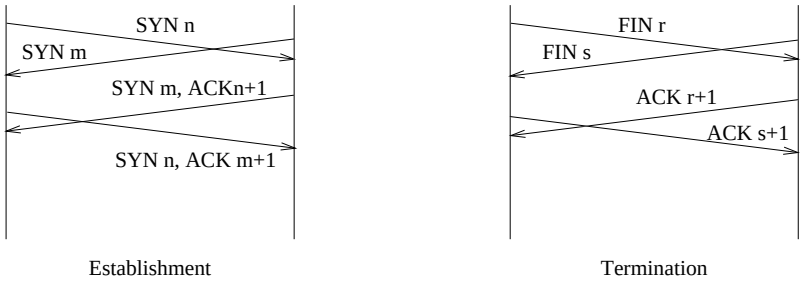


Figure 52: TCP Simultaneous Open and Close

13.11.2 Connection Termination Protocol

Four segments are used to take down a connection. This is because a connection can be *half closed*. Since TCP is full duplex, we can close one direction of traffic independently of the other. The classic example is sending a sequence of integers to a server to be sorted: closing the client send end of the conection is used to indicate the end of the sequence, while the reverse connection is kept open to receive the reply.

The FIN flag is used to indicate close. This will be ACKed by the other end. A FIN consumes one sequence number. Later, when the other end is done, it will send a FIN, and get an ACK back.

There are many variations on this. Either end can send the first FIN and do the *active close*, and then the other end will do a *passive close*. The FIN of the passive end can be piggybacked on the ACK of the active FIN, so that only three segments are used. The ends can do a *simultaneous close*, i.e., both send a FIN before receiving one: the protocol still works.

13.12 Resets

There is another way for connections to be broken: use a *reset* (RST) segment. This is normally for cases in error, e.g., a segment arrives that doesn't appear to be for the current connection. This can happen when the server crashes and reboots. Recall that a connection is characterised by the quad (source IP address, source port, destination IP address, destination port). So if a segment arrives from a client to a server that is not expecting it, a RST happens. This is why connections often stay up until the remote machine has rebooted, and then we get "connection reset by peer".

Another example is if a TCP connection is attempted to a port that has no process listening on it, a RST returned to the client. UDP, on the other hand, sends an ICMP error packet.

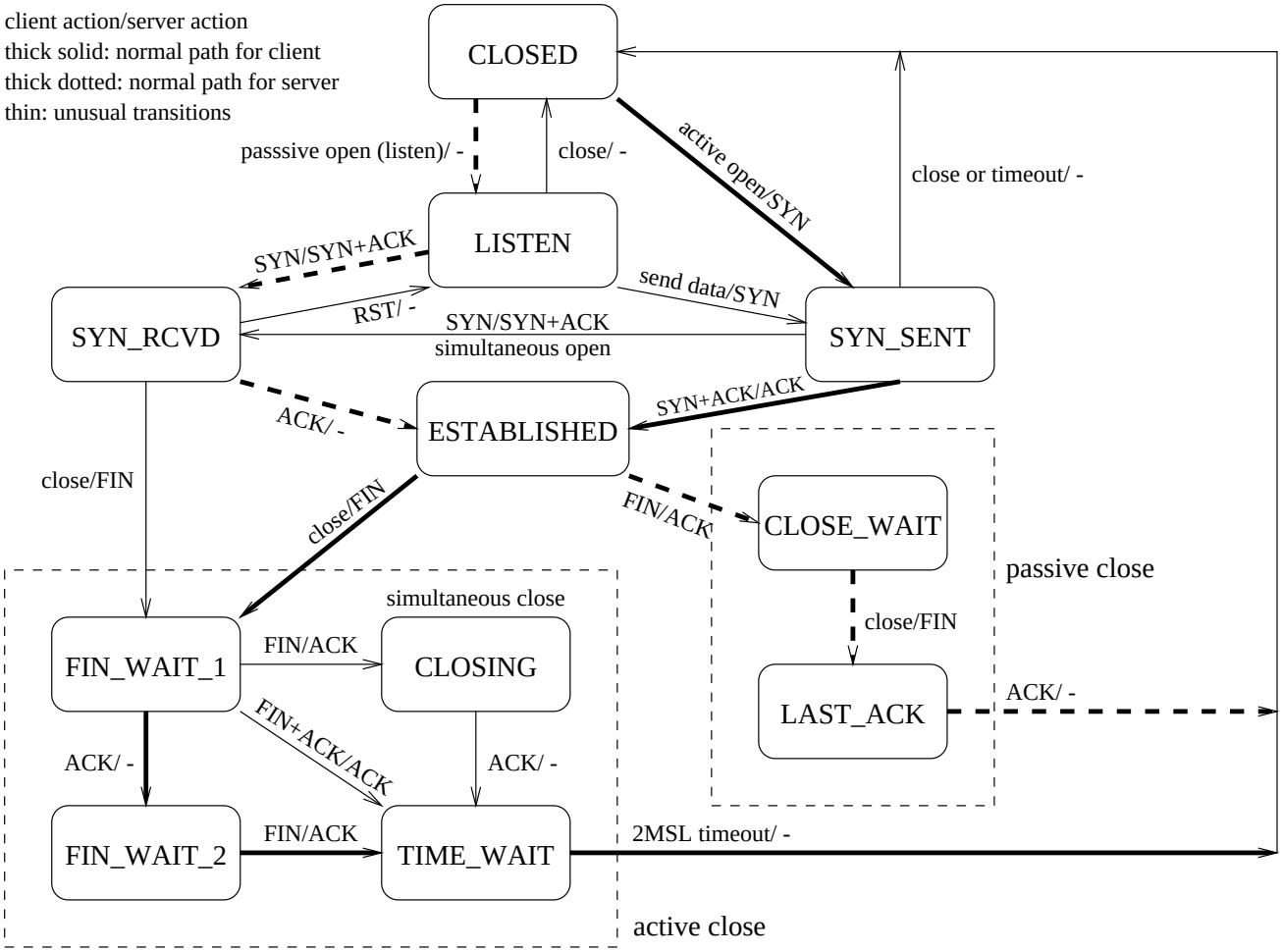


Figure 53: TCP State Machine

A connection that terminates using FIN is said to have an *orderly release*, while one terminated by a RST is called an *abortive release*. A RST will flush all buffered segments for that connection and pass an error message up to the application layer.

RST segments are not ACKed: the connection stops right here.

13.13 TCP State Machine

The transitions between states in TCP are complicated. There is a standard TCP State Transition Diagram that indicates how a TCP connection must act in all cases.

The states you see in this diagram are what you get from `netstat -f inet`.

You should spend some time working through examples on this diagram. You may wish to try: open, orderly close, simultaneous open, simultaneous close, abortive close. Note that this state diagram is applied for *each* TCP connection.

There is one state in the active close that is worth spending some time on. This is the TIME_WAIT state, also called the 2MSL state. Each implementation must choose a value for the *maximum segment lifetime* (MSL). This is the longest time a segment can live in the network before being discarded. This time is bounded, as the TTL field in the IP header ensures segments will die at some point.

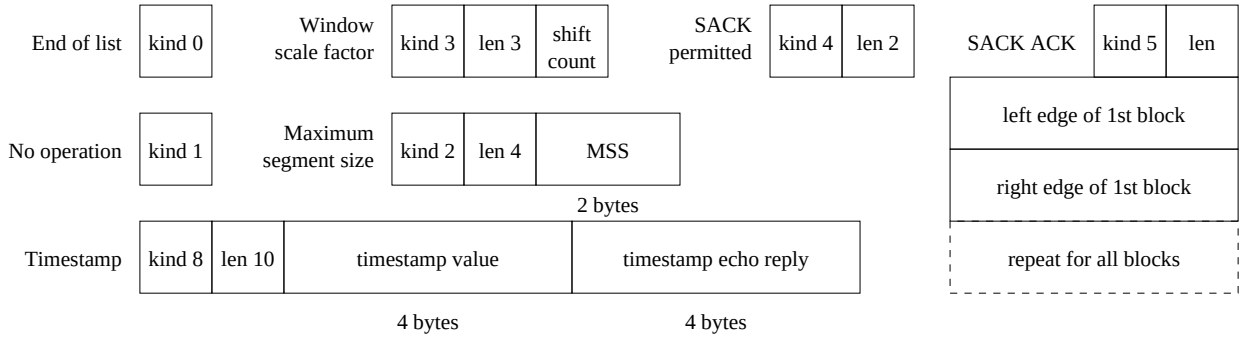


Figure 54: Some TCP Options

RFC793 RFC793 specifies a MSL of 2 minutes. You will often find values of 30 seconds, 1 minute or 2 minutes.

