Networking

MJ Rutter
mjr19@cam.ac.uk

Lent 2006

Networking

- **L**ocal **A**rea **N**etworks
- **W**ide **A**rea **N**etworks
- Software

The Structure of the Internet

The structure of the internet is very similar to that of the postal service. One can quickly discern different mechanisms for transport depending on the distances involved. Local deliveries might be done by bicycle, more remote ones by van, and yet more remote by aeroplane. At each stage, the carrier knows sufficient to complete that stage and pass the package on to the next stage, but not enough to perform the final delivery.

And of course, competition exists at every level. For local deliveries, should one write ‘UMS’ on the thing and throw it into the internal mail, or should one hand it to the nearest undergraduate? Does either mechanism offer any guarentees against loss, interception or misdelivery?

Start Small

The first network we shall consider is the local network which directly connects a small number of computers. We shall consider the most common: ethernet. Ethernet itself describes a protocol which can exist in several physical forms: optical, copper twisted pair at 10, 100 and 1000 MBit/s, copper co-axial cable at 10 MBit/s, and various sorts of wireless to list a few.

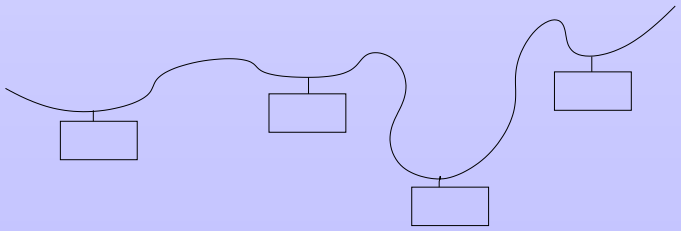
We shall start by considering in detail the oldest and simplest: co-axial cable. However, this also has much in common with wireless, so cannot yet be consigned to the dustbin of history.

10Base2

With a 10Base2 network (10 MBit/s, 2 hundred metres), all the networked computers are connected by a single co-axial cable. The co-axial cable acts as a standard transmission line of 50Ω impedence, and needs terminating correctly to eliminate reflections.

Such a network has obvious physical limitations. As there is a single wire forming the backbone, everything is transmitted to all computers. This is simple: it needs no electronics outside of the computers themselves. However, it does mean that the 10 MBit/s are shared between all computers on the network.

It also suffers from *collision* problems: only one computer can transmit at any one time, but Einstein, and computers' slow reactions, decree that occasionally two computers will talk simultaneously, at which point both need to shut up for a random time interval before attempting to retransmit.



Packets

Data are transmitted in *packets*. Each packet contains a header recording information such as its length (ethernet permits the length to vary, some competing technologies used fixed-size packets), and a check-sum used to ensure that the data are not accidentally corrupted.

The packet also contains the address of both the sender and the intended recipient. These addresses are the six-byte ethernet (or MAC) addresses, usually written out as twelve hexadecimal digits. Although the packet will be sent to everyone, the network card will ignore packets not intended for it, and not pass them on to the operating system.

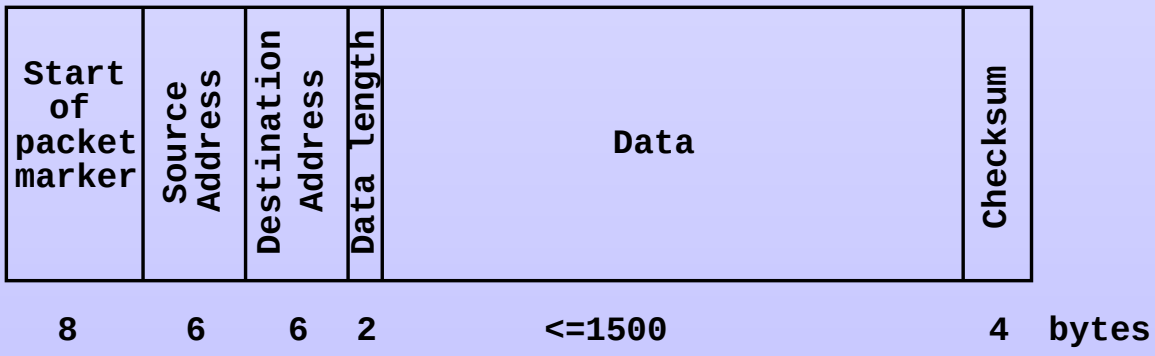
There is one special address, the *broadcast address*, which is intended for everyone. It is `ff:ff:ff:ff:ff:ff`.

MAC: Media Access Control

Flow control

Although anything using the network must be capable of receiving a full packet, it is not necessarily the case that it will be able to receive a long stream of them back-to-back. It might be simply unable to cope with data arriving that fast. How can it slow the sender down?

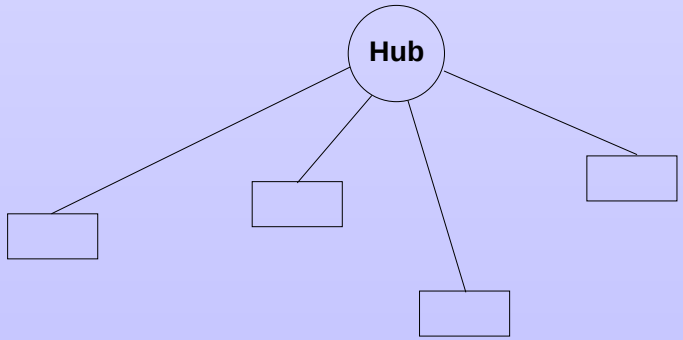
Simple. Send something itself, whilst the sender is transmitting. This will immediately cause a packet collision, and the sender will have to back off for a short, random time interval.



Snakes or Stars?

A single long cable snaking around a building can produce reliability problems. When someone unplugs a bit of it, finding the break is awkward. When someone wants to add to the network, making a break is awkward. There is also a maximum cable length, which is fairly modest (200m).

A more convenient, but more expensive, topology is a *star* topology. All cables are simply point-to-point links back to a central hub. The *hub* simulates the effect of the single long cable. Anything received on any of its ports is immediately transmitted back down all its other ports. Collisions happen as before.



CAT5

Tradition dictates that the cables are now the cheaper, more flexible, '*twisted pair*' copper cables. These contain four pairs of wires, each pair twisted together with a pitch of around one centimeter. This twisting eliminates dipole radiation from the pairs when they carry a signal, as the pitch is less than the wavelength of the signal, and so a piece of cable and the next piece which has been twisted by π radians are carrying the same signal, and thus radiate in a fashion which interfers destructively.

Just two pairs are used: one for transmitting, and one for receiving. Clearly the transmit pair on the hub needs to be connected to the receive pair on the computer, and this leads to fun with *straight through* versus *crossover* cables, although modern network equipment will automatically detect and correct for this sort of miswiring.

CAT5: CATegory 5 twisted pair ethernet cable, good for 100MBit/s. Also CAT3 (10MBit/s), and CAT5e (enhanced), 1GBit/s as 500MBit/s per pair. The standard covers the sockets and maximum lengths, as well as the cable itself.

Bright Stars

Intelligent ‘hubs’ can examine the source addresses on the ethernet packets they see, and learn who lives where. Then they can send packets to just the port to which their destination is attached. This is called *switching*, and dramatically increases the capacity of the network, as now several pairs of machines can communicate simultaneously at the full speed of the network without seeing each other’s traffic. It also permits links to carry traffic in both directions simultaneously (full duplex).

Broadcast packets are still broadcast to all, as are packets to machines whose location the *switch* (as such a hub is called) has yet to learn. This probably means that the machine in question has not sent out a packet in the past ten minutes or so (perhaps because it has only just been connected, or because it does not exist!).

A switch might be connected to another switch. Fortunately switches can cope with the result: several ethernet addresses all appearing to live on the same port.

Inside a Switch

A switch has one other trick up its sleeve: it can connect machines talking at different speeds. Unlike a hub, which transmits with no delay the instant each bit arrives, a switch's port will wait until it receives a whole packet. It will then forward this packet internally within the switch (using a very fast ring network), and then the outgoing port will transmit the thing.

