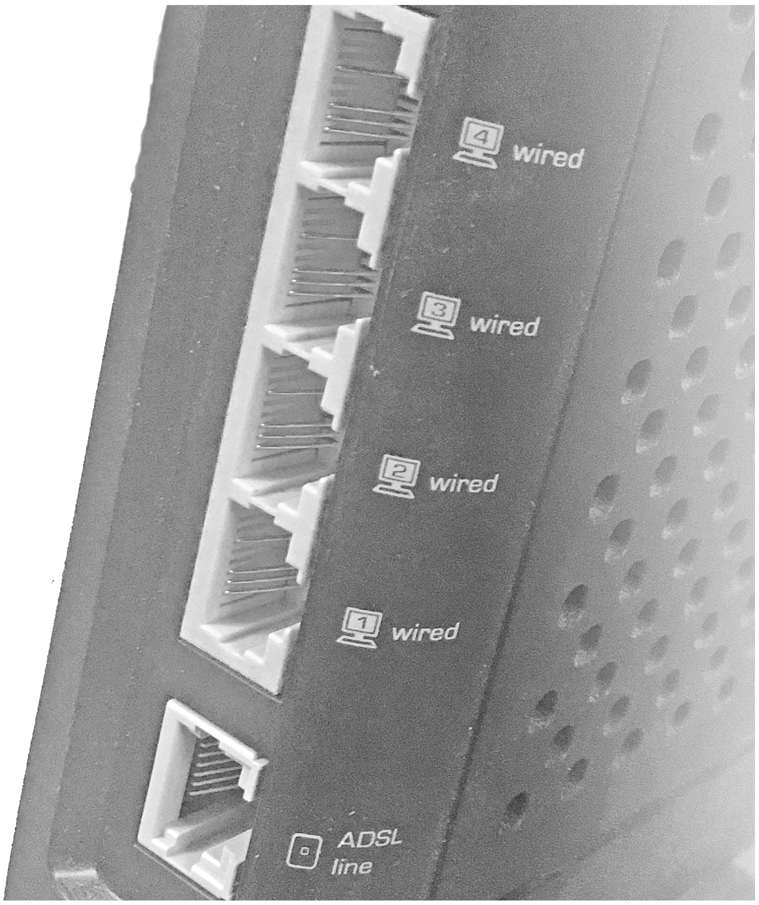
Networks: Being Connected 13



13.1 Foreword

In this chapter we will explore the concepts of **networks** and **distributed systems**, primarily from a systems and operating systems perspective. This means that we will focus on how networks and choices made in network configurations will impact upon performance, and how an operating system manages to support the ways in which networks are utilised. Having already given a brief introduction into network technology in Chapter 8, the aim here is to understand how these components are utilised in delivering services and capabilities with the aid of the operating system.

13.2 Putting modern networks in context

Networking in a modern computer system can cover a large range of possibilities. In the early days of computer networks, these were simply computer systems linked via some form of communication channel, radio, wired, or some other scheme usually operated by one company or organisation. If they were in the immediate area (perhaps the same or adjacent buildings), then this became known as a **local area network** (**LAN**). If the network covered more distant locations, it would be known as a **wide area network** (**WAN**). However, these days, networks can cover distances of a few centimetres up to the entire size of the earth[167] , and the user may not even know the difference in everyday use of their systems: the power of the operating system is that it can make a thermostat in your living room appear just as accessible as a temperature sensor at the north pole, with no discernible difference to the user. This is of course achieved with some quite sophisticated components of the operating system relating to network capabilities. We will explore some of these in this chapter.

[167] On the small scale we have smart watches talking to phones in our pockets, and on the large scale even the sky is not the limit: the International Space Station (ISS) now has an internet capability.

The concept of using a remote computer may not be new, but in the 1960's the computer might have been a mainframe in the basement of a large corporation and the users might have been located in various offices above. Today the same office users may be using computing resources that are 100's of miles away, or even on another continent.

The idea of multiple users utilising one machine in a shared fashion, such that each user appears to have their own resource, is also not new. However, whereas users in the 1960s, 70s, and 80s were likely to see this in the form of multi-user access to particular enterprise applications via terminal systems, it is possible in modern systems to have entire computer system environments delivered to each user from a server or servers, and have them sharing the same resources without realising it. This is known as **virtualisation**.