When TCP does an active close, it must stay in the TIME_WAIT for twice the MSL. This is in case the final ACK was lost, and gives the other end chance to time out and retransmit its final FIN. Any other segments that are received in this state are simply dropped. This means the connection (i.e., the quad above) cannot be reused until 2MSL has elapsed. This is not a problem if the client does an active close, as it likely using an ephemeral port, and any other port will do if we need to make a new connection within the 2MSL. On the other hand, if the server does the active close there is likely a problem. Since servers often listen on specific well known ports, a server cannot restart until the 2MSL has passed.

RFC793 There is a slight problem if machine in the 2MSL state crashes, reboots and starts a new connection with the same quad all within 2MSL. There is a chance that delayed segments from the old connection will be interpreted as part of the new connection. To remedy this, RFC793 states that TCP should not create any new connections until MSL after rebooting: this is called the *quiet time*. Not many people implement this, as it usually takes more time than this time to reboot: though many get close, and high availability machines can be back very quickly.

Another state of interest is FIN_WAIT_2, where we have sent an FIN and the other end has ACKed it, but not yet sent their FIN. If the remote crashes we can be stuck here forever waiting. Many implementations set a timer on this state, and if nothing is forthcoming for 10 minutes 75 seconds, they violate the protocol and move to TIME_WAIT.

13.14 TCP Options

RFC793
RFC1323

An option starts with a 1 byte *kind* that specifies what this option is to do. Options of kind 0 and 1 take 1 byte. All other options next have a length field that gives us the total length of this option. This is so that an implementation can skip an option if it does not know the kind.

The NOP is to pad fields out to a multiple of 4 bytes.

Stevens p. 253 for an example.

Maximum Segment Size (MSS) specifies how large a segment we can cope with without fragmentation. The bigger the better, of course, as this reduces the overheads of headers. MSS has maximum value 65535. The Window!Scale option gives the number of bits to scale the TCP window size, from 0 to 14. A value of n multiplies the size by 2^n. This gives us up to $65535 \times 2^{14} = 1,073,725,440$ bytes in a window. A gigabyte is a big buffer! (Though for a 10Gb/sec connection, that's about a second's worth of data!) For an Ethernet, whose physical layer allows 1500 bytes, the MSS can be as large as 1460 bytes when we take the 20 byte IP header and 20 byte TCP header into account. For non-local networks, the MSS often defaults to the minimum of $576 - 40 = 536$ bytes.

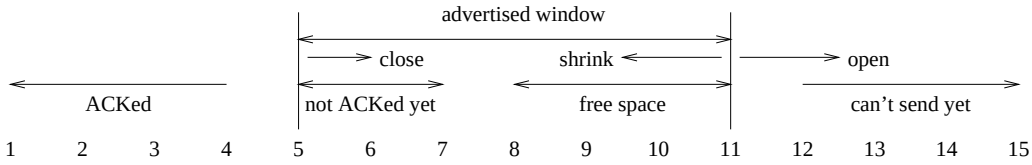


Figure 55: TCP Sliding Window

Timestamp puts a time-of-day value in a segment, giving us some idea of how long segments are taking to travel through the network. *Selective Acknowledgement* (SACK) Permitted is a modern option that allows us to be more specific about which bytes we are ACKing in a segment.

Many options are only available in SYN segments, e.g., Window Scale, MSS and SACK Permitted.

14 TCP Strategies

TCP gets its reliability by acknowledging every segment sent. Does this mean that two segments for every packet? It is possible to implement TCP like this, but you would get poor performance. Instead, TCP implementations use a variety of strategies to improve performance, but still sticking to the letter of the law of the TCP protocol.

14.1 Sliding Window

The TCP header in a returning segment contains a field that informs the sender of how much buffer space is free at the destination. The amount of free space depends on two things:

1. how fast the sender is sending data, and
2. how fast the destination application is consuming the data.

If the sender is producing data faster than the application reads it, the buffer space will soon be used up. The *advertised window* is a mechanism to tell the producer to slow down when necessary. This is a form of *sliding window* protocol that operates as a *flow control*.

The idea of the sliding window is that it describes the range of bytes that the sender can transmit: the sender should never send more bytes than the window size. If the receiver is having problems keeping up, the window gets smaller, and the sender will send fewer bytes. When space frees up in the receiver, the window gets bigger, and the sender can transmit more. The advertised window is a dynamic value that the sender recomputes every time it receives an ACK.

The sliding window has its left hand edge defined by the ACK value, and the right hand edge by the TCP window size field. The window size is filled in by the receiver on every ACK returned and it represents the range of bytes the destination is willing to receive at this moment in time. As more ACKs are returned, the window *closes* by the left edge advancing. As data is removed from the buffer by the application, the window *opens* by the right edge advancing. There is the (fairly unusual) possibility of the window *shrinking*, perhaps when the amount of buffer space available to a TCP connection is reduced due to it being needed elsewhere.

Bytes to the left of the window (bytes 1–4) are ACKed and safe. Bytes to the right (12 and onwards) cannot yet be sent. Bytes within the window range fall into two classes: not ACKed yet, and free space. The unACKed bytes (5–7), usually a fairly small range, are those that have been read, but the ACK has not yet been sent. The free space (8–11) is the actual range of bytes that the receiver can buffer. The sender can compute this as the advertised window minus the number of unACKed bytes.

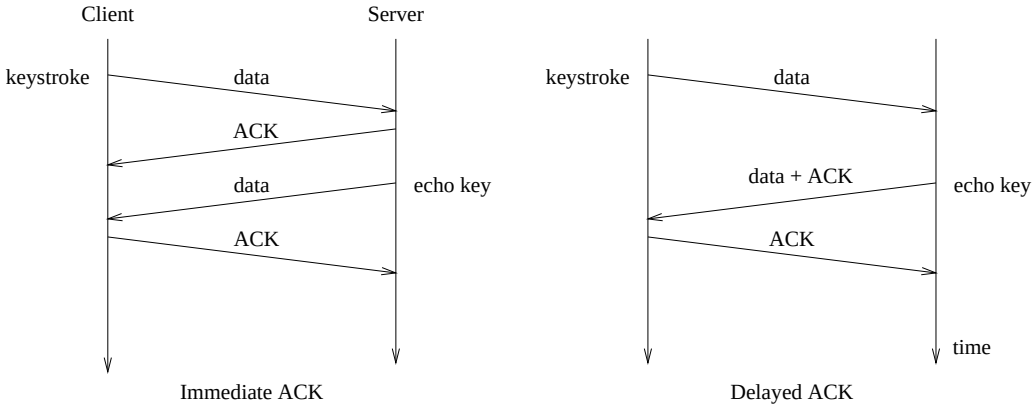


Figure 56: Delayed ACK

It is perfectly possible for the window size to reduce to 0. If this happens the sender will have to wait before sending any more data. When the receiver is ready for more data it will send a duplicate ACK with the new different window size: this is a *window update* segment.

Complications arise if window updates get lost: see the *Persist Timer* (section 14.7).

14.2 Delayed acknowledgements

Instead of immediately ACKing every segment, maybe we should wait a little until we return a data segment, and piggyback the ACK on it. If no return data is forthcoming, we send a normal ACK.

For example, when using `telnet` to connect to a remote machine, each keystroke is echoed to your screen by the remote machine. An immediate ACK would use four segments. Delaying the ACK just a little allows it to piggyback on the echo segment.

The total time taken for the exchange is the same, but less segments are used. However, fewer segments means fewer possible collisions, and this is good, particularly in the case of a heavily loaded network.

The big question is: how long do we delay an ACK? If we delay too long, the sender might think the segment was lost, and resend; if we delay too short, we don't get as many free piggybacks. A typical implementation will delay an ACK for up to 200ms.

RFC2581 You must not delay an ACK for more than 500ms.

This another of many *timers* associated with TCP. Each time you receive a segment you (i.e., the TCP software) must set timer for that segment that runs out after 200ms. If the segment has not yet been ACKed, do so then. If fact, many implementations cheat and have just one global timer (rather than a timer for each segment received) that goes off every 200ms, and any outstanding segments are ACKed then. This means there is a maximum of 200ms delay, but it could be much shorter. A single timer is easier to implement, of course.

If you receive an out of sequence segment (the sequence number is not the one you are expecting next), for example when a segment gets lost, then you must not delay, but send an ACK immediately. This may mean resending an ACK you have sent already: a *duplicate ACK*. This is to inform the sender as quickly as possible that something has gone wrong.

14.3 Nagle's algorithm

RFC896

When sending individual keystrokes using TCP over a network there is a big waste of bandwidth going on. One keystroke is (typically) one byte. This is sent in a TCP segment with 20 bytes of header, which is sent in a IP segment with 20 more bytes of header. Thus we are sending a 41 byte segment for each byte of data. Such a small segment is sometimes called a *tinygram*. On a LAN this is not too bad as they generally have bandwidth to spare, but on a WAN this proliferation of tinygrams causes additional congestion.