There are disadvantages to this approach. The latency is slightly higher: the outgoing packet will not start to be transmitted until the whole of the incoming packet is received. For a 1.5KB packet at 1GBit/s, this is a 12μs delay. It also means that the destination might not be as fast as the source, which will inevitably cause packet loss if the source is sending packets one after another at full speed.

Indeed, switched networks always have potential for packet loss. If one has a 24 port switch, and the machine form 12 pairs which talk to each other, all is well. If 23 machines talk to the one remaining port (likely, if that port is actually the link to the ouside world) there may be considerable packet loss. (Often a link to the rest of the world, an 'uplink', on a 24 port switch is ten times faster than any of the other ports.)

Detection and Negotiation

The speed of a link can be detected trivially, as pulses arriving at 10, 100 and 1000 MBit/s are readily distinguished – very readily, as the first 54 bits of the ‘start of packet’ marker are alternating ones and zeros.

The duplex setting cannot be detected readily. Even if the other end transmits whilst it is being transmitted to, it is unclear whether it is supporting full duplex, or attempting flow control by provoking a collision.

Moderm network devices perform a brief negotiation when the link is started in order to determine the best setting which are supported by both ends. Unlike the negotiation performed by modems, they make no attempt to assess line quality, so if one tries to use 100MBit/s capable devices on poor-quality cable, they will not automatically back off to 10MBit/s.

Sane devices default to 10MBit/s half (i.e. not) duplex if autonegotiation fails (probably because the other end is too old to support it). Mad devices default to 100MBit/s full duplex.

Wider Still and Wider

A hub-based network becomes awkward with more than a dozen machines connected to it. The sharing of network bandwidth quickly becomes an issue.

Switches alleviate this problem considerably, and one can readily have a hundred machines on a single switched network. However, the broadcast traffic is a problem, and will stop such a solution looking sensible beyond about a thousand machines.

An ethernet address contains no information about a machine's location: a laptop will not change its ethernet address as it moves between offices, departments, universities or countries. For efficient wide-area networking, an address which does encode something about location would be useful.

Vendors 'guarantee' that an ethernet address is globally unique. It only needs to be unique on the local network.

A global authority assigns numbers to vendors in blocks of 2^{24}. If a MAC address starts 00:03:ba, then it is Sun, if 00:09:3d, it is Newisys, etc.

IP

IP (Internet Protocol) provides for global routing. Its addresses are assigned in blocks to specific networks. The networks are joined by *routers*, each of which may have several networks attached to it, and which needs to know which network a packet should be sent to to move it towards its destination. The choice is fairly simple, as there will often be a large category of ‘not around here’, which will cause the packet to be passed further up the hierarchy of routers.

The address on an IP packet is just four bytes long: a mistake, which causes people to worry about the resulting shortage of addresses in *IPv4* as the result is known. Tradition dictates that a numeric IP address is written bytewise in base 10, with periods separating the bytes, unlike the hexadecimal and colons of ethernet addresses.

Note that although an IP packet is often transported within the data segment of an ethernet packet, this is not mandatory. It is perfectly possible to use IP without ethernet, e.g. with ATM.

Local Routing

Any packet whose address starts `131.111` is destined for this University. That is all anyone outside of this University needs to know. That the University chooses to route addresses further once they are received, and that only those in the range `131.111.62.128` to `131.111.62.255` will end up in TCM, is a detail and complication entirely internal to the University. Similarly the router in the JANet network will know which addresses are within JANet, and which need to be passed to external routers.

It is very much like the good old days of UK regional telephone numbers:
01 - London (little else matters)
  235 - BELgravia
  546 - KINgston
  748 - Barnes (RIVerside)
  946 - WIMbledon

JANet: the UK's Joint Academic Network
Unfortunately UK 'phone numbers soon stopped being so entertaining once there was a shortage of free prefixes.

A Global Journey

```
tcma:~$ traceroute www.ox.ac.uk
traceroute to www.ox.ac.uk (163.1.0.45): 1-30 hops, 38 byte packets
 1  tcmr.phy.cam.ac.uk (131.111.62.234)  0.977 ms  0.976 ms  0.0 ms
 2  route-west-3.cam.ac.uk (131.111.62.190)  0.0 ms  0.977 ms  0.0 ms
 3  route-cent-3.cam.ac.uk (131.111.2.126)  0.976 ms  0.977 ms  0.0 ms
 4  route-enet-3.cam.ac.uk (192.153.213.193)  0.976 ms  0.976 ms  0.0 ms
 5  cambridge-bar.ja.net (146.97.40.49)  0.977 ms  0.0 ms  0.977 ms
 6  po11-0.lond-scr.ja.net (146.97.35.9)  4.88 ms  4.88 ms  4.88 ms
 7  po4-0.read-scr.ja.net (146.97.33.74)  5.86 ms  5.85 ms  5.86 ms
 8  po2-0.read-bar2.ja.net (146.97.35.106)  5.85 ms  4.88 ms  5.85 ms
 9  pos2-0.oxford-bar.ja.net (193.63.108.74)  5.86 ms  6.83 ms  5.86 ms
10  146.97.40.82 (146.97.40.82)  6.83 ms  6.83 ms  5.86 ms
11  www.ox.ac.uk (163.1.0.45)  6.83 ms !A  6.83 ms !A  6.83 ms !A
```

A trace going outside the civilised world will look similar for the first six stages, and then start to leave JANet. In 17 hops one can reach that small island of eternal civilisation which is surrounded by Rome. Those hops which are more than about 5km cannot be ethernet, for even over fibre ethernet has quite low distance limits.

Many commercial networks block `traceroute`, which can make diagnosing network problems hard. However, excessive use of `traceroute` will certainly annoy important people.

A Classless Society

In the Bad Old Days, there were only three sizes of networks: class A (first byte specified, last three not), B (first two bytes fixed), and C (only last byte unspecified). This assumed that all IP networks would fit neatly into boxes of size 16 million, 65 thousand or 256.

These days the division between the part of the IP address which is constant for all machines within a network, and the part which varies, is done at the bit level, permitting network sizes of any power of two. Thus TCM has half a class C (in the old language), or 131.111.62.128/25 in the new, the /25 meaning that the first 25 bits are fixed in this range, and just the last seven vary.

IP Meets Ethernet

Every computer needs a small routing table to tell it whether addresses are local or remote. If an address is local, then it can be delivered directly by ethernet, if not, then it will need to be sent to a router (which must exist on the local network) for passing on to a remote network. A router performing this role is often called a *gateway* for obvious reasons.

Every time an IP packet passes through a router the number in its ‘time to live’ field is decremented. Should this reach zero (it usually starts at 64 or 128), the packet is thrown away, on the assumption that a routing loop has just been discovered. An error message is returned for this condition, and it provides a mechanism for finding which routers a packet will pass though: first set the TTL field to one, send, record who sends back the error message (the first router). Then try again with the TTL field set to two, etc.

This is approximately what the `traceroute` command does.

ARP

The routing table will reduce any IP address to one which is on the local network. However, this is still an IP address, not an ethernet address, so is still useless. A translation occurs using the Address Resolution Protocol.

A special ethernet packet is broadcast containing the question ‘who has a given IP address?’ Precisely one response is expected, this time unicast, saying ‘me!’ The machine which sent the question now knows which ethernet address wants the traffic for a given IP address. It will remember the answer for a while, to save having to repeat the broadcast for every packet it needs to send.

Security?

There is no security in the above procedure. The first person to respond saying ‘give me all traffic for IP address `w.x.y.z`’ is believed and trusted. This can be useful. If someone erroneously configures their computer to believe that Hermes is on TCM’s local network, our router will see the ARP request, reply to it pretending to be Hermes, and forward the traffic correctly. This is called *proxy-arp*.

There is a mallicious form, when a computer deliberately and unnecessarily uses false ARP replies to redirect traffic through itself, traffic which it can then analyse. Of course, with hubs, or wireless, or 10Base2, such tricks are unnecessary: everyone sees all the traffic anyway.

Untrusted computers on a local network are a menace (and ‘untrusted’ may simply mean hacked and under control of persons unknown).