For example, a single well-specified machine might simultaneously support numerous instances of a Linux desktop environment - one physical machine - but multiple users and independent desktops. Or it might run many web-servers configured as separate virtual machines, one per customer. This allows the customer to manage their resources as if they are logged on to a real machine in an office, but this is done remotely by a suitable capability, of which we will find out a little more shortly.

As a result of all of these new developments, it can be understood that networks and distributed systems are a very diverse subject, and many technologies exist. As in earlier chapters, we will gain an overview here, rather than an expert knowledge of every aspect, so that the interested reader can then travel on towards a fuller understanding through additional study and reading.

13.3 The OSI reference model

In order to understand general principles of networks we do not need to have a full understanding of the OSI reference model. Nonetheless, this is a well established definition which defines the principles of computer networks in fine detail, and obtaining a working knowledge of its principles should be helpful.

The Open Systems Interconnection (OSI) reference model, as illustrated in Figure 13.1, defines a network as having a number of layers relating to the software and hardware required to achieve a network. The lowest layer represents the physical network, which could be as simple as a piece of wire with two voltage levels, and the highest layer represents the application that the user interacts with in order to send or receive information via that network. In between these two layers are numerous other layers, which in theory at least, have particular purposes. In practical implementations of OSI some of these layers end up being combined into single entities simply because the simplicity of that network makes this convenient to do so. In other cases, the full OSI model is implemented in a modular fashion.

One of the most important aspects of the OSI model is the **transport layer**. This ensures that rather than a single connection existing, multiple data flows can coexist. It does this by interacting with the **network layer**,

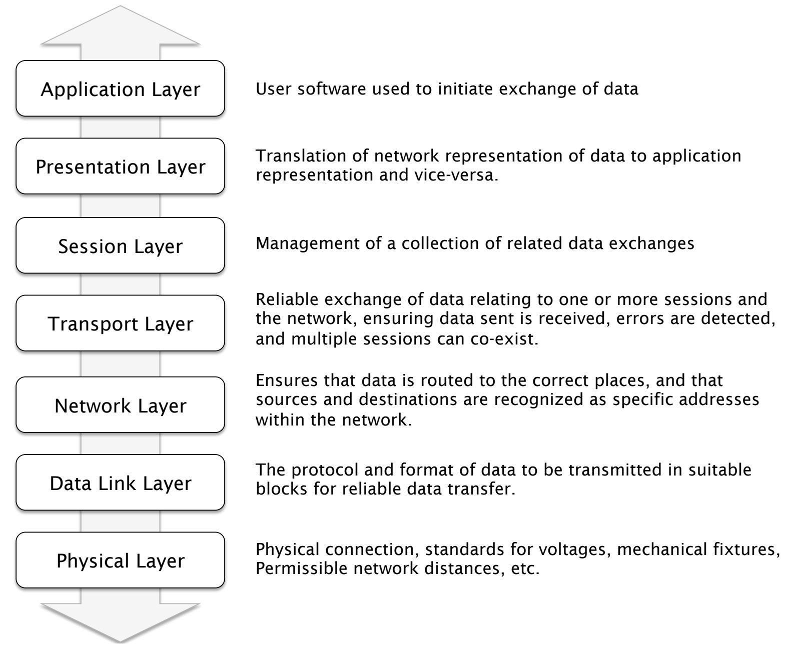

Application Layer
User software used to initiate exchange of data
Presentation Layer
Translation of network representation of data to application representation and vice-versa.
Session Layer
Management of a collection of related data exchanges
Transport Layer
Reliable exchange of data relating to one or more sessions and the network, ensuring data sent is received, errors are detected, and multiple sessions can co-exist.
Network Layer
Ensures that data is routed to the correct places, and that sources and destinations are recognized as specific addresses within the network.
Data Link Layer
The protocol and format of data to be transmitted in suitable blocks for reliable data transfer.
Physical Layer
Physical connection, standards for voltages, mechanical fixtures, Permissible network distances, etc.

Figure 13.1: OSI reference model. Each layer within the reference model has a defined purpose, ranging from the physical network infrastructure up to the application layer. Each layer provides an abstraction away from the underlying complexity of the lower layers.

where packets are sent and received, on behalf of instances of the **session layer**, which connects to the user and their particular need of use of the network. So in fact, a live network may have multiple Application-Presentation-Session layer stacks, all talking to a local transport layer that manages multiple sessions. This is of course another instance of the concept of multiplexing of a resource, which we have encountered a few times already in other contexts.