Nagle invented an acknowledgement strategy that reduces this effect. This applies to the sending side (the client) rather than the receiving side (the server). This is *Nagle's algorithm*:

> a TCP connection can have only *one* unacknowledged small segment outstanding. No additional small segments should be sent until that acknowledgement has been received.

Any small waiting segments are collected together into a single larger segment that is sent when the ACK is received. This segment can also be sent if you buffer enough tinygrams to fill a segment, or have exceeded half the destination's window size.

This leaves open the definition of "small", and there are variants that choose anything from "1 byte" to "any segment shorter than the maximum segment size". The latter is more appropriate when combined with other strategies (see "silly window syndrome," section14.4).

This is a very simple strategy to apply, and reduces the number of tinygrams without introducing extra perceived delay (over that delay that results from a slow WAN). The faster that ACKs come back, the more tinygrams can be sent. When there is congestion, so ACKs come back more slowly, fewer tinygrams are sent. When a network is heavily loaded, Nagle's algorithm can reduce the number of segments considerably.

> Notice that it is to the client's advantage to hold back and not flood the system with tinygrams, as it reduces overall congestion and improves its own throughput.

Sometimes it is better to turn off Nagle's algorithm. The usual example of this is using X Windows over a network. Each mouse movement translates to a tinygram on the network. It would not be good to buffer these up, as that would cause the cursor to jump erratically about the screen. Such applications can call a function to disable Nagle for this connection.

14.4 Silly Window Syndrome

Another problem that can cause bad TCP performance is *silly window syndrome*. Recall that segments advertise a window, that is they say how much space there is available to buffer incoming segments. This is intended to slow down a sender if the recipient can't keep up with the data flow.

Consider what can happen if the sender (A) is sending large segments, but the receiver (B) is reading the data slowly at one byte at a time.

- B's buffer fills up, and B sends a segment (probably the ACK of A's last segment) saying "window size 0".
- B reads a byte.
- B send a window update segment: "window size 1".
- A gets this, and sends a segment containing one byte.
- B ACKs, with "window size 0"

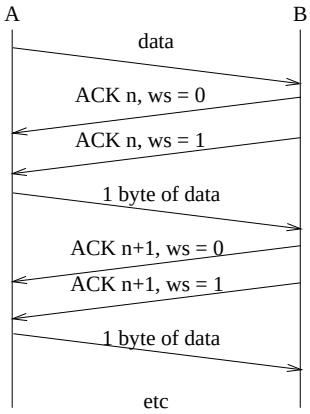


Figure 57: Silly Window Syndrome

- B reads a byte.
- Repeat.

We are back to the two segment, high overhead per byte scenario. This is the *silly window syndrome*, where the advertised window is not helping throughput. Better would be for B to wait until a decent sized window is available
RFC813 before it sends a window update. Clark's algorithm to avoid SWS is never to send window updates for 1 byte. Instead, only advertise a new window when either enough buffer space is available to accommodate a maximum segment size, or the buffer is half empty, whichever is smaller.

This also answers the question of "small" in Nagle's algorithm. If "small" means "less than the maximum segment size", Nagle and Clark fit together naturally.

14.5 Congestion Control

RFC2001

Congestion happens when more traffic is sent to the network than it can handle. There are several strategies that deal with congestion in TCP connections.

One point arises: how can we spot that congestion is happening, given that it may be many hops away from us? The trick is to watch for segment loss. Segments can be lost through poor transmission, or through being dropped at a congested router or destination. In these days of fibre optic transmission the former does not happen, so it is safe to assume that all segment loss is due to congestion. So TCP implementations watch for segment loss (missing ACKs), and treat this as an indication of congestion.

Congestion can happen in two places. Firstly in a router somewhere on the path to the destination due to lack of capacity in the next link. Secondly in the destination itself if the destination is just too slow to receive data at the full rate the network can deliver.

The TCP advertised window size is there to deal with congestion at the destination. The path congestion needs to be addressed in a different way.

14.5.1 Slow Start and Congestion Avoidance

If we have lot of data to send we do not want to wait for an ACK before we send each segment. Instead, we use the network's bandwidth much better if we send off several segments before stopping to wait for ACKs to see if the segments were successfully received. We want to avoid too much waiting as this is wasted time when we could potentially be sending more segments. However, sending *too* many segments at a time is just as bad if the network is

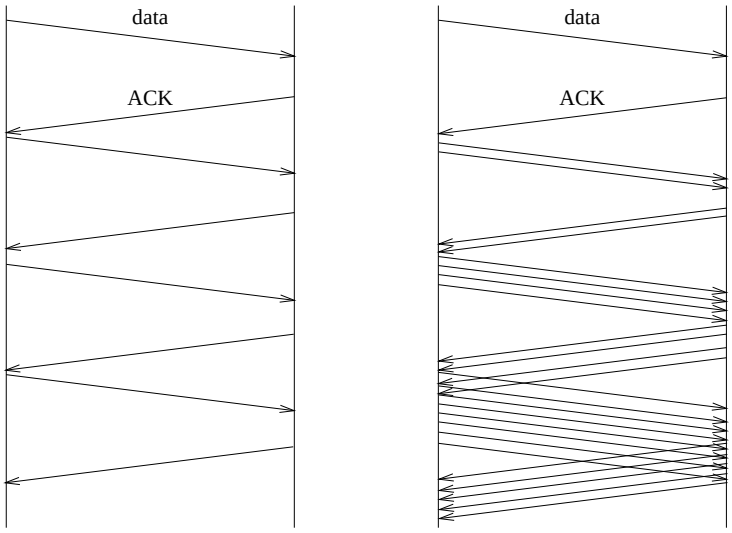


Figure 58: Slow Start

congested and cannot cope with this many segments. We need a way to estimate the capacity of the network in order to send as many segments as we can, but not too many. If we get it right, we can have a continual stream of data segments being sent out while a continual stream of ACK are coming back.

The *slow start* algorithm tries to estimate the path congestion by measuring the number of ACKs that come back.

Slow start adds a new window, the *congestion window*, which is an estimate of the capacity of the path. This will be an additional constraint on the amount of data to transmit: the sender can send data up to the minimum of the advertised window size and the congestion window size.

The congestion window size is initialised to the maximum segment size of the destination. Also, a variable, the *threshold* is initialised to 64KB. Every time a timely ACK is received, the congestion window is increased by one segment. So at first the sender can send only one segment; then two at a time; then four at a time, and so on.

The technique is called “slow start”, but actually this is an exponential increase in the congestion window over time.

This doubling only repeats until we reach the current threshold. When the congestion window reaches the threshold, the slow start algorithm stops, and the *congestion avoidance* algorithm takes over. Rather than an exponential increase, congestion avoidance uses a linear increase of $1/\text{congestion window size}$ each time.

Eventually the network’s limit will be reached, and a router somewhere will start discarding segments. The sender will realise this when ACKs stop coming. At this point, the threshold is halved, and the current window is dropped back to one segment size again.

Slow start takes over until the new threshold is reached, then congestion avoidance starts. And so on.

If no congestion occurs, the congestion window grows until it reaches the advertised window size: in this case the receiver is the limiting factor, not the network.

Notice that the advertised window is a constraint imposed by the receiver to deal with congestion at the destination, while the congestion window is a constraint imposed by the sender to deal with congestion in the network.

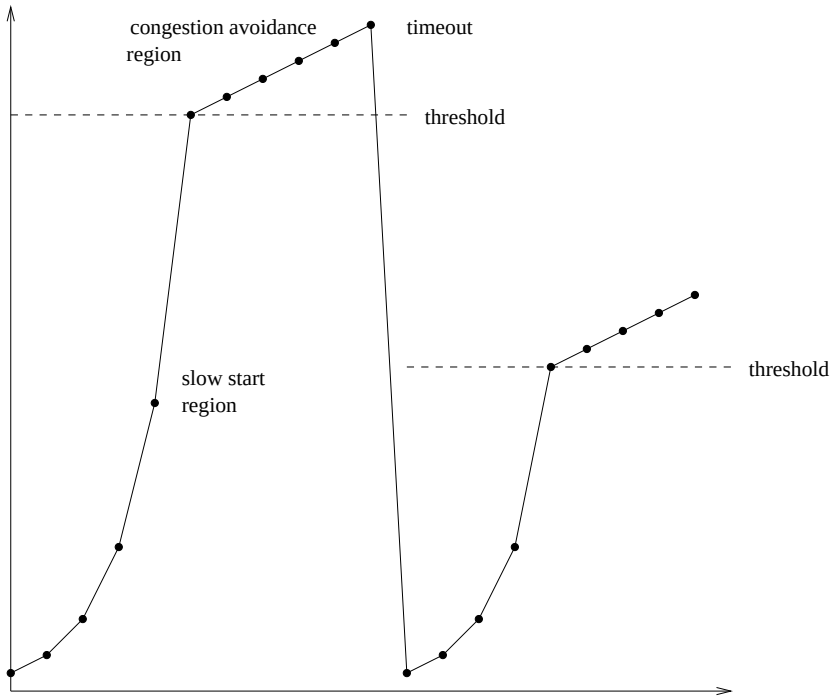


Figure 59: Slow Start/Congestion Avoidance

14.5.2 Fast Retransmit and Fast Recovery

When an out-of-order TCP segment is received, the TCP protocol calls for an immediate ACK: it must not be delayed. This is to inform the sender as soon as possible which segment sequence number was expected. We saw above how duplicate ACKs arise from a lost segment.