ICMP

The simplest IP packets are those of the Internet Control Message Protocol.

There are several of these, the one which is most familiar is ‘Echo Response,’ which is responsible for replying to ‘ping’ requests. Others include a time to live exceeded message, if a router ever decrements a TTL field to zero.

One can use ‘`ping`’ to attempt to diagnose networking problems. By default (on most systems) it sends one packet per second for ever, and when it is terminated (control ‘C’, ‘`kill -INT`’), it prints a summary of packets sent, packets received, and average round-trip time.

Please *don’t* leave it running unnecessarily: it places a greater load on the network than you understand, and doing so will considerably annoy network administrators when they notice, even assuming that you don’t have a broken version which sends reverse DNS requests for every packet received. Used sparingly it can be extremely useful.

Ports

UDP and TCP also add another two bytes to their source and destination addresses, which form a 16 bit *port number*. This sort of port has nothing to do with a port (or socket) on a network switch, and the reuse of the name causes much confusion.

The port number is used for routing within the computer. Anything addressed to port 80 will be sent to the program which registered that it wanted to receive on port 80: probably the WWW server. Similarly port 22 will find the ssh server, port 25 an SMTP server (outgoing email), etc. Source ports are assigned (mostly) at random, but each TCP connection is uniquely identified at any one time by the tuple of source IP address with port and destination IP address with port. In other words, if a single machine makes two simultaneous ssh connections to the same server, although the destination IP address and port will be the same for both, as will the source IP address, the source port address will still distinguish them.

UDP

UDP, which is one of the simpler IP protocols, adds little more than this. From ethernet it has gained global routing and port numbers. It retains the concept of disconnected packets, with no safeguards against loss. If a reply is expected but not received, there is no way of telling whether the loss occured on the outward leg, the return leg, or whether the remote server ‘forgot’ to respond. UDP does, however, add its own checksum to the data.

The concept of a broadcast still exists. On each local IP network there will be one broadcast address meaning ‘everyone’. This can be useful for finding certain UDP-based services automatically. These tricks are more common in the Windows world than the UNIX world. Sensible people frown on broadcasts: they are expensive as the packets need to be sent to every computer on the local network, and they are generally insecure, as asking a hundred computers ‘which of you should I trust to tell me what my root password is?’ (yes, really, I have seen this several times) is silly. It takes the fun and challenge out of hacking.

UDP: User Datagram Protocol, sometimes known as Unidirectional DP, or Unreliable DP.

Independence

TCP/IP makes no assumptions about the underlying network. It too uses packets, with checksums to guard against corruption. The TCP part also has the concept of a *connection*. Whereas ethernet packets are entirely unrelated to each other, TCP packets may form what appears to be a dedicated bidirectional link.

To do this they need an order (for packets are not necessarily received in the order in which they were sent: they may take different routes), a mechanism for detecting and retransmitting lost packets, and a mechanism for acknowledging the receipt of packets. TCP also has its own mechanism for indicating that the other end should slow down. The header of a packet now contains a sequence number, and many flags.

Whereas with ethernet there is no way of telling if a packet was received, with TCP there is. However TCP has no concept of a broadcast: it constructs single point-to-point links.

Funny Handshakes

A rough outline of the procedure to set up a TCP connection is:

```
client:port  ->   SYN   ->   server:port
client:port  <- SYN/ACK <-   server:port
client:port  ->   ACK   ->   server:port
```

with only the final packet containing useful data. The closing procedure, which can be initiated from either side, is

```
client:port  ->   FIN   ->   server:port
client:port  <-   ACK   <-   server:port
client:port  <-   FIN   <-   server:port
client:port  ->   ACK   ->   server:port
```

SYNchronise, ACKnowledge and FINalise are three of the possible flags which may be set in TCP headers.

SYN Attacks and Scans

An OS will have a limit on the number of simultaneous TCP connections it can handle, as each will need the kernel to reserve space to record its state and to assemble packets.

Once a SYN arrives, space must be reserved to handle the connection, although the relevant application or daemon will not be notified that anything has happened until the ACK arrives and the connection is fully estabilished. If the ACK never arrives, the connection remains half open until it times out after a couple of minutes.

If one floods a computer with SYN packets, it will be unable to accept new connections until some time out: a simple denial of service.

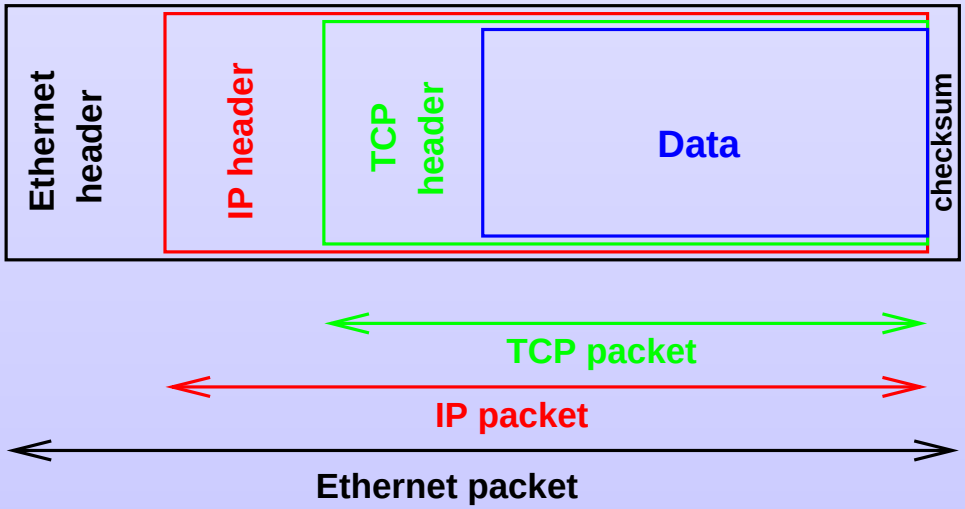
If one scans through all ports, and records which respond with SYN/ACK, one can construct a list of which ports have services associated with them.

Neither of these can be logged at the application level, but either may be logged by the kernel or a firewall.

Either would breach the AUP for the CUDN, so don’t think about it.

Errors and Encapsulation

If a connection is attempted to a port on a server which has no process listening for incoming connections, an immediate RST is sent, as it would be in response to any packet which is clearly bogus, such as a packet other than a SYN packet arriving for a connection which is not estabilished.



RST: ReSeT

More Encapsulation

TCP provides for a maximum data length of 64K, but a TCP packet wholly contained in an ethernet packet must fit its data, and both the IP (20 bytes) and TCP header (28 bytes) sections, into the 1500 byte maximum data length of an ethernet packet.

One solution is to fragment the TCP packet across multiple ethernet packets, and then reassemble the fragments on receipt. Indeed, the IP layer provides this functionality.

However, it is not ideal. TCP's ability to check and request retransmission of mislaid packets works on whole TCP packets only, not fragments. If a TCP packet is split into ten fragments, and one gets lost, all ten need retransmitting.

The details of how often to request retransmits, and how long to wait, vary between implementations. If short, dead connections are quickly detected and terminated, but there will be unnecessary connection losses on poor networks.

TCP overheads

If one believes that the maximum size for a TCP packet is 1500 bytes including data (a common view: setting the “MTU” to 1500 is a widespread practice), then one has 1452 bytes of data in a 1518 byte ethernet packet. If the data transfer is unidirection, there still needs to be an acknowledgement packet for each packet transmitted, and this packet will be 66 bytes. So a 4.5% data overhead in each direction.

Both an ethernet and a TCP checksum will have been calculated for the data.

If the data transfer was short, the overhead of starting and finishing the connection (the SYN and FIN packets) could be (very) significant too.

Still, TCP was standardised in 1981, and is in universal use 25 years later. It isn’t going away soon.

In some cases, an MTU of below 1500 is appropriate. This is particularly the case for IP in IP tunnelling tricks, which may occur with VPNs and certain ISPs.