Layers below the transport layer are responsible for ensuring that data is transmitted in manageable portions, using packets and frames, ensuring data is guaranteed to arrive after being sent, detecting and dealing with errors, re-sending data after faults, and making sure physical data values are transmitted in suitable ways according to agreed standards.

Because errors are dealt with in these intermediate layers, the user and their applications will ideally never see errors during operation: the lower layers will always detect and correct these before they become visible to the user or higher service layers.[168] This allows a **reliable service provision** to be established and maintained. User applications can then rely upon this without having any knowledge of the underlying network technology or what is happening with the data being transmitted.

[168] Typical methods of error management include cyclic redundancy checks (CRC) and parity based error detection. We will discuss this in a little more depth in section 14.7.

The network layer also ensures that data exchanges between machines

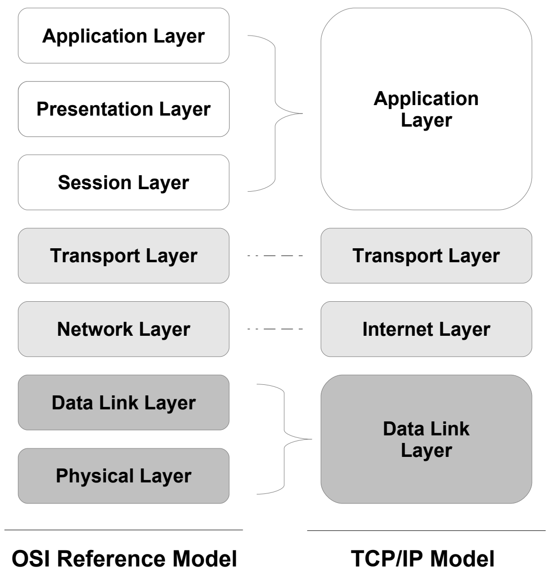


Figure 13.2: TCP/IP model The TCP/IP model has four layers, broadly equivalent to the seven layers of the OSI reference model.

are handled in the correct routing and addressing format, such that every device on the network has a unique network ID, and data is sent to the correct place to arrive at that machine.

The process of writing functional modules, relating to OSI layers themselves, requires quite a bit of knowledge of networking, and of the particular language and network functionality it provides. They have their own terminologies and details. Therefore, typically, a pre-existing network standard will be adopted, and suitable software libraries will be available for that purpose. These generally have the advantage of being well tested and regularly updated.

13.3.1 TCP/IP: Another example of layer abstraction

Whilst the OSI model defines a theoretical system of layers, each potentially distinct from each other, the practicalities of real implementations of such systems often result in some of these layers being merged. A good example of such a networking framework is the TCP/IP standard, often referred to in software terms as the TCP/IP stack. This model reduces the implementation model to four layers[169] , as shown in Figure 13.2.

[169] Although there are occasionally variants, with five layers for example.

TCP (Transmission Communication Protocol) and IP (Internet Protocol) are actually two standards operating together to form TCP/IP. The

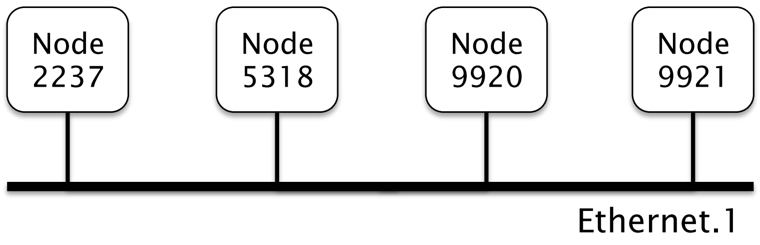


Figure 13.3: Simple network example. All nodes exist on a single network segment and share the network bandwidth.

TCP model was first demonstrated in 1974 and has evolved since then into one of the primary mediums by which modern computer systems communicate.

13.3.2 Security

Security and integrity of data transmission is another aspect that is managed within the OSI model, or equivalent layers of a networking model. We will delve a little deeper into encryption in Section 14.6. However, at this point it is worth highlighting that encryption is not a singular process. Taking the OSI model as a reference point, we can identify multiple levels of encryption at different levels of the OSI hierarchy. Applications can exchange data securely with their own encryption standards, known as **end-to-end encryption**, whilst lower levels of the protocol stack will support other forms of encryption at intermediate and lower levels, at the data-link layer for instance, even if applications do not do so.