Jacobson's *fast retransmit* takes the idea that several duplicate ACKs is indeed symptomatic of a lost segment. The argument is that one or two duplicated ACKs might just be due to out-of-order delivery of segments. But three or more, and something is badly wrong. If this happens, retransmit the indicated segment immediately, without waiting for the usual timeout.

Next, perform congestion avoidance, and not do not go into slow start. This is the *fast recovery* algorithm. We do not want slow start, as a duplicate ACK indicates that later data has actually reached the destination and is buffered by the destination. So data is still flowing, and we don't want to abruptly reduce the flow by going into slow start.

14.5.3 Explicit Congestion Notification

RFC2481

A new mechanism for congestion avoidance is called *explicit congestion notification* (ECN). This gives routers the ability to put a mark on packets as they go past that indicates that that route is becoming congested. This can be used to give prior notification to the hosts to slow down before packets start getting dropped.

ECN uses bits 6 and 7 in the type of service (TOS) field in the IP header. These are bits that previously were unused and set to zero. Bit 6, the *ECN-capable transport* (ECT) bit indicates that the hosts are aware of the ECN mechanism. This is to provide backwards compatibility with hosts that do not understand ECN, as those hosts should already be setting bit 6 to zero. Bit 7, the *congestion experienced* (CE) bit is set when a router is congested and the hosts are ECN aware.

On receipt of a CE packet, a host is recommended to act as if a single packet had been dropped. For example, with TCP, this would trigger congestion avoidance and halve the congestion window.

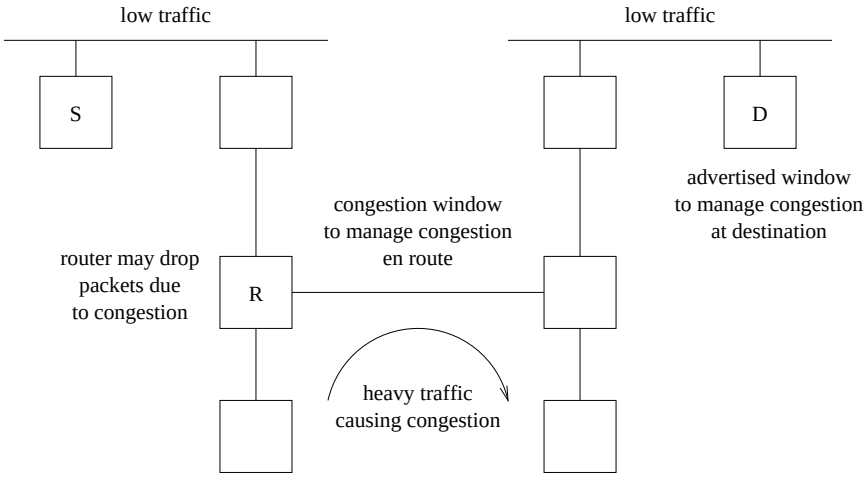


Figure 60: Congestion Windows

Unfortunately, in early deployment of ECN some ECN unaware routers and hosts treated packets with TOS bit 6 set as corrupted and dropped them. This meant that ECN capable hosts sometimes could not reach hosts behind ECN unaware routers. Until all routers are at least *aware* of ECN (even if they do nothing about it) the only preactical recourse is to turn off ECN for that destination.

RFC2309 A technique called *Random Early Detection* (RED) can be used in routers to gauge when congestion is about to happen. This monitors the queues of packets that are waiting to be relayed onwards and notifies congestion when the average queue length exceeds a threshold. Previously, the only way to notify congestion was to drop the packet, but now ECN is available.

RFC2884 Performance measurements show ECN to be good, as early notification of congestion allows IP to slow down before segments are dropped, and has the added benefit of fewer retransmits consuming network bandwidth. Theoretically, a fully ECN-enabled Internet would have no packet loss.

14.6 Retransmission Timer

We now look at the timer that determines when to resend a segment. Too short a time is bad since we might just be on a slow connection. But we do want as short a time as the network allows.

What would be even better would be to have a dynamic timeout interval that adjusts itself to the current network conditions: if the network slows down (e.g., extra traffic somewhere en route) the timeout should increase. If the network speeds up (e.g., a quiet period), the timeout should decrease.

Jacobson gave an easy to implement algorithm that does this for us. For each connection TCP keeps a variable, RTT, that is the best current estimate for the *round trip time* of a segment going out and its ACKs getting back.

When a segment is sent, the timer is started. If the ACK returns before the timeout, TCP looks at the actual round trip time M and updates RTT

$$RTT = \alpha RTT + (1 - \alpha)M,$$

where α is a smoothing factor, typically set to $7/8$.

Next, we need to determine a timeout interval given RTT. Jacobson suggested taking the standard deviation into account: if the measured RTTs have a large deviation it makes sense to have a larger timeout. The standard deviation is hard to compute quickly (square roots), so instead Jacobson used the *mean deviation*

$$D = \beta D + (1 - \beta)|RTT - M|.$$

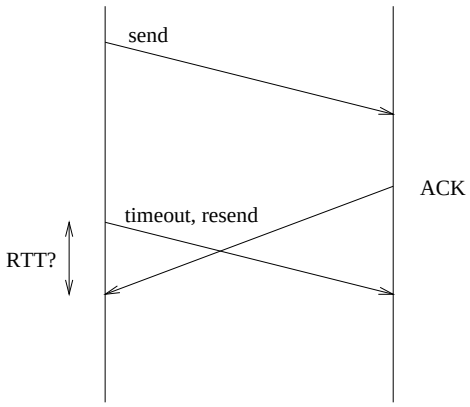


Figure 61: Retransmission Ambiguity

D is close to the standard deviation, and is much easier to calculate. A typical value for β is 3/4.

The timeout interval is set to

$$T = RTT + 4D.$$

The 4 was found to be good in practice.

What should we do if the segment is sent, and the timeout is triggered? We need to increase RTT somehow. Simply using the RTT of the resent segment is not a good idea, as we might might get the the ACK of the original segment (the *retransmission ambiguity problem*). In this case the value would be way undersized, and we wouldn't want to update the RTT using it.

Karn's algorithm is to double the timeout on each failure, but do not adjust RTT. When segments start getting though, the normal RTT updates resume, and the RTT value quickly reaches the appropriate value. The doubling is called *exponential backoff*.

14.7 Persist Timer

The TCP protocol needs several timers. The most obvious is the retransmit timer. Another is the 2MSL timer. There is also the *persist timer* (or *persistence timer*). Its role is to prevent a deadlock:

1. A sends to B
2. B replies with an ACK, and a window size of 0
3. A gets the ACK, and holds off sending to B
4. B frees up the buffer space, and sends a window update to A
5. This gets lost

At this point A is waiting for the window update from B, and B is waiting for more data from A. This is deadlock.

To fix this, A starts a persist timer when it gets a window size 0 message. If the timer goes off, A prods B by sending a 1 byte segment. The ACK of this *window!probe* will contain B's current window size. If this is still 0, the persist timer is reset, and the process repeats. If B's buffer was still full, the ACK will indicate that the data byte will have to be resent.

The persist timeout starts at something like 1.5 seconds, doubling with each probe, and is rounded up or down to lie within 5 to 60 seconds. So the actual timeouts are 5, 5, 6, 12, 24, 48, 60, 60, 60, ... The persist timer never gives up, sending window probes until either the window opens, or the connection is terminated.

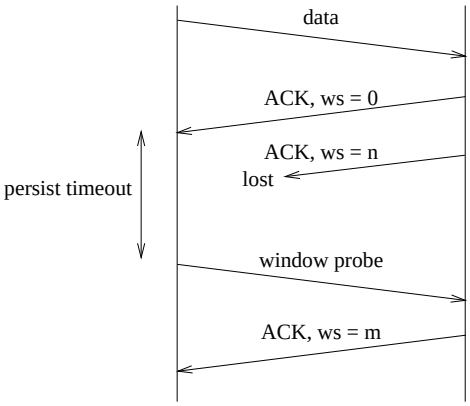


Figure 62: Persist Timer

The persist timer is unset when a non-zero window size is received.