Extensions to permit a single ACK to cover multiple packets are frequently discussed, but have not yet happened. Every time you download a double layer DVD of data, you send enough ACK traffic to fill over half a CD. In the Bad Old Days of simplex networks, ACK traffic was a significant source of collisions.

Packet Sizes

A maximum packet size of 1500 bytes seemed reasonable in the 1980s. Such a packet takes over 1ms to transmit over 10Base2, for which time the shared network is monopolised by whoever is transmitting it. The network card will need at least that much memory in its receive buffer, the operating system will need to be able to recieve packets of that size from the network card, and most things that needed transfering would not take that many packets.

Today it looks silly. The links are effectively point-to-point, so one cares less about monopolising a shared resource, and 1500 bytes takes just 12μs to transmit at 1GBit/s. The OS cannot handle data efficiently when it is broken into such small pieces, each requiring separately checksumming, copying and generally juggling, with an interrupt being generated for each incoming packet. Ethernet is usually carrying IP which could cope with much larger packets.

Of course, when faced with an analogue modem, 1500 bytes is a lot: at least 0.2s. A maximum packet size of around 600 bytes can be more reasonable.

Giants and Jumbos

An extension called ‘jumbo frames’ allows ethernet packets with a data length of up to 9200 bytes. This can significantly improve performance, particularly NFS performance, but it requires *all* ethernet devices on the network to support the feature. (The checksum used by ethernet becomes less effective beyond about 11,000 bytes.)

Sometimes a mere baby giant, rather than a jumbo, is sufficient. A baby giant is just a few bytes larger than the standard ethernet packet, and allows either for extra fields in the ethernet header, whilst retaining a 1500 byte data section, or encapsulating the whole of a standard ethernet packet in the data section of a slightly enlarged packet.

Routers can route IP packets between networks with different maximum packets sizes (MTUs: Maximum Transfer Units). They will fragment packets if neccessary, and, more rarely, reassemble packets when moving to a network with a higher MTU.

Humans

Humans dislike dealing with numeric IP addresses, and prefer textual names. A lookup system exists for converting from one to the other: the DNS (Domain Name Service). This is a distributed database with a strict hierarchy, but much redundancy and caching. The servers in Cambridge will resolve Cambridge addresses for anyone, and any address for machines in Cambridge. The ‘any address’ bit they will do by passing the request on, and is not a service which they will offer to the whole world.

The service is very hierachical. There is a list of servers for dealing with `.uk`, those servers will be able to reveal who deals with, for instance `.ac.uk`, those `.cam.ac.uk`, and there the buck stops (in most cases).

A single name may map to several IP addresses (for load balancing), and multiple names may map to a single IP address (to amuse humans), although in the latter case there must be one canonical name which the reverse loop-up (number to name) will always return.

The Errors of the DNS

Getting an IP address out of a DNS is simple. One sends the local DNS server a query (a single packet), waits, and then back comes the answer. It may have had to pass on the question to a succession of other servers, spread across the world, but that is its problem.

Only it isn't. The protocol for talking to DNS servers is UDP-based: the exchange of data was thought too simple to warrent the overheads of TCP. But one might have to wait tens of seconds for a response if the request does have to go through a succession of remote servers, and yet there is no way of telling whether one is waiting because of a slow server, or because the request got lost. Timeouts before requests are repeated are typically long, and as soon as network congestion causes packet loss, performance suffers badly.

DNSes need to be very reliable. Email sent to a broken email server will retry for days, but, if the DNS server is unresponsive, it will fail immediately reporting an invalid address. Sane institutions use multiple, well-separated DNSes – we mirror MIT, and vice versa, for instance. Tin-pot polytechnics put all their DNSes on the end of the same single cable.

Private addresses

IP addresses are, generally, global. This can often be somewhere between unnecessary and unhelpful. Should TCM's printers really be addressible directly from Australia, and be using up some of those scarce 2^{32} possible addresses?

Fortunately several ranges of addresses have been designated as *private*: never to be seen ouside of an institution, nor in the global DNS. Thus these addresses can be used independently within each institution, and no packets from other institutions can ever be sent to them. This improves security, and frees up addresses.

As 'institution' merely means a cooperating group whose network is contiguous, some of these networks are private and JANet-wide, some Cambridge-wide, and some TCM-wide.

The standard which defines the private address ranges is RFC1918. Hence such addresses are sometimes called 1918 addresses.

DHCP

The Dynamic Host Configuration Protocol is the usual way of providing mostly-automatic network configuration for roving laptops. The laptop sends out a request, a broadcast ethernet packet, saying ‘who am I?’ The DHCP server kindly tells it who it is, where its gateway is, where its DNS is, and what machines are on its local network.

Security? None: standing in the middle of the street shouting ‘who am I, and whom should I trust’ is naïve at best. However, one should use the reply, not trust it. . .

DHCP believes that addresses are leased, with no freehold given. The lease is typically for four hours. If no attempt is made to renew the lease after that time, the address may be given to another machine: hence the ‘D’ in DHCP. One is intended to consider a pool of addresses, a larger pool of laptops, and the hope that not all laptops are connected simultaneously.

Zero Config Networking

Modern versions of MacOS and Windows, if given neither a fixed IP address nor a DHCP server, attempt to find an IP address for themselves using a protocol called ‘Zero Configuration Neworking.’ It might help on small home networks, but it is a menace elsewhere.

The theory is that the machine chooses an IP address in the private range. As such an address is not globally unique, but unique only on the local network, a local broadcast will discover whether it is in use of not. So the machine send out an ARP request for it, to see if anyone responds. (Responses to ARP requests are sent to the sender’s ethernet address, not IP address, so there is not catch-22 situation here.)

If no response is received, the computer believes the address is free, and takes it. If a response is received, the computer picks another IP address and repeats. Once it has an IP address it hopes that it can find other resources (the ‘Network Neighbourhoood’ in the language of a certain OS) by sending UDP broadcasts.

Zero Problems

There are many. ARP requests and replies may be lost, so one needs to test several times before concluding an address is free. Not all implimentations do this.

The scope for multiple machines on the same network ending up with identical IP addresses is rather large.

A common configuration of routers/gateways is to reply to all ARP requests which are not for addresses on the local network. Unless they know about this private range, they will respond to all addresses in it, and the poor machine will never find a free one, but drive everyone mad whilst it looks.

Humans and Private Addresses

Humans spend most of their time looking at WWW pages, doing email (which can usually be presented via a WWW browser), and accessing machines in their own institution. All this is trivially doable from a private address, provided that a WWW cache and DNS server are accessible.

Sometimes a more direct TCP connection out of a private range is useful. This can be achieved automatically via a NAT server (Network Address Translation). The NAT server effectively acts as a router, but changes the source IP address to its own. Thus it will receive the reply directly, and then it changes the destination address of the reply back to the originator and forwards it on. The result is that several machines successfully hide behind a single NAT server, and just the NAT server needs a public IP address. As a single machine can cope with a few tens of thousands of simultaneous TCP connections, one can place quite a large network behind a single NAT server.

IP in Practice

A computer trying to send out an IP packet will follow this rule set:

Check down its routing table to find the narrowest network which matches the IP address given. The ‘catch-all’ default network of 0.0.0.0/0 will match everything, but with the lowest priority.

Is the result of this match a remote gateway? If so, repeat using the address of the gateway.

We must now have an IP address which is local to a network to which the computer is directly attached. If the IP address is the broadcast address for that network, send to the ethernet broadcast address. If it an address for which the corresponding ethernet address is cached in our ARP cache, send to that ethernet address. Else send an ARP request for it, and on receiving a reply, send to that ethernet address.

A Routing Table

```
Kernel IP routing table
Destination   Gateway         Genmask         Flags MSS Window irtt Iface
131.111.62.0  0.0.0.0         255.255.255.0   U       0 0          0 eth0
172.24.25.0   0.0.0.0         255.255.255.0   U       0 0          0 eth0
127.0.0.0     0.0.0.0         255.0.0.0       U       0 0          0 lo
0.0.0.0       131.111.62.62   0.0.0.0         UG      0 0          0 eth0
```

The network 127.0.0.0/8 (sometimes just 127.0.0.1) is the *loopback* network, used by programs running on the same computer which wish to talk TCP to each other. Addressing 127.0.0.1 always reaches the computer one is running on.