13.4 Network structures

The simplest networks consist of nothing more than a shared wireless wifi channel used by multiple devices, or a single shared network cable, operating very much like a standard bus architecture. In these cases, all of the devices, or **network nodes**, are visible on a single physical network domain, and every node can, in theory, see every other node on that network.

An example of this simple case is given in Figure 13.3, where we see four network nodes, each with a unique identity. We will find out more about node identification later; for now we will just assume each node has a unique 4-digit number.

We can see that any node on this network can communicate with any other node on the network, simply by transmitting a data packet with the correct **destination address**. Node 2237 can communicate with node 9920 for example.

This is very simple to implement, but has drawbacks. Because the network is a shared resource, the nodes communicating with each other must compete for network bandwidth. As more and more nodes are connected to the same simple network, there is more and more competition for network bandwidth. If this continued, then eventually no single pair of nodes would be able to obtain enough network bandwidth to maintain their desired quality of service (QOS) level. In other words, this network has particular limitations on its **scalability** to large numbers of nodes.

13.4.1 Bridging the gap

A **bridge** is a network component, often described as a special component that can be used to join together two networks (or indeed two busses, as we discovered earlier). In some cases, joining two existing networks together with a bridge is exactly what is needed, and that is all good. A bridge will allow traffic to flow smoothly between two separate networks, appearing as if they are a single network.

However, we can turn this on its head and also say that a bridge allows us to split one network into two sections, or **network segments**. With this view, we can see how a bridge might well become a valuable performance tool for dividing network utilisation into manageable partitions.

Consider Figure 13.4, which shows our original network, but now split into two segments, named Ethernet.1 and Ethernet.2. Notice that nodes **2237** and **5318** are on one segment, and nodes **9920** and **9921** are on another segment.

The bridge will still allow traffic to flow from nodes on Ethernet.1 to nodes on Ethernet.2. So when node **2237** wishes to communicate with node **9920**, this will be facilitated by the bridge transferring packets between the two segments in the direction required.

However, if node **9920** wants to communicate with **9921**, the bridge will be able to recognise that those data packets do not need to cross the bridge from segment 2 to segment 1. Likewise, communications between nodes **2237** and **5318** do not require traffic to cross from segment 1 to segment 2. This process is known as filtering.

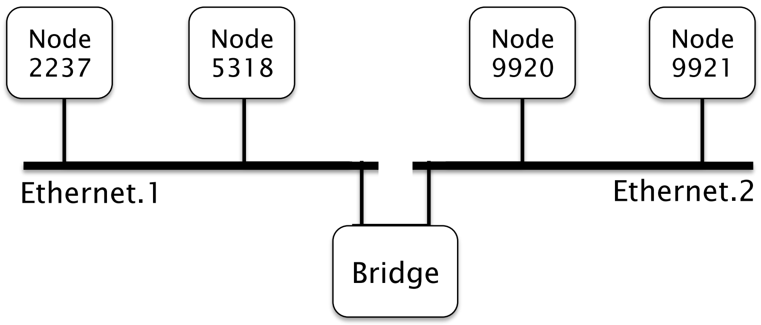


Figure 13.4: Bridged network example. The network is divided into two segments by a bridge, with each being able to be operated independently.

As a result of a bridge being able to filter network traffic, our new network now has two segments that can operate independently or jointly, depending upon the required node-to-node source-to-destination combination. When packets are not crossing the bridge, both segments can operate independently, providing twice the network bandwidth for the system as a whole. The implication is that, if we understand the traffic behaviour in a network, we can reconfigure it to maximise performance by using a technique such as bridging to create network segments that can operate concurrently.

Consider the following example:

Suppose our network has the following traffic flow during operation:

(a) 2237 to 5318: 10 Megabytes/sec
(b) 2237 to 9921: 1 Megabyte/sec
(c) 9920 to 9921: 10 Megabytes/sec

The total traffic flow on the single network given on Figure 13.3 is simply the sum of the components, 21 Megabytes/sec.

However, on the segmented network as shown in Figure 13.4, traffic flows (a) and (c) can occur concurrently, because they involve different segments, whilst only (b) involves both segments and must operate cooperatively. The total is thus 11 Megabyte/sec for each network segment. But this does not mean we have 22 Megabyte/sec of network traffic volume[170]. We have 10+10+1 = 21 Megabytes/sec total traffic volume, because we cannot count (c) twice.