14.8 Keepalive Timer

Yet another timer in TCP is the *keepalive timer*. This one is not required, and some implementations do not have it, and some people regard the facility as controversial.

When a TCP connection is idle no data flows between source and destination. This means that something along the path could break and be restored without the connection being any the wiser. This is probably a good thing. On the other hand, sometimes the server wants to know whether the client is still alive: maybe the connection uses some resources in the server that could be better used elsewhere. If the client has crashed, the server can reallocate the resources.

To do this the server sets the keepalive timer when the connection goes idle. This typically is set to time out after 2 hours. When the timer goes off, a *keepalive probe* is sent. This is simply an empty segment. If an ACK is received, all is well. If not, the the server may well assume that the client is no longer listening.

The client can be in one of four states:

1. the client is up and running: the keepalive probe is ACKed, and everybody is happy. The keepalive timer is reset to 2 hours
2. the client machine has crashed or is otherwise not responding to TCP: the server gets no ACK, and resends after 75 seconds. After 10 probes, 75 seconds apart, if there is no response the server terminates the connection “connection timed out”
3. the client machine has crashed and rebooted: the client will get the probe and respond with a RST. The server gets the RST and terminates the connection “connection reset by peer”
4. the client is up and running, but not reachable (e.g., a router is broken): this is indistinguishable from scenario 2 as far as the server is concerned, so the same sequence of events ensues. It may be that an ICMP “no route to host” is returned by an intermediate router in which case this information can be passed up to the application rather than “connection timed out”.

If a machine is shut down normally (rather than crashing), normal closedown of processes will cause TCP to send FINs for any open connections. These allow the server to close the connections.

There are several reasons not to use keepalive:

1. they can cause a good connection to be dropped because of an intermittent failure on a router, say
2. they use bandwidth
3. some network operators charge per packet

It is usually possible to disable the the keepalive feature in applications: indeed some people think keepalive should not be in the TCP layer, but should be handled by the application layer.

14.9 Path MTU Discovery

RFC1911

The *path MTU* is the largest segment that can be sent from source to destination without being fragmented somewhere along the way. Strictly defined, it is the smallest MTU of *any* path that connects the course to destination, as a segment could conceivably use any path.

Path MTU discovery tries to find the path MTU by adjusting the sizes of the packets it sends. When a connection starts, it uses a MSS which is $\min\{\text{MTU of interface}, \text{MSS announced by other end}\}$. If the other end does not specify an MSS, it uses the value 536. With this value IP can be sure that segments will not be fragmented because of the inability of the source or destination to cope.

All subsequent IP datagrams are sent with the DF (don't fragment) flag set. If fragmentation is needed en route, the segment is dropped, and an ICMP "fragmentation needed" is returned. The source can then reduce the MSS, and try again.

Modern versions of ICMP return the *next hop MTU* in the ICMP error, and the source can use this. Older implementations don't, and the source just picks a suitable smaller value from a table of MTUs:

65535	Hyperchannel, maximum MTU	RFC1044, RFC791
32000	just in case	
17914	16Mb/sec Token Ring	
8166	Token Passing Bus 802.4	RFC1042
4464	4Mb/sec Token Ring (maximum MTU)	
4352	FDDI	RFC1188
2048	Wideband Network	RFC907
2002	4Mb/sec Token Ring (recommended MTU)	
1500	Ethernet	RFC894
1492	IEEE 802.3	RFC1042
1006	SLIP	RFC1055
576	X.25	RFC877
512	NETBIOS	RFC1088
296	point-to-point (low delay)	RFC1144
68	minimum MTU	RFC791

If such an ICMP error occurs with a TCP connection, its congestion window should remain unchanged, but a slow start should begin.

RFC1191 You should try larger MTUs once in a while since routes can change dynamically. The recommended time is 10 minutes, but you will see intervals of 30 seconds being used.

Some poorly configured routers are set never to return ICMP errors, allegedly for "security". Such routers don't respond to pings. Also, these routers fail to operate with Path MTU Discovery if they have a small MTU: you never get the ICMP reply. Turning off the DF will allow the packet to be fragmented and get through the router. Thus, sometimes you can get a working TCP connection only if you turn off path MTU discovery! Clearly, this is a poor state of affairs, even worse than the ECN problem (section 14.5.3) since the systems administrators should know better.

14.10 Long Fat Pipes

Wide area networks have quite different problems to LANS. Instead of bandwidth being the major limiting factor, the difficulty lies with the speed of light: the actual time the bits take to get to the destination dominates.

The *bandwidth-delay product* is a measure of the capacity of a network.

$$\text{capacity in bits} = \text{bandwidth in bits/sec} \times \text{round trip time in seconds.}$$

The "Fat Pipe" which connects JANET to the USA contains an ATM link running at 155Mb/s. The RTT is approximately 70ms (as measured by `ping`). This gives a bandwidth-delay product of roughly 11Mb/s (1.4MB/s). This is the total number of bytes that can be in flight within the pipe. An Ethernet, running at a fairly good 5Mb/s and with a reasonable 3ms delay has a capacity of 1900 bytes. A gigabit network running over a satellite link (delay 0.5 sec) has a capacity of 62.5MB.

A network with a large bandwidth-delay product is called a *long fat network* (LFN, or "elephant"), and a TCP connection over an LFN is a *long fat pipe*.

Long fat pipes have many problems.

1. To be used efficiently, you must have a huge MTU, much bigger than the 65535 limit that standard TCP gives us. The *Window!Scale* option exists to fix this.
2. Packet loss in a LFN is disastrous. If a single packet is lost we have (a) lost a big chunk of data since the MTU is large, and (b) the subsequent reduction in segment size in the slow start/congestion avoidance algorithms hits us badly. The fast retransmit and recovery algorithms *must* be used here.
3. The TCP sequence number counts bytes using a 32 bit value. This wraps around after 4Gbytes. In a 10 gigabit network we can transmit this in about 3 to 4 seconds. It is possible that a segment could be lost for this amount of time, and reappear to clash with a later segment with the same sequence number. The *Protection Against Wrapped Sequence numbers*, or PAWS option co-opts the timestamp TCP option, and uses it to distinguish segments with the same sequence number that were sent at different times.

RFC1323

The biggest problem is *latency*, or the time a packet takes to get to the destination. If we want to send a megabyte across the Atlantic this will take about 76ms on a 155Mb/s link. The first bit arrives after 70ms, and the last 6ms later. If we could increase the bandwidth of the Fat Pipe ten-fold to 1550Mb/s, the first bit would *still* arrive after 70ms (it's the speed of light!), and the last just 0.6ms later, for a total of 70.6ms. The ten-fold increase has reduced the latency by just 7%.

Clearly, with such a fast wide network we are latency limited, not bandwidth limited. The problem is much worse with a satellite link with a delay of 0.5sec.

14.11 Timestamps

RFC1323

The TCP header can contain an optional *timestamp*. This is a 32 bit value, not necessarily a time, that should be echoed unchanged back to the sender. The timestamp option is negotiated during the SYN handshake: if both ends include a timestamp option in their SYN segments then timestamps can be used for this connection.

This is a mechanism for the sender to make accurate RTT measurements, and therefore better estimates for the retransmission timeout. The timestamp value is simply a value that gets incremented once in a while, though the increment period should be typically between 1ms and 1s. A typical value is 500ms. The sender notes its system time when a timestamped segment is sent, and thus can compute the RTT when the timestamp returns.

RFC1323

The destination may choose to ACK more than one segment in a single reply: which segment's timestamp should it put in the reply? The timestamp used is the one for the next expected segment. So if the destination receives two segments

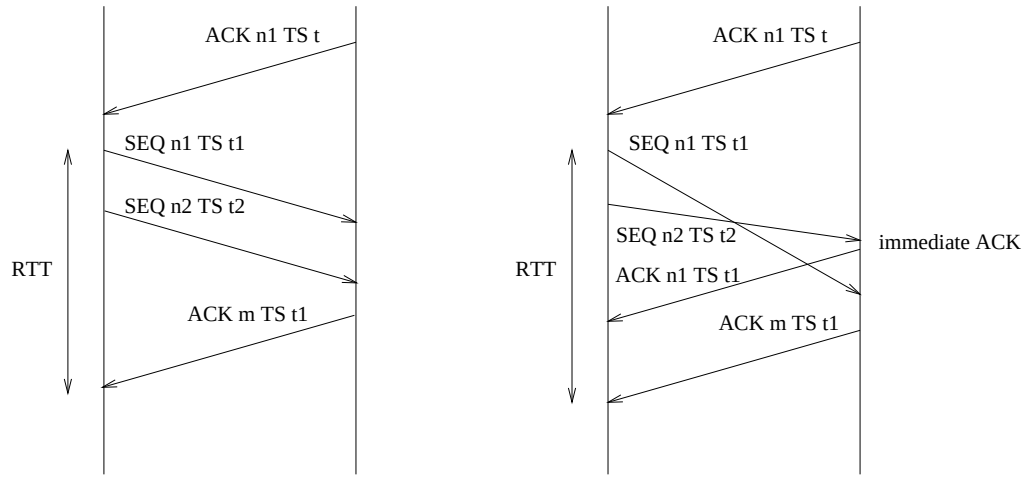


Figure 63: Timestamps

and sends one ACK, it uses the timestamp of the first segment. This is so the source can compute RTTs including the ACK delay. If a segment arrives out of sequence, the destination will still use the timestamp for the expected segment when it finally arrives.