Flags: U - up, G - via a gateway.
MSS, Window and irtt are most uninteresting.
169.254.0.0/16 (deleted from the above): multicast, something this lecture is too short to cover.

Routers in Practice

A router will receive a packet which will be sent to the ethernet address of one of its interfaces, but to an IP address which is nothing to do with it. It will examine the IP address, work out which interface it should be sent out of, just in the same way that any other computer would, and then send the packet out again. It will change the ethernet source and destination addresses, but keep the IP contents unchanged, save for decrementing the 'Time To Live' byte.

One can consider that the router has removed the IP packet from the ethernet packet, and then re-encapsulated it. The operation is not completely trivial: it will checksum the incoming ethernet packet, discard it if the checksum does not match, and place a new checksum on the outgoing packet (the changes to the ethernet addresses and TTL fields will require a change to the checksum).

Fortunately most modern ethernet cards calculate these checksums in hardware, not using the computer's/router's main CPU, as, although checksums are simple, it takes some effort to sustain the 400MB/s of checksumming needed to keep a full duplex 1GBit/s network line saturated.

TCM in Detail

TCM's public IP address range is 131.111.62.128/25.
Its private Cambridge-wide range is 172.24.25.0/24.
Its gateway is 131.111.62.190 (or 172.24.25.62).

The gateway will proxy-arp and happily cope with those (many) machines which believe that TCM's range is 131.111.62.0/24 and that the gateway is 131.111.62.62. Historically this was the right answer anyway.

It is less happy with those machine which don't understand that both the above networks are local. Traffic within this local ethernet network should not go near the router. However, machines on 172.24.25.0/24 which believe that they need to send packets via the router to get to 131.111.62.128/25 end up with their packets being sent unnecessarily to the router and bounced straight back again: the router did not want the extra work.

Local routing

TCM's network is connected via our firewall directly to the CUDN. Our point of contact with the CUDN is 'route-west-3', a large router which lives in a cupboard by Reception in the Cavendish, which serves directly the rest of the Cavendish, the CL, the Vet School, the CMS, Churchill College, and several other institutions. It in turn is connected to other similar routers on the CUDN, such as the one on the Sidgwick Site, the one on the New Museums Site (to which the Computing Service, the HPCF, and the link to the rest of the World, are connected), one at Trinity, etc. These routers are all multiply connected: should one fail, its institutions suffer, but the others should all remain connected.

Routers have many mechanisms for keeping in contact with their fellow routers, and noticing if one dies or is generally overloaded, and modifying the routes they use to take account of this entirely automatically. Such dynamic load balancing can cause packets to be received out-of-order.

Not Networking

Networking used to be above getting packets to their destinations. These days it is increasingly about stopping packets one suspects of trying to cause trouble.

The Computing Service blocks certain ports at the border between Cambridge and the outside world, and a shorter list on the routers within Cambridge. Our firewall blocks little, but can rapidly be reconfigured.

Blocking based on IP port number is crude and simple. A block on port 21 will stop people accessing ftp servers, for instance, and is one of the incoming blocks that the UCS has (`ftp.tcm` is exempt from this). More sophisticated firewalls attempt to analyse traffic content, and use much more complicated rules for blocking.

Applications

The following pages demonstrate a little of the detail of some common network protocols. Please note that local (University of Cambridge) rules specifically forbid attempting to talk SMTP to mail servers “by hand” in the fashion shown here. One reason is that you are bound to get it wrong, and then waste the time of the server’s administrator whilst he looks in confusion at his logs.

finger

One of the simplest, and still common, TCP services is that provided by the finger daemon, which listens on port 79.

```
tcma:~$ finger mjr19@cus
[cus.cam.ac.uk]
Login       Name          TTY      Idle       When    Where
mjr19    M.J. Rutter      pts/9       *  Jan 25 10:33 tcma.phy.cam.ac
```

The `telnet` program is useful for connecting to remote TCP services:

```
tcma:~$ telnet cus 79
Trying 131.111.8.20...
Connected to cus.cam.ac.uk.
Escape character is '^]'.
```

at this point one can type '`mjr19`', and the response is

```
Login       Name          TTY      Idle       When    Where
mjr19    M.J. Rutter      pts/9       *  Jan 25 10:33 tcma.phy.cam.ac
```

followed by the connection being closed.

More `finger`

Those familiar with `finger` will know that it has a more verbose mode too, triggered in most clients by using ‘`-l`’. What does this do in the finger protocol? It simply prepends ‘`/w`’ to the username. So, for most clients

```
finger -l mjr19@cus
```

and `finger /wmjr19@cus`

are equivalent, and, if one wishes to use telnet and talk the fingerd protocol by hand, the ‘`/w`’ is necessary.

The WWW

HTTP is the protocol used to extract information from WWW servers. It has gone through several revisions, and everything should now use HTTP version 1.1 (2006). We will first consider version 0.9...

```
tcma:~$ telnet www.cus.cam.ac.uk 80
Trying 131.111.8.18...
Connected to all-ipv4.cus.cam.ac.uk.
Escape character is '^]'.
GET /
<head>
<title>CUS home page</title>
</head>
<body>
<p>The <a href="http://www.cam.ac.uk/cs/unix/">Central
Unix Service</a> (CUS) provides a Unix platform with a wide range
of applications software for staff and postgraduate students.</p>
</body>
</html>

Connection closed by foreign host.
tcma:~$
```

HTTP 0.9

It is that simple: the URL of the page, less the hostname, is given in a GET command (must be in capitals), and the contents of the page are sent straight back. And that is it. GET is the only possible command in HTTP 0.9.

The next version, 1.0, added the more commands, and provided much fuller status information.

HTTP 1.0

```
tcma:~$ telnet www.cus.cam.ac.uk 80
Trying 131.111.8.18...
Connected to all-ipv4.cus.cam.ac.uk.
Escape character is '^]'.
GET / HTTP/1.0
```
[This line is blank]
```
HTTP/1.1 200 OK
Date: Mon, 30 Jan 2006 16:47:04 GMT
Server: Apache/1.3.34 (Unix) mod_ucam_webauth/1.2.2
Last-Modified: Tue, 08 Feb 2005 10:17:29 GMT
ETag: "91b06b-163c-420891b9"
Accept-Ranges: bytes
Content-Length: 5692
Connection: close
Content-Type: text/html

<head>
...
```

Improvements

The response now starts with a return code (200), which enables various classes of errors and successes to be recorded. It then provides some header information, including the total length of the document, and the time it was last modified. If the command `HEAD` is used instead of `GET`, then just this header information is returned.

Notice that the headers say when the document was last modified. If the client has a previously cached version, it may choose to send a `HEAD` command to see if the document has been modified since it last downloaded it, and, if not, it can then use its previously-stored version.

Nasty people lie in the `Last-Modified` field, for it can give embarrassing evidence about how out-of-date a WWW site is. Therefore certain companies keep it current, and thereby make caching almost impossible.

Many WWW browsers refuse to display the information sent to them with every page about the server and last modified time: very dull.

Multihosting

Several DNS names may map to a single IP address. A WWW server has no idea which DNS name was used to reach it, only which IP address. Unless one speaks HTTP/1.1:

```
tcma:~$ telnet www.tcm.phy.cam.ac.uk 80
Trying 131.111.62.173...
Connected to tcmwww.phy.cam.ac.uk.
Escape character is '^]'.
HEAD / HTTP/1.1
Host: www.tcm.phy.cam.ac.uk
Connection: Close
 [This line is blank]
HTTP/1.1 200 OK
Date: Tue, 31 Jan 2006 14:21:19 GMT
Server: Apache/2.0.55 (Unix) mod_ucam_webauth/1.0.2
Last-Modified: Fri, 24 Jun 2005 09:25:38 GMT
Accept-Ranges: bytes
Connection: close
Content-Type: text/html; charset=ISO-8859-1

Connection closed by foreign host.
```

Hydras and Leaks

Repeating the request with the `Host` line replaced by `www.bio.phy.cam.ac.uk` produces a different response. The DNS returns the same IP address for both hosts, but if the server is told which name was used, it can change the page it returns.