[170] **Traffic volume** is how much we actually have happening on the network, as opposed to **traffic capacity**, which is how much we can accommodate.

So, we can observe that in the traffic case given in the example, the network bandwidth on each segment required to perform the described service has been almost halved. If this is a service that is being provided to end users, this means that we can deliver almost twice as much of this service using the new configuration when compared to the old one.

We can estimate network performance in a slightly different way. Suppose we have a 100 Megabyte/sec network bandwidth, which we then divide into two identical segments, and we find that 10 Megabyte/sec needs to cross the bridge. What will the total network capacity now be:

Network Capacity example

Each network segment has 100 Megabyte/sec available.
Each network segment cooperates in the 10 Megabyte/sec shared data traffic.
Bandwidth remaining on each network is therefore:
100-10 = 90 Megabyte/sec.
And then, total available traffic volume =
90+90+10 = **190 Megabyte/sec**.
This represents a network capacity improvement of 90%.

So we see that the improvement depends upon the number of segments and the amount of shared versus independent network traffic per segment.

13.4.2 Switching things around

A bridge has a minimum of two network connections (called network ports), one for each side of the bridge. However, a bridge can have more than two ports, and such **multi-port bridges** are more commonly defined as **network switches** or **hubs**, depending upon their capabilities.

A **hub** is one of the least sophisticated network components: simply receiving incoming packets on any connected port and resending them out on all of the other ports. There is no filtering, it is simply a way to connect several devices or networks together reliably so they can communicate jointly.

A **network switch** performs a very similar job to a bridge in practice, but rather than simply acting as a middle-man between two network segments, it acts as a kind of interchange between three or more segments.

A switch will still achieve the traffic filtering effect described earlier, and will allow a network to be split into multiple segments. However, the

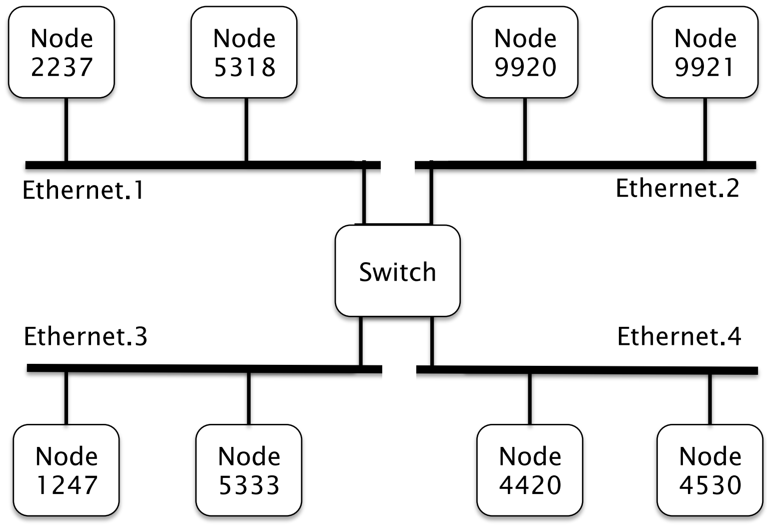


Figure 13.5: Switched network example. Where a switch allows many network segments to be linked according to the traffic flow between segments.

switch must know a little more about the network than a bridge, since it must decide which of its ports provides the segment where the destination node is accessible. It then switches those packets to the correct network segment using a mechanism involving a look-up algorithm based upon a table of destinations, or some other mechanism. An example of network with a switch is given in Figure 13.5.

Once multiple network segments are introduced, with multiple switches, it becomes possible for network segments to link to each other via more than one path. This might be to provide resilience to failures in network segments, or to increase available data transfer capacity, but it also adds to the complexity of the network switching strategy: there may be several routes that data can flow through to get from a to b, some faster than others. To manage this, more complex switching policies can be envisaged.

13.5 Routing

Routing is the process by which network hardware decides which of the multiple available network paths should be presented with network traffic from a source, in order to reach the desired destination.

In its simplest form, routing is already present to some extent in bridges and switches. In the case of a bridge, a packet appearing on one network segment can either be routed across the bridge to the the other segment, or simply ignored because it is already present on the destination segment (also known as a **subnet**). So filtering in this instance could be considered as a form of selective routing.

In the case of a switch, a packet arriving on one network port could potentially be switched to any of the other ports at that switch. The switch decides which port to send the packet to, depending upon its knowledge of which nodes and further 'downstream' switches are connected to each of the other ports and subnets.