RFC1323 PAWS also uses the timestamp option, as noted above.

14.12 Theoretical Throughput

An Ethernet is rated at 10Mb/s. When the physical, IP and TCP headers are taken into account, and the minimum inter-packet gap, and ACK packets, and so on, the theoretical maximum throughput of a TCP connection on 10Mb Ethernet can be calculated as 1,183,667 bytes/sec. Implementations have measured 1,075,000 bytes/sec which is pretty close. Faster networks are correspondingly better, but the final limitations are

1. the speed of light
2. the TCP window size

If you have a network running significantly slower than this, the problem is in the implementation, not TCP!

14.13 TCP for transactions

RFC1379 T/TCP: TCP for transactions, an experimental protocol. This tries to get the speed of UDP (few packets sent) with the
RFC1644 reliability of TCP. It is designed for use in *transactions*, specifically

1. Asymmetric: the two end points take different roles; this is a typical client-server role where the client requests the data and the server responds.
2. Short duration: normally, a transaction runs for a short time span.
3. Few data packets: each transaction is a request for a small piece of information, rather than a large transfer of information both ways.

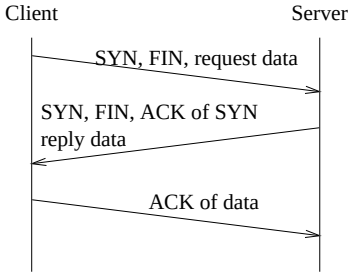


Figure 64: T/TCP

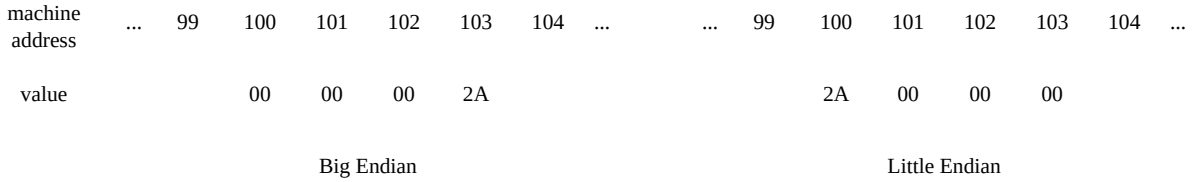


Figure 65: Endian

T/TCP is an extension of TCP, using the standard TCP header, but with some new options to indicate T/TCP. The trick is to put the request data in the SYN segment sent to the server, and get back the response in the SYN plus ACK of SYN in the reply to the client. Then one last ACK of the reply. This transaction takes three segments in the no-error case, only one more than UDP. A normal TCP connection takes three segments to establish; three for request, reply plus ACK of request, and ACK of reply; then up to four for closedown.

In case of lost segments, the normal TCP timeouts and resends apply.

T/TCP would be of great benefit to HTTP, the protocol used to fetch web pages.

14.14 TCP performance

TCP has been a major success. From 1200 bits/sec telephone lines to gigabit and beyond, it has turned out to be massively flexible and scalable. Not without a lot of work from a lot of people, though!

15 The Presentation Layer

The job of the Presentation Layer is to ensure data at one end of a connection is interpreted the same when it reaches the other end. Currently the biggest presentation problem is the byte order of representations of numbers on different machine architectures. An integer is typically represented in a machine by using four bytes: but how those bytes are used varies.

Some machines (e.g,. Sparc) use a *big endian* format. This stores the most significant byte (big end) at the lowest machine address, less significant bytes at increasing addresses. Other machines (e.g., Intel) use the *little endian* format, and store the least significant byte (little end) at the lowest machine address, and more significant bytes at increasing addresses.

This is the problem to address: if a machine receives four bytes 0000002A, does that mean the integer 42 or the integer 704643072 (hex 2A000000)? A typical solution to the this is to pick a single representation, the *network byte order*, and always transmit bytes in that order. When a machine wants to send an integer, it converts it into network byte order. When a machine receives an integer, it converts it to its native byte order. The *de facto* network byte order as

used on many networks is big endian. For a big endian machine, conversions of integers in and out of network order are trivial: nothing needs to be done. For a little endian machine, the conversions are to reverse the order of the bytes.

Functions `htonl` (host to network long) and `ntohl` (network to host long) exist to do this for us. Notice that when a little endian machine communicates with a little endian machine both machines still do the reversals. This is simpler than trying to negotiate endiannesses and having separate code for the reversal and non-reversal cases.

More generally, we have the presentation problem for all kinds of data. A particular problem is the different representations of floating point numbers. These are not simply restricted to byte orders, but also to internal formats like the numbers of bits used to represent exponents and mantissas.

RFC1832 The XDR package is a widely used collection of functions that convert data to a specific network format. XDR functions swap the order of bytes in integers if necessary, and convert between different floating point formats.

The IP layering model does not include a presentation layer, so presentation issues are not addressed by IP. Applications that run on top of IP must use something like XDR explicitly if they want to work properly between different architectures.

16 The Application Layer

There are very many applications that use TCP and UDP.

RFC854 1. Telnet. Logging in to remote machines for interactive use. First appeared in 1969. Uses TCP for reliability.

RFC959 2. File Transfer Protocol (FTP). Basic way of transferring files between machines. Over TCP.

RFC821 3. Simple Mail Transfer Protocol (SMTP). Email, the killer app for the Internet. Over TCP. The SMTP uses messages like `HELO`, `MAIL FROM:<Smith@Alpha.ARPA>`, i.e., human-readable rather than binary. This is for maximum interopability between different systems.

RFC1034 4. Domain Name System (DNS). UDP for (most) lookups, TCP for zone transfers. See Chapter 10.

RFC1035

RFC3010 5. Network File System (NFS). Allows remote disk to appear as local disk (transparent file access). Unlike FTP which only transmits whole files, you have random access to the remote file system. Older versions use UDP, while the latest versions use TCP.

RFC2616 6. HyperText Transfer Protocol (HTTP). For requesting and transferring Web pages. Again, uses human-readable commands, e.g., `GET http://www.bath.ac.uk`.

7. Many others. Finger, Whois, X Window System, ssh, timeserver, pop, imap, https, talk, Usenet news, print spooling, routing, etc. See `/etc/services`.

16.1 RPC and NFS

RFC1831

RFC3010

Sun's *Network File System* (NFS) is an approach to providing *transparent* file sharing between machines. By "transparent" we mean that a program will be unaware whether a file it is using is local to its machine or is actually on some other machine elsewhere across the network. Other file sharing systems exist (e.g., SMB, AFS) but NFS is by far the most widely adopted system.

RFC1831 First, though, we must address the topic of *Remote Procedure Call* (RPC). Sometimes we wish to execute a function or program on some other machine (a *remote server*), perhaps in a parallel program, or perhaps to gather information

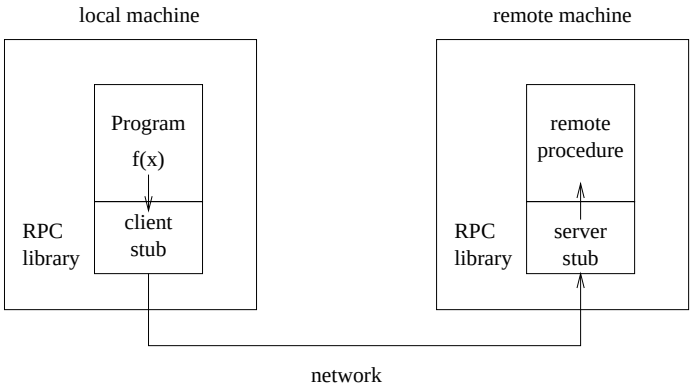


Figure 66: RPC

from the remote server. RPC is an abstraction layer (also invented by Sun) designed to make calling functions on remote machines more-or-less as simple as calling functions on the local machine. In practice it's not as quite as simple as that!