What other information might the client give?

```
GET / HTTP/1.1
Host: tcmpc34.phy.cam.ac.uk
User-Agent: Mozilla/5.0 (X11; U; OSF1 alpha; en-US; rv:1.7.12)
 Gecko/20050930 Firefox/1.0.7
Accept: text/xml,application/xml,application/xhtml+xml,text/html;
 q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.tcm.phy.cam.ac.uk/~mjr/dummy.html
```

Errors

A WWW browser may receive one of the following errors. It should be able to distinguish between them when reporting them to the user.

- Server name not found in DNS
- Attempt to contact server gives ICMP port unreachable
- Attempt to contact server times out
- Attempt to request page gives
  - Page not found
  - Page moved (redirect)
  - Page requires authentication

Remote Disks

One protocol which cannot be driven ‘by hand’ is NFS, the Network FileSystem, used for accessing remote disks. From the viewpoint of an application, a remote and local disks are indistinguishable, it is merely that the OS will route requests for a local disk to a local disk controller, and for a remote disk to a remote computer.

Disks usually use a block size of 4KB or 8KB. This is the minimum amount of data which can be written. In order to write a smaller amount, in general a whole block must be read, and then the whole block written with the small fraction modified. As NFS attempts to commit writes immediately, the block size of the packets it receives must be a multiple of the block size of the filing system on its disk or performance will be very poor.

This is not good with ethernet’s 1500 byte maximum packet size, and it means that NFS will fragment its packets (which are usually UDP) across multiple ethernet packets. As NFS is designed to work with UDP, it has its own system for acknowledging packets. If NFS is run over TCP (which is becoming more common), packets get acknowledged at both the TCP and NFS layers: unnecessary.

Email

Sending email is, by tradition, an unauthenticated plain-text protocol. The relevant daemon listens on port 25.

SMTP, and, its successor, ESMTP (Extended Simple Mail Transport Protocol) is more chatty than HTTP: there is quite a dialogue between the client and server as the email is sent out. Every command is four characters, in capitals. Each response from the server is prefixed by a three digit number:
2xx success
3xx more data required
4xx temporary error: retry later
5xx permanent error: go away

(A similar scheme is also used by the FTP protocol.)

```
220 tcmsf1 ESMTP Exim 4.50 Tue, 31 Jan 2006 16:19:23 +0000
EHLO tcma.phy.cam.ac.uk
250-tcmsf1.phy.private.cam.ac.uk Hello tcma.phy.cam.ac.uk [131.111.62.164]
250-SIZE 52428800
250-PIPELINING
250 HELP
MAIL FROM:<mjr19@cam.ac.uk>
250 OK
RCPT TO:<mjr19@tcm.phy.cam.ac.uk>
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
From: mjr19@cam.ac.uk
To: mjr19@tcm.phy.cam.ac.uk
Subject: Salve!
Date: Tue, 31 Jan 2006 16:15:00 +0000
Message-ID: <unique1@tcma.phy.cam.ac.uk>

This is a test message.
.
250 OK id=1F3yEH-00038O-81
QUIT
```

Envelopes and Headers

Notice that the commands which cause the email to be sent out are quite distinct from the contents of the email itself. The `MAIL FROM` command and `RCPT TO` command for the envelope, and they are essentially the only part of the message which will affect its delivery. There is no reason for the email server to examine any of the message data at all. (In practice the email server may well examine the data part for signs of viruses and other malpractice, and reject the message based on what it sees there.)

The email's headers and its body are separated by a blank line.

Forging email is (mostly) trivial. One can enter any false information one wishes, and most SMTP servers will happily swallow it.

```
220 tcmsf1 ESMTP Exim 4.50 Tue, 31 Jan 2006 17:21:39 +0000
EHLO roma.gub.imp
250-tcmsf1.phy.private.cam.ac.uk Hello roma.gub.imp [131.111.62.164]
250-SIZE 52428800
250-PIPELINING
250 HELP
MAIL FROM:<spqr1@roma.gub.imp>
250 OK
RCPT TO:<mjr@tcm.phy.cam.ac.uk>
250 Accepted
DATA
354 Enter message, ending with "." on a line by itself
From: spqr1@roma.gub.imp
To: brutus@roma.gub.imp
Subject: Salve!
Date: Tue, 31 Jan 2006 16:15:00 +0000
Message-ID: <unique2@tcma.phy.cam.ac.uk>

This is a test message from mjr19
.
250 OK id=1F3zCg-0003Az-Od
QUIT
```

Lies, lies, lies

So we lied about our machine name (it is not `roma.gub.imp`), we lied about our return address (it is not `spqr1@roma.gub.imp`), we lied about our `From:` address in the data section, and we stuck a fallacious `To:` address in the data section, to which no delivery attempt was ever made. Did we get away with it? No:

```
From spqr1@roma.gub.imp Tue Jan 31 17:22:18 2006
Received: from [131.111.62.164] (port=3467 helo=roma.gub.imp)
        by tcmsf1.phy.private.cam.ac.uk with esmtp (Exim 4.50)
        id 1F3zCg-0003Az-Od
        for mjr@tcm.phy.cam.ac.uk; Tue, 31 Jan 2006 17:22:18 +0000
From: spqr1@roma.gub.imp
To: brutus@roma.gub.imp
Subject: Salve!
Date: Tue, 31 Jan 2006 16:15:00 +0000
Message-ID: <unique2@tcma.phy.cam.ac.uk>

This is a test message from mjr19
```

Postmarks

Our mail server, `tcmsf1`, like all mail servers, added a '`Received:`' line to the headers in the data section. This was rather revealing:

```
Received: from [131.111.62.164] (port=3467 helo=roma.gub.imp)
```

Although it records the lie that the email was sent from `roma.gub.imp` it also records the truth that it received it from IP address 131.111.62.164.

Of course one can easily add false `Received` headers, but one has no control over those added once the message is sent out.

And like all poor forgers, I forgot to change the message ID to something consistent.

Of course, most email programs do not bother to show the `Received` headers, despite the fact that they are the most reliable bit, and quickly show up most of those emails allegedly from Barclays accounts department as obvious forgeries.

Storing Email

Email is traditionally stored in what is sometimes called the UNIX mailbox format. This is simply a concatination of email messages, separated by blank lines. The envelope-from information is also stored immediately before each email, in a line starting '`From `' (single space, no colon). This distinguishes it from the 'from' line in the headers which will have a colon.

This store is a simple flat file. To find the boundaries of the emails, a program has to scan though looking for blank lines followed by '`From `'. To avoid a disaster should someone start a paragraph in the body of the email with the word 'From', any such thing is replaced by '>From'.

This format is inefficient for large mail folders (although easy to deliver to: one simply appends). Of course, email is *not* designed for transfering large files, or even non-ASCII files. That someone invented ways of encoding large binary files as even larger ASCII files (currently MIME/base-64) is unfortunate.

Receiving Email

SMTP transfers email, with no authentication, between mail servers. When one actually wishes to retrieve one's email, some form of authentication is necessary. The traditional method is to log on to the machine which acts as your email server, and find your own email neatly presented in a file. A slightly more modern and convenient way is to use IMAP (Internet Message Access Protocol).

In the IMAP protocol, each command from the client is prefixed with a unique identifier, which the server then uses to prefix its reply.

IMAP is designed for reading individual headers and messages from files in the above mailbox format, and for deleting individual messages. It does not allow for the manipulation of arbitrary binary files.