The routing of packets can be based upon a fixed set of rules, known as a **static routing algorithm**. This has the advantage that routing decisions are fast and predictable. However, if links in the network fail, there is no fault tolerance, as the routing plan cannot adapt itself.

Another approach is **dynamically adaptive routing**, whereby information about the network is constantly passed to all switches in the network using additional packets, but at the cost of using some network bandwidth to do so. This information then allows switches and routers to make choices based on real-time network behaviour, for example, choosing between a faster route ‘a’ which is temporarily heavily loaded, versus slower route ‘b’ which has low traffic levels. Perhaps route b is the better choice at that particular moment? Dynamic routing is also potentially able to work around network faults.

The differences between a switch and a router are fairly esoteric for the every-day user. Since network switches and bridges perform some level of traffic directing, they already incorporate some level of routing within them. A dedicated router is generally designed to connect a local area network to a larger system, such as a connection to the wider internet. The methods of connectivity to the larger network environment can be via a number of options:

- Dial-up modem over analogue telephony infrastructure,
- ADSL Asymmetric digital subscriber line, an optimised version of copper-wire telephony infrastructure,
- Fibre channel connection, via optic fibre cable domestic comms infrastructure, using shared cable infrastructure.

Dedicated 'hard lines' and custom subscriber lines provide high bandwidth connections into the internet system, often private connections rather than being shared with other subscribers[171].

[171] Telecommunications providers often share the bandwidth of a physical cable across multiple customers, leasing a proportion of the bandwidth to each customer.

A good example of this might be a 'big city' trading location, where that company requires high bandwidth, uninterrupted service, and no variation in service quality. This cannot always be guaranteed if the line is shared by multiple other users with potentially conflicting needs for bandwidth throughout the day. Also, by cutting out intermediaries and extra hardware, such as switches used to share bandwidth on a common cable, the response time (latency) of the link may also be much faster, which could be critical where automated trading systems make million-dollar decisions on a timescale of seconds or fractions of seconds.

Another example might be a large organisation such as a university, where local telecomms infrastructure is not sufficient to meet the average or peak bandwidth requirements. A university could conceivably have 5,000 simultaneous users connected to the internet. Here, dedicated high bandwidth lines will be configured and leased specifically to connect to the main campus network access points.

13.6 Network node identification

The Internet is the greatest embodiment of computer technology yet realised. It is global, and almost ubiquitous. These days, even people in remote parts of relatively under-developed countries can gain access to some forms of internet connectivity via smartphones.

One can envisage a time where every individual has at least one computing device with a connection to the internet. Currently there are estimated to be in excess of 7 billion internet connected devices in operation, though the balance across the global population is not equitable. If this trend continues, we can imagine 20 or even 50 billion devices ultimately being simultaneously connected to the internet in our lifetimes.

In order for any device to be recognisable on the internet, or indeed even on a local network, it must have a unique identifier. This is often categorized as an IP address, or as a MAC address. There is an important difference to be noted here:

MAC Address (Media Access Control Address): This is a permanently assigned identifier that is preset by the manufacturer of every Network interface card and wifi controller, even those inside your laptop if you have one. In theory this code ensures that no two devices can appear to be duplicates. If this ever happens it may be because someone has deliberately forced a device to appear to be another device for some security-breaching mischief[172]. MAC addresses, in their current form, consist of six pairs of hexadecimal digits, which provides 256^6 possible unique codes[173].

[172] There may be other reasons - manufacturers not following the rules, or mistakes for example, so whilst they should be, we cannot safely say that every MAC address is unequivocally unique to one device.

[173] This amounts to around 281 trillion possibilities in total, so they are unlikely to be used up in the foreseeable future.

Internet Protocol (IP) address: This is a dynamic identifier, assigned to any device to allow it to be identified on the internet, regardless of the MAC address of the device. The IP address is assigned to a device that wishes to appear to be visible on the internet. The **IPV4** address standard currently consists of a 4-part number where each part consists of an 8-bit numerical field (256 values each). The IP address therefore supports 256^4 addresses.[174]

[174] This equates to nearly 4.3 billion addresses.

It is interesting to note that the most widely used IP standard at present is the IPV4 model, which as mentioned above supports 256^4 addresses, giving almost 4.3 billion permutations. This numbering system has already been exhausted (after all, there are at least 7 billion devices using the internet already). The more recent **IPV6** identification scheme is beginning to become the new standard at the present time, with a huge number of possible permutations, equivalent to 256^{16} possible addresses.[175]

[175] About 340 trillion trillion trillion permutations (3.4×10^{38} potential addresses).