RPC is typically layered on top of TCP or UDP, though theoretically it can be layered on top of any transport layer. If a program on the local machine wants to call a procedure on the remote machine, RPC does this:

1. the local function calls a function generated by the RPC system called the *client stub*
2. the client stub collects the arguments of the function call and packages them into a network message which it sends to the server together with enough information to describe which function we want to call on the server
3. a *server stub*, also generated by the RPC system, on the server receives the message, decodes it and calls the appropriate function
4. the results are similarly packaged by the server stub and sent back to the client
5. the client stub receives the results message, unpacks it, and returns the results to the original calling function.

A utility, `rpcgen`, exists to help write the client and server stubs.

The programmer's view of the world is that they called a function, and got some results. The details of the networking are hidden, in particular things like timeouts and retransmits over UDP, presentation issues (RPC uses XDR), and so on.

One particular RPC service is called the *portmapper*. This purpose of this service is to provide a mapping from RPC service to UDP/TCP port number. If a program wishes to contact, say, the network file server `nfs` it must know the port number that service is listening on. When the `nfs` program starts it registers itself with the portmapper (i.e., tells it that it running) and provides to the portmapper its port number, 2049, say.

Now when a client want to find the status on our server it first contacts the portmapper (which always runs on port 111) saying "where is nfs?" and gets the reply "port 2049". Now the client can directly contact the status server.

This rather roundabout way of doing things allows for a dynamic configuration where services come and go and listen on a variety of port numbers: this all works as long as the portmapper is running!

A portmapper service is identified by a triplet of numbers:

1. Program number. This is a number which tells us which service we want. For example, `nfs` has assigned number 100003.

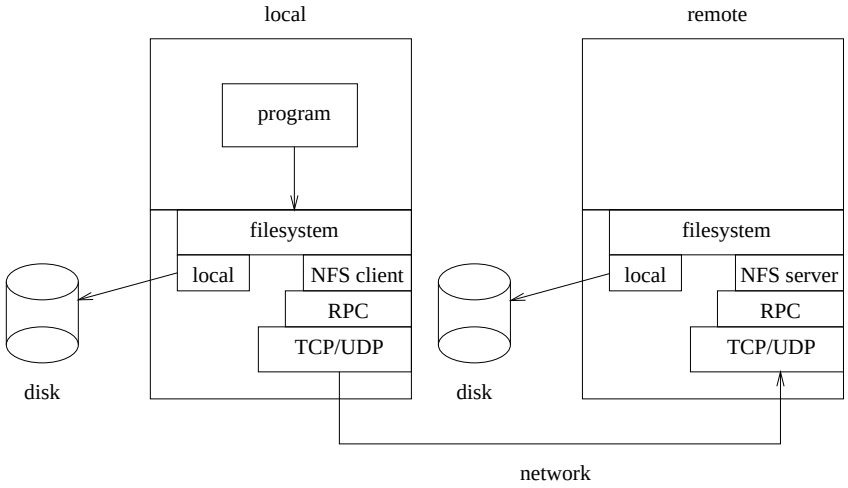


Figure 67: NFS

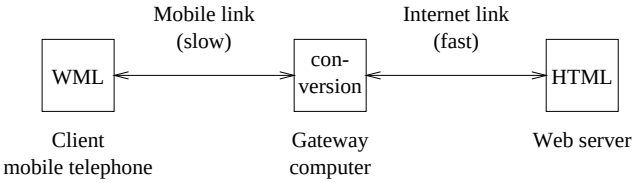


Figure 68: WAP

2. Program version. There can be different versions of the same service, perhaps with different protocols. For example, versions 2 and 3 of `nfs` are both commonly used. Note that we can provide more than one version of a service simultaneously as this number will tell us which version is required.

3. Procedure number. A service can offer several operations. For example, `nfs` allows us to create, delete, read, etc., files, and this number chooses which thing to do.

The `rpcinfo` program will give details of the RPC services available on a machine. Try `rpcinfo -p machine`. Also tt /etc/rpc lists the program number to program name association.

16.2 NFS

NFS sends RPC requests to remote server.

17 Other Bits and Pieces

17.1 WAP

The *Wireless Application Protocol*, as used in the emergent “next generation” mobile telephones.

This tries to tap into the information content provided by the WWW, but recognises the limitations of mobile devices:

- slow interconnect speed

- limited resolution of display
- limited computational ability
- limited memory

It is pointless downloading a 1600×1200 pixel, 16 million colour image over a slow link if you have a 100×100 pixel black-and-white display. Instead of communicating directly with a Web server, a WAP client talks to a *WAP gateway*. The gateway relays a request from the client to the server, and receives the reply. It then performs *content conversion* on the reply: this includes transforming from the server's HTML to WML, the simplified and compressed form of HTML that WAP clients run; and compressing images. Compressing an image might involve converting to black-and-white and reducing the resolution to (say) 100 by 100 pixels. This very much smaller version of the original (maybe KB instead of MB) is sent over the slow link to the client. The client benefits since this this is very much faster than downloading the original and then downgrading it itself.

Also a WAP client can run WMLScript, analogous to JavaScript. The gateway does the compilation of the WMLScript code into a compact byte stream which the client then executes.

WML is an instance of XML. The WML is transmitted over the air in a compressed binary format rather than a human-readable ASCII format.

17.2 Attacks

17.2.1 SYN flooding

This is a denial of service attack. A TCP connection starts with a SYN from the client. The server notes this and ACKs with its own SYN. Also, the server saves a chunk of information about this potential new connection (e.g., information to recognise the third ACK of the three way handshake). A SYN flood is where the attacker sends very many active open SYN segments, and never completes the three way handshake. This consumes resources on the server that are not released until a suitable timeout period has passed. The server can run out of memory space, and thus is unable to service real TCP connection requests.

There have been several solutions suggested. When resources are low we can start dropping half-open requests in some sensible manner, e.g. oldest first, or at random. A real connection request may be dropped, but the probabilities are that it will get accepted eventually.

An alternative is to use *syncookies*. This stores no information about the putative connection on the server, but cleverly encodes it in the server's initial sequence number for this connection. When the second ACK comes back, its ACK value can be decoded to tell us which connection it refers to. This method is good as the server can never run out of resources but is tricky to get right as 32 bits of sequence number is not a lot to work with, and the value must be carefully encrypted, or else the connection will be open to spoofing.

17.2.2 Distributed Denial of Service

This is a way to SYN flood. Many thousands of poorly protected hosts are subverted by crackers: these hosts are called *zombies*. At a signal all zombies send (SYNs for) HTTP requests to a server, e.g., Yahoo. This overload severely reduces the level of service for other users, often down to zero.

The World Trade Organisation meeting at Seattle's website was overloaded in a similar way: but the zombies were actual people.

17.2.3 Ping of Death

Some machines were vulnerable to oversized ping packets. These are much larger than the MTU, and overflowed kernel buffers when the host tried to reassemble the fragments: the writers of the code never expected ICMP packets to be that large. The probable result is a crash.

We can also generate this by constructing a fragment of length (say) 1000 and fragment offset 65400. This would imply a packet of length greater than 65535 bytes, which is the longest a packet can be (16 bit header field).

The easiest solution is to ignore ICMP packets that are larger than the MTU. Real ICMP packets are never large enough to require fragmentation, so we can safely drop ICMP fragments. Of course, you could fix the reassembly code, too.

In a similar vein, some systems could not cope with *fragment bombs*. This is when a large number of packets fragments are received, but all the packets have fragments missing, and therefore cannot be reassembled. Again, a lack of resources leads to problems.

17.2.4 Others

Mostly problems with Microsoft systems.

- Win Nuke. Out of band data sent to a listening port.
- Jolt (aka sPING). Fragmented ICMP packets.
- Land attack. Source IP addresses on TCP SYNs spoofed to be same as destination. Destination tries to respond to itself.
- Teardrop. Overlapping fragments cause problems on reassembly.
- New Teardrop (aka Bonk, Boink, Teardrop2): overlapping fragments of a UDP packet reassemble to form a whole packet with an invalid header.
- Bandwidth attack. Simply send so much data the destination cannot cope. E.g., Smurf. A ping packet is sent to broadcast address of victim network. The replies flood the network. The source address is set to some other victim: the replies flood back there, too.
- Zero length fragments. These were saved but never used, and eventually used up all memory.

And so on.

18 Security and Authentication

IP was not designed with security in mind. Quite the reverse: as it developed in the academic environment it was always assumed that all parties involved were benign. So an email message is readable as it progresses hop-by-hop to its destination. Some liken (unencrypted) email to sending postcards: everyone who haldes a postcard can read it. In fact, it is much worse that that, as it is easy to implement programs that automatically read emails or other data in a fast and transparent manner.