An IMAP Session

```
* OK [CAPABILITY IMAP4REV1 LITERAL+ AUTH=LOGIN] mercurius
IMAP4rev1 2004.357 at Sat, 18 Feb 0026 14:38:32 +0000 (GMT)
a001 LOGIN spqr1 DeusSum
a001 OK [CAPABILITY IMAP4REV1 LITERAL+ IDLE NAMESPACE
UNSELECT SCAN SORT MULTIAPPEND] User spqr1 authenticated
a002 SELECT mbox
* 2 EXISTS
* 0 RECENT
* OK [UIDVALIDITY 1140270526] UID validity status
* OK [UIDNEXT 3] Predicted next UID
* FLAGS (\Answered \Flagged \Deleted \Draft \Seen)
* OK [PERMANENTFLAGS (\* \Answered \Flagged \Deleted \Draft \Seen)]
  Permanent flags
* OK [UNSEEN 1] first unseen message in /home/spqr1/mbox
a002 OK [READ-WRITE] SELECT completed
```

a003 FETCH 1:2 (FLAGS BODY[HEADER.FIELDS (FROM SUBJECT)])
* 1 FETCH (FLAGS (\Seen) BODY[HEADER.FIELDS (FROM SUBJECT)] 51
From: Brutus <brutus@sen.roma>
Subject: Coena?

)
* 2 FETCH (FLAGS (\Seen) BODY[HEADER.FIELDS (FROM SUBJECT)] 62
From: Anna Ambasha <aa1234@aol.com>
Subject: ANNA AMBASHA

)
a004 FETCH 2 (BODY[TEXT])
* 2 FETCH (BODY[TEXT] 2430
Dignissime Domine,

Ego sum Anna Ambasha, uxor Imperatoris Ambasha, qui in seditionem interfectus est. Scio quia homo bonus et honestus es: ideo auxillium tuum rogo.

Vir meus quinque milia (MMMMM) talentium habuit. Sepulta sunt, loco

```
a004 OK FETCH completed
a005 STORE 2 +FLAGS (\Deleted)
* 2 FETCH (FLAGS (\Seen \Deleted))
a005 OK STORE completed
a006 EXPUNGE
* 2 EXPUNGE
* 1 EXISTS
* 0 RECENT
a006 OK Expunged 1 messages
a007 LOGOUT
* BYE mercurius.roma IMAP4rev1 server terminating connection
a007 OK LOGOUT completed
```

Note the plain text password (so make sure that you always use IMAPS or IMAP + TLS), and the entertainly chatty responses which the poor IMAP client must interpret.

X11

The X11 ‘system’ is very useful, and much misunderstood. It was a research project within MIT intended to investigate the feasibility of remote graphical display. Much of the project was well done.

Client applications link against an X11 library which understands the concept of separate windows, and contains functions for displaying text, lines, rectangles, etc. in windows, and handling mouse and keyboard events and focus, passing them on to the correct application.

The library does not communicate directly to the graphics hardware, but rather to an X server, either via a UNIX socket, or a TCP connection. From the programmer’s point of view, a UNIX socket and a TCP connection are almost identical. The practical difference is that a UNIX socket connects two processes running on the same machine, whereas a TCP socket can connect two processes in different continents.

The hardware-dependent code needed for a particular video card exists in the X server only: the client never has to care.

A Success?

The result worked fairly well. A client could talk to the local X server via a UNIX socket (to which normal filesystem ownerships and permissions apply), or to a remote X server, usually found listening on port 6000. Clients can not only pop up their own windows, but also grab screen-shots (e.g. `xwd`, `xv`, `gimp`), grab keyboard focus and refuse to let go (e.g. `xlock`), send fake keyboard events to other applications (few known legitimate uses, and fortunately most modern applications ignore these), and ask the X server for a copy of all keyboard events, even when the application does not have focus (few known legitimate uses, very useful for intercepting typed passwords / credit card numbers).

Because X11 (or X10, as the first version was called) was merely a demonstration of remote visualisation, no security was ever intended. Anyone could connect to your X server, take a screen-dump of it, push keystrokes into any of your applications, and record all your keystrokes. The client could be directed to any display by using the `DISPLAY` environment variable.

Patching the Holes

The first ‘solution’ was an access control list. The X server would accept connections from a given list of machines only. This is awkward if one of the machines is multi-user, or if you believe that IP spoofing is possible. The `xhost` program deals with this, including offering the possibility of typing ‘`xhost +`’ which removes all access controls. Typing ‘`xhost +`’ on a machine connected to a public network is, in general, illegal.

The second ‘solution’ was for the X server to generate a random key when it started, and to place that key in the home directory of the user who started it. The X server would accept connections only from clients which presented this key. The key was a 128 bit ‘random’ binary number.

Magic Cookies

This session key is often refered to as an MIT or X11 ‘magic cookie’. It has some sensible properties. Although it is not encrypted as it is sent to remote X servers, it becomes useless as soon as one logs out, and a new one will be generated when you next log in. Thus the traditional hacker, who leaves a network sniffer running, and returns a couple of days later to examine its logs, should find that most of the magic cookies he has intercepted are useless.

If a user wants to display from a remote computer, then everything is automatic if the machines share home directories, otherwise he needs to transfer the cookie explicitly. There are many ways of doing this: `xon -permit`, `scp`, `sftp`, even cut-and-paste with the `xauth` program.

`telnet` and `rlogin`

Two ancient ways of getting a text-only connection to a remote computer are `telnet` and `rlogin`. The telnet daemon, traditionally listening on port 23, prompts for a username and password as soon as it is contacted.

The rlogin daemon (port 513) can prompt for a username and password, but can also grant access based on a list of hostname and account name pairs. It assumes IP spoofing is impossible, so that a hostname can be trusted, and it trusts the information it receives from the remote machine about which user is running it on the remote machine.

How can it trust the remote user not to have installed a lying version of the `rlogin` program? It insists that the originating port number is less than 1024, and, in the UNIX world, only a program running with root privileges can arrange that – outgoing connections originate from port numbers $\geq$1024 otherwise. Thus the server can distinguish between a program installed by a random user, and one which the machine's administrator has installed setuid root, as the 'real' rlogin program will be.

`rsh` and friends

A simpler program than `rlogin` is `rsh`. It cannot prompt for a password, but can only use the access-list based authentication of `rlogin`. Like `rlogin`, it needs to be installed setuid root (on most UNIX systems). And of course one can never trust it from multiuser systems which do not make UNIX's distinction between 'privileged' and 'unprivileged' originating ports (e.g. Windows).

Two common programs use `rsh`: `rcp` and `xon`. In the days when trusting local networks was reasonable, it had all sorts of uses:

```
laptop$ cat mozilla.ps | rsh tcmpc1 lpr -Pps2
laptop$ rsh tcmpc1 tar -cf - project | tar -xf -
```

‘And What is your Password?’

This is a question you have been taught not to answer unless you are certain of the identity of the person asking it, and unless you are certain that no-one else is listening.

That a machine has just responded to a TCP connection that you thought you were making to `hermes.cam.ac.uk` is little evidence that it really is `hermes.cam.ac.uk`, or that nothing else is listening to the connection.

One does not simply need encryption (to make life harder for evesdroppers), but also some form of challenge to ensure that `hermes` really is `hermes`. Fortunately both are relatively easy.

Asymmetric Cyphers

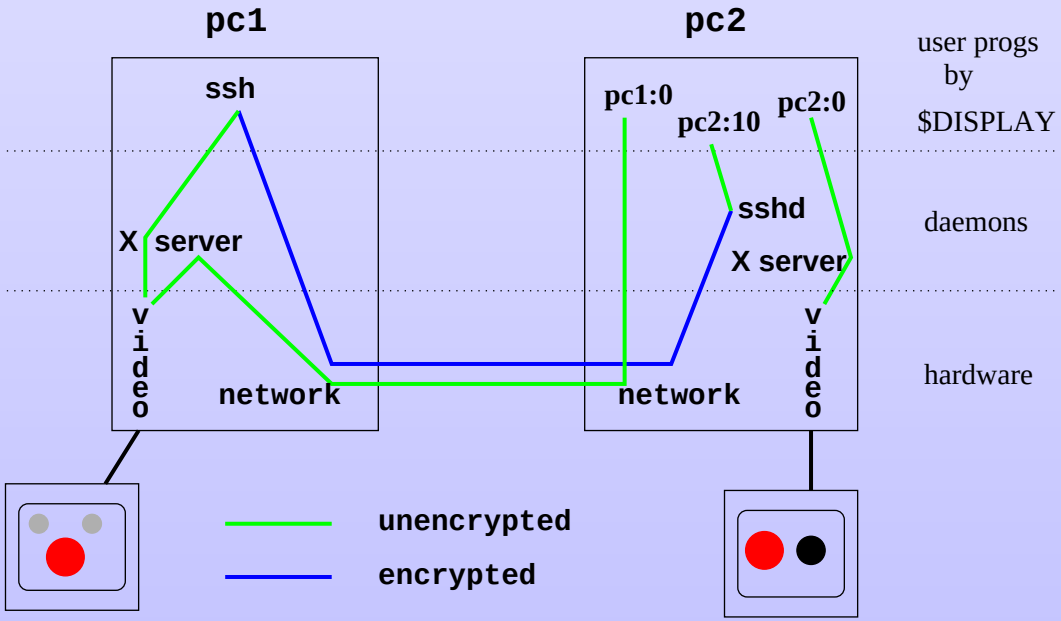
A class of cyphers exist in which the encryption and decryption keys are different, and the decryption key is (effectively) unobtainable from the encryption key. Hence one can freely distribute the encryption (or *public*) key, whilst keeping tight hold of the decryption (or *private*) key.

Identity checks are then easy. Assuming that the client already has a copy of the server's public key, the client takes a random message, encrypts it, and sends it to the server, requesting it decrypt it. No-one else possibly can, for no-one else has the decryption key.

Such asymmetric cyphers are usually computationally expensive, so next the client chooses a key for a 'cheap' symmetric cypher, encrypts it again with the server's public key, and sends it to the server. From this point, client and server can communicate using this symmetric key, which no-one else can possibly know.

The Joys of `ssh`

The `ssh` command is more than just a `telnet`/`rlogin`/`rsh` replacement. Firstly, it understands X11. On many systems, simply sshing from A to B is sufficient to enable any program run on B to have access to whatever X display the ssh program run on A could access. It does this by setting up a dummy X server on B, which sends its X traffic back to the ssh client running on A, which can then forward it to whatever X server it can contact.



More `ssh`

Although ssh does some magic which is specific to forwarding X connections, such as creating a new magic cookie for the dummy server at the remote end, and automatically creating a display name which typically ends `:10` and which is chosen so as not to clash with any other displays currently in use on that machine, the basic theory works with any TCP connection. Ssh will happily tunnel a TCP connection between its two end-points. It will even happily forward it on to a third machine unencrypted. This can be useful for getting around trivial access problems.

```
laptop$ ssh -L 12345:wwwcache.cam.ac.uk:8080 tcmpc52
tcmpc52:~$
```

and you now should be able to use localhost:12345 as your HTTP proxy on your laptop, even if you are not in Cambridge, and be treated as though you were in Cambridge. (No need for a VPN for WWW browsing.)

Yet more `ssh`

Such port forwarding tricks work with most protocols which use a single TCP connection. This includes SMTP, POP and IMAP, telnet and finger, but excludes FTP, rlogin and rsh (they use multiple TCP connections), and HTTP 1.1 (the protocol includes the name of the server, and the server will be surprised to find the name of the end of the tunnel there instead).

The port one chooses at the local end must be $\geq$1024 (unless one is running ssh as root). This means that one may have to persuade the client to use a non-standard port number in order to use the tunnel.

Of course encryption and decryption is not instantaneous, and creating long chains by sshing from A to B, then from B to C, and maybe on to D, is silly: all traffic from D to A will go via C and B being decrypted and re-encrypted at each stage, so one needs a good reason to do this.

More or Less `ssh`

Two versions of ssh are still in use. Version 1 uses an encryption algorithm which is slower, and more vulnerable, than version 2. However, even version 2 is not instantaneous. Large data transfers, or the remote display of animated graphics, across well-trusted networks, may be better done unencrypted without using ssh.

Version 2 of ssh does provide `sftp`, something which appears to humans to be identical to `ftp`, but has all the encryption advantages of `ssh`. It even has GUI front-ends for Windows (e.g. WinSCP, which is actually an sftp client).

Of course ssh and friends run natively under MacOS X, natively within the Cygwin environment on Windows, and outside of Cygwin in forms such as PuTTY (which includes an sftp client).

Other Encryption

The other common approach to encryption is to use SSL (Secure Sockets Layer). This encrypts a link in a similar fashion to ssh, but simply provides encryption, and must be combined with some other protocol to produce something useable. When combined with HTTP, it is called HTTPS, and usually found on port 443. When combined with IMAP it is called IMAPS, and usually found on port 993. It has been similarly combined with POP and even telnet.

TLS (Transport Layer Security) is approximately equivalent to version 3 of SSL. Unlike SSL, in which the encrypted server runs on one port (e.g. 993 for IMAPS), and the unencrypted server on another (e.g. 143 for IMAP), TLS is usually used by starting an unauthenticated unencrypted session to the usual port, then entering some command ('STARTTLS' in the case of IMAP) to switch to an encrypted session before attempting to send any important data, such as passwords. This avoids the need for every encrypted service to need a separate port from its unencrypted equivalent.

Unencrypted = Anonymous

In general, anonymous access is sensible to perform unencrypted: normal WWW pages, passwordless ftp sites, etc. Once more than a trivial level of authentication is required, encryption, and challenge-response identification, is almost essential.

And, of course, you should never use the same password for encrypted services as you use for the unencrypted ones.

VoIP

Sending voice traffic over IP produces new, and old, challenges. The days when telephone networks were entirely analogue have long passed, and telephone calls already get digitised at the nearest exchange, sent in digital form over packet-based networks to the recipient's exchange, then converted back to analogue form. Mobile phones do the conversion in the handset, and the signal sent is digital.

So sending voice traffic over a packet-based digital network is nothing new. It doesn't require much bandwidth either. A single channel of uncompressed CD-quality audio (44kHz, 16 bit) is 88KB/s, whereas phone quality is 8kHz, 8 bit, or 64kbits/s. Mobile phones use compression and need about 9.6kbits/s.

VoIP: TCP or UDP?

Whereas most protocols have abandoned UDP, VoIP much prefers it. TCP may offer guarantees against packet loss by being able to request retransmission, but it is not very useful to receive a lost audio packet a few seconds after one should have played it. If a packet is lost, too bad. So the lower overhead of UDP is prefered. However, the ability to spot out-of-order packets is useful, so there exists RTP (Real Time Protocol), which is layered on top of UDP.

Ideally one tags the low-bandwidth but loss-sensitive VoIP traffic so that switches can preferentially drop other traffic should they become congested. This can be done via 'IEEE 802.1p' quality of service tagging, which allows an ethernet packet to be tagged with a three-bit field to distinguish between 8 quality types.

This increases the length of the ethernet headers by four bytes, and so decreases the maximum length of the data section.

VoIP packet sizes

Decent compression reduces audio streams to around 1KB/s. Given that an ethernet packet can be up to 1500 bytes, this can be accomodated by transmitting one packet every 1.5s. This would lead to long gaps if any packet loss occurs, and long lags: one would have to speak for 1.5s before the data packet is transmitted, and would suffer a 3s round-trip lag.

If the packets are too small, then the overhead of ethernet, IP, UDP and RTP headers becomes significant (58 bytes total, 8 bytes fewer than TCP). This often happens, as, for decent quality, twenty to fifty packets per second are required, at which point the header traffic is likely to exceed the data traffic!

The flight time of the packets will vary slightly, due to other traffice on the network. An extra delay must be added by the receiver to accomodate averagely-late packets.

VoIP Standards

If one is using ethernet switches supporting ‘power over ethernet’, the 15W which they can supply is quite sufficient to power a VoIP telephone handset. Currently the most common protocol is SIP, and there are many open-source software SIP clients.

Although SIP is standardised, it still leaves room for the end-points to negotiate their own audio compression algorithm. Depending which highest common denominator they find, quality may vary. Nasty people will offer something decent and proprietory, and something common and ghastly. Then one gets decent quality when talking to the same brand of VoIP client, and poor quality otherwise. . .

Then there is Skype, which uses a closed proprietory protocol for everything.