As previously mentioned, IP addresses are assigned in a dynamic process. An assigned address may well be retained for a very long period of course, but it is not predetermined when the device is manufactured. Instead, an IP service provider, such as your cable company, will already have acquired a large block of IP addresses for its own use, and it will assign these individually to customers via their routers and server equipment.

Once a device is assigned an IP address, then in theory it can be made publicly visible on the internet. However, to do this conveniently, it is necessary to acquire a **domain name**, or a **subdomain**, and link it to the assigned IP address. When this is done, the relationship between the domain name and the IP address is stored in a globally accessible database, hosted by a machine called a **Domain Name Server** (**DNS Server**). The DNS Server allows any textual internet domain name to be translated into an IP address for subsequent direct access.

For example, if a domain name was registered by a company with the name **www.fancydomainnameexample.com** then they (as the owner of that domain) could submit an IP address to the DNS database to create a link to this domain name. The IP address can be any valid address of a device on the internet that they own or have permission to use for that purpose. This is essentially how the internet allows us to enter web addresses (though more typically as URLs[176]) into web browsers and arrive at a named page. If we actually already knew the IP address we could just type that in instead, and end up at the same place, though this is far less convenient, and offers nothing appealing in terms of domain name branding that we are now so familiar with.

[176] URL, or Uniform Resource Locator, is a textual string of information consisting of the domain name, followed by any additional locality-specific information, such as the name of a webpage.

It is important to recognise that the IP address is not a permanent identifier, though they can be allocated and remain the same for long periods. The server that provides **www.fancydomainnameexample.com** may be assigned a particular IP address, and that might stay the same for several years. However, if the company then decides to move its domain name to another service provider, with another server, then the IP address may have to change. The users of this website will have no awareness of this, and will not need to be told anything. This is because the DNS lookup data is updated to point to the new IP address at the time of the changeover. This is the beauty of using domain names rather than IP addresses.

A further point to note is that within the internal constraints of a local area network, such as a university-wide network, machines may be assigned IP addresses independently of the global internet IP addressing scheme. This is only used where the machines are to be used internally in a local area network and not individually publicly accessible via a DNS lookup. To maintain some form of standard for this kind of private IP assignment, certain IP address ranges are reserved for this purpose, and organisations setting up IP addressing within their own 'closed' networks would be expected to use one of these ranges as appropriate.

For example, the following IP address ranges are designated as such:

10.X.X.X
172.16.X.X
192.168.X.X

Here, 'X' represents any number between 0 and 255 in each digit position. The X's represent the lower portion of the network IP address range (subnet) that is available for custom allocation. So a valid address could be allocated as **172.16.**254.32 for example. Whenever a new user connects their computer to a local area network in a workplace, this process takes place dynamically via a system known as **Dynamic Host Configuration Protocol** (**DHCP**), or is manually set by an IT engineer on each machine individually.

This scheme is also used for home IP address assignments, which are managed by the cable or broadband router in your home. For example, a laptop on a home network was found to have the IP address 192.168.0.46. assigned to it on the occasion that this was checked. However, on another occasion, after leaving the network and rejoining, the IP address may be different.

Some caution is needed when typing IP addresses directly into a web browser or using them in programming scenarios. For example, an IPV4 address consists of four numbers, but each of these numbers can be typed into a browser as decimal values or octal. Surprisingly, the IP address **192.168.0.46** is not the same as IP address **192.168.0.046**, because the leading zero in the final number tells the web-browser that this number is octal 046, and this is converted to its decimal equivalent, yielding an IPV4 address **192.168.0.38**. This may be confusing to those who aren't aware of this quirk of IP notation. Therefore, one should avoid using leading zeros in IP addressing unless it is intended to use octal notation.[177]

[177] The Octal system has digits 0 to 7 (a base-8 number system), as compared to decimal (base-10), Hexadecimal (base-16), and of course Binary (base-2).

13.7 Network services

The concept of a network service is simply any capability that operates on a user's computer that facilitates some service via a network. Typically these are user applications that are initiated by the user when needed for a particular purpose, or are running on a remote server to provide some resource when a remote request is received. There are many examples of user applications in this class:

- Email systems,
- Web browsers,
- Remote file systems,
- Video and audio conferencing,
- Online gaming,
- Video streaming,
- Corporate enterprise systems.

All of these services rely upon some underlying operating system components, or additional modules that interact with the operating system. Some important and widely used components to be aware of are:

- **SSH** (**Secure Shell**) used to permit login on remote computers via the network, access to the command shell and execution of commands, and passing back responses to the initiator.
- **FTP** (**File Transfer Protocol**) used to facilitate exchange of data files between two systems on a network.
- **HTTP** (**HyperText Transfer Protocol**) used to support web-browser services, including queries sent by browsers and responses sent by web-servers.

- **SMTP** (**Simple Mail Transfer Protocol**) used to support email transmission and reception in conjunction with mail-servers.
- **POP** (**Post Office Protocol**) used to support email transmission and reception in conjunction with mail-servers.
- **CGI** (**Common Gateway Interface**) a mechanism by which servers and web-browsers can exchange data.
- **X11**: Also known as **X-windows**, an essential component for many remote graphical operating system applications. Along with equivalent systems, X11 provides the ability to view a window-based operating system and its applications remotely.

13.7.1 Sockets Interface

Many internet connection schemes now rely upon the idea of **sockets**, as originally developed in the 1980s[178]. In this model, a service on a remote machine, **the server**, has a **socket ID** assigned to it when it starts up and initiates a connection. A program on the user's computer also has a socket ID assigned when it opens a connection and attempts to link to an application or process on the remote machine. Consequently, if both entities are successful, two machines will be connected, each with an IP address identifying the specific computer, and then a socket ID relating to the specific process or thread possessing that socket. In a sense this is not unlike a pipe between two threads as we encountered earlier, but on different machines in different locations.

[178] Sockets Interface, developed in 1980's at University of California, Berkeley.

For example:

Server 144.32.128.230:**1228** (Socket ID=1228)

Client 144.146.128.200:**3007** (Socket ID=3007)

The implication here is that the client process has assigned port **3007** as a socket for the user process, and the server has assigned port **1228** as a socket for the server's service-providing process. These are unique to those two physical machines and the two specific processes running on them.[179] Remember that many programs can run on one computer and connect to many other computers doing the same thing. Therefore, it is not enough to simply have the IP addresses alone.

[179] In theory, the client and server could both exist as sockets on the same machine. That might well be useful for a diagnostic purpose for example, but not usual.

Certain socket addresses have reserved purposes (indeed, all socket ID's up to 1023 are reserved for special purposes). This means that connections can be made to machines knowing in advance what service this will be linking to (again, a bit like a named pipe in our earlier reading). For any other connection, the operating system will assign unique socket IDs between 1024 and 65535 to any processes requiring them.

13.8 Distributed systems

When an application is developed which requires network capability, it is generally built on top of one or more of the services mentioned previously (or similar alternatives). When the functionality of the application is divided between two or more systems at different locations, this is known as a **distributed system**.

A very simple example of a distributed system might operate as follows: Joe has installed a small computer in his house to control the central heating system. The computer can be accessed by a control panel in the house, but it also has an **SSH** capability and runs an operating system that supports command line commands (like an MS-DOS or Linux terminal shell).

At the simplest level, Joe can just open up a command line terminal on a PC at work, activate an SSH connection to his remote computer, and then begin typing commands at the command line as if Joe is sitting at home in front of the machine. This is a basic level of remote access.

However, to make life easier, and the access to heating control more user friendly, Joe writes a simple application program, 'Heatcon', to run on his work computer. This invokes an SSH session on the work PC to connect to his home computer but displays a **graphical user interface** (**GUI**) to Joe. The application can then transfer commands captured by the GUI to the remote computer to turn the heating on or off, or change its temperature setting, mimicking the commands that Joe would otherwise have to type in manually.

Any responses from the remote computer, which would normally be read by Joe as text in the command shell, are read by the application and processed to understand what information is being sent back. If the application is sophisticated enough it can then convert this information into part of its application display. For example, a graphical temperature gauge could be displayed, driven by a textual response from the remote computer.

Joe has, in fact, implemented a simple remote service application: a remote central heating controller. When someone uses the Heatcon application, they only see a box pop up which requests a heating temperature setting. This is the user interface (UI), or graphical user interface (GUI).

Everything else that happens, operates 'under the hood' in other words, it is hidden from the general user and is carried out automatically behind the