The Regulation of Investigatory Powers Act (RIPA) in the UK allows the police or government to install snooping equipment in any ISP.

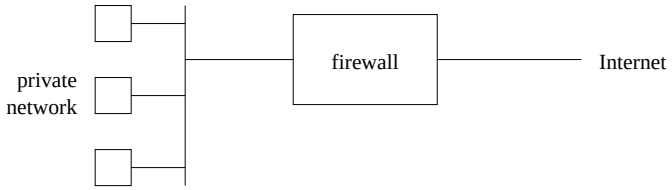


Figure 69: Firewall

These days, things are very different as a lot of valuable and sensitive data are transmitted over many and varied networks.

Security can be applied at any layer. For example, take the IP model.

- Application. The application provides security by encrypting the data itself. For example, you might encrypt an email using PGP before sending it. RFC2440
- Transport layer (TCP/UDP). If trusting the user to encrypt messages is too problematic, we might get the transport layer to encrypt for us. Note that this will be transparent to the application. Protocols like *secure socket layer* (SSL, developed by Netscape) and *transport layer security* (TLS), an evolution of SSL, exist to do this. RFC2246
- Network layer (IP). Even lower, we have IPSec, as discussed in section 6.18. RFC2406
- Data link. We can use security even in the data link layer. For example, when PPP sets up a new connection it can use cryptography to ensure that the user dialling in is authorised to connect. RFC1994

Encryption is just a small part in making a system secure as many other factors must be taken into account, particularly human factors. There is no point using military grade encryption software if you have an easily guessable password.

18.1 Firewalls

A *firewall* is a router that sits between a private network and the wider Internet and tries to protect the private network from attacks from the outside world by looking at each packet as it goes through and making a decision on what to do with it.

For example, supposing we wish to disallow FTP connections to machines on the private network: perhaps we are uncertain if FTP servers can be made safe from external hacking. The firewall can be instructed not to relay TCP packets to the private network that have destination port 21, the FTP port. So even if a FTP server is running on a private machine, no machine outside the firewall can connect to it as the packets simply never get there. Notice that other machine *inside* the firewall *can* connect.

Conversely, there can be outwards filtering. For example, we may wish to prevent a potentially virus or worm infected machine from connecting another machine to replicate itself. This could be achieved by blocking certain ports on packets travelling outwards. Outwards filtering can be also use to enforce policy, such as barring connections to file-sharing programs like Napster.

Configuring a firewall correctly is a difficult business, and is not something to be taken lightly: it is easy to convince yourself you have a secure firewall just when you have overlooked some unusual combination of source and destination and service.

Firewalling in conjuction with NAT (see section 6.16.2) is a particularly powerful combination.

A Example Programs

Simple clients and servers that talk using TCP and using UDP.

A.1 TCP Server

This listens on a port for a connection, and sends a simple message.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>

#define PORT 12345
#define MESSAGE "hello"

int main()
{
  int sock, conn, clilen;
  struct sockaddr_in server_addr, client_addr;

  /* create a STREAM (TCP) socket in the INET (IP) protocol */
  sock = socket(PF_INET, SOCK_STREAM, 0);

  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

  /* create server address: this will say where we will be willing to
     accept connections from */

  /* clear it out */
  memset(&server_addr, 0, sizeof(server_addr));

  /* it is an INET address */
  server_addr.sin_family = AF_INET;

  /* the server IP address, in network byte order */
  /* accept connections from ANYwhere */
  server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

  /* the port we are going to listen on, in network byte order */
  server_addr.sin_port = htons(PORT);

  /* this socket is going to listen to connections from this
     address (ANY) on this port */
  if (bind(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
    perror("bind failed");
```

```
    exit(2);
  }

  while (1) {

    /* now set this socket listening */
    if (listen(sock, 5) < 0) {
      perror("listen failed");
      exit(3);
    }

    /* now wait until we get a connection */
    printf("waiting for a connection...\n");
    clilen = sizeof(client_addr);
    conn = accept(sock, (struct sockaddr *)&client_addr, &clilen);

    if (conn < 0) {
      perror("accept failed");
      exit(4);
    }

    /* now client_addr contains the address of the client */
    printf("connection from %s\n", inet_ntoa(client_addr.sin_addr));

    printf("sending message\n");

    write(conn, MESSAGE, sizeof(MESSAGE));

    /* close connection */
    close(conn);
  }

  return 0;
}
```

A.2 TCP Client

This makes a connection to the server, and reads the message it sends.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>

#define PORT 12345
#define SERVADDR "127.0.0.1"

int main()
{
  int sock;
```

```
  struct sockaddr_in server_addr;
  char buffer[1024];

  /* create a STREAM (TCP) socket in the INET (IP) protocol */
  sock = socket(PF_INET, SOCK_STREAM, 0);

  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

  /* create server address: where we want to connect to */

  /* clear it out */
  memset(&server_addr, 0, sizeof(server_addr));

  /* it is an INET address */
  server_addr.sin_family = AF_INET;

  /* the server IP address, in network byte order */
  inet_aton(SERVADDR, &server_addr.sin_addr);

  /* the port we are going to send to, in network byte order */
  server_addr.sin_port = htons(PORT);

  /* now make the connection */
  if (connect(sock, (struct sockaddr *)&server_addr,
      sizeof(server_addr)) < 0) {
      perror("connect failed");
      exit(4);
  }

  printf("reading message\n");

  read(sock, buffer, 1024);

  printf("got '%s'\n", buffer);

  /* close connection */
  close(sock);

  return 0;
}
```

A.3 UDP Server

This waits on a port for a datagram, and returns a simple message.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
#include <string.h>

#define PORT 12345
#define MESSAGE "hello"

int main()
{
  int sock, clilen;
  struct sockaddr_in server_addr, client_addr;
  char buffer[1024];

  /* create a DGRAM (UDP) socket in the INET (IP) protocol */
  sock = socket(PF_INET, SOCK_DGRAM, 0);

  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

  /* create server address: this will say where we will be willing to
     accept datagrams from */

  /* clear it out */
  memset(&server_addr, 0, sizeof(server_addr));

  /* it is an INET address */
  server_addr.sin_family = AF_INET;

  /* the server IP address, in network byte order */
  /* accept datagrams from ANYwhere */
  server_addr.sin_addr.s_addr = htonl(INADDR_ANY);

  /* the port we are going to listen on, in network byte order */
  server_addr.sin_port = htons(PORT);

  /* this socket is going to accept datagrams from this
     address (ANY) on this port */
  if (bind(sock, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
    perror("bind failed");
    exit(2);
  }

  while (1) {

    /* now wait until we get a datagram */
    printf("waiting for a datagram...\n");
    clilen = sizeof(client_addr);
    if (recvfrom(sock, buffer, 1024, 0, (struct sockaddr *)&client_addr,
 &clilen) < 0) {
      perror("recvfrom failed");
      exit(4);
    }
```

```
    /* now client_addr contains the address of the client */
    printf("got '%s' from %s\n", buffer, inet_ntoa(client_addr.sin_addr));

    printf("sending message back\n");

    if (sendto(sock, MESSAGE, sizeof(MESSAGE), 0,
       (struct sockaddr *)&client_addr, sizeof(client_addr)) < 0) {
      perror("sendto failed");
      exit(5);
    }

  }

  return 0;
}
```

A.4 UDP Client

This sends a datagram to the server, and reads the message it returns.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>

#define PORT 12345
#define MESSAGE "hi there"
#define SERVADDR "127.0.0.1"

int main()
{
  int sock, clilen;
  struct sockaddr_in server_addr, client_addr;
  char buffer[1024];

  /* create a DGRAM (UDP) socket in the INET (IP) protocol */
  sock = socket(PF_INET, SOCK_DGRAM, 0);

  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

  /* create server address: where we want to send to */

  /* clear it out */
  memset(&server_addr, 0, sizeof(server_addr));

  /* it is an INET address */
```

```
  server_addr.sin_family = AF_INET;

  /* the server IP address, in network byte order */
  inet_aton(SERVADDR, &server_addr.sin_addr);

  /* the port we are going to send to, in network byte order */
  server_addr.sin_port = htons(PORT);

  /* now send a datagram */
  if (sendto(sock, MESSAGE, sizeof(MESSAGE), 0,
     (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
      perror("sendto failed");
      exit(4);
  }

  printf("waiting for a reply...\n");
  clilen = sizeof(client_addr);
  if (recvfrom(sock, buffer, 1024, 0, (struct sockaddr *)&client_addr,
       &clilen) < 0) {
      perror("recvfrom failed");
      exit(4);
  }

  printf("got ’%s’ from %s\n", buffer, inet_ntoa(client_addr.sin_addr));

  /* close socket */
  close(sock);

  return 0;
}
```

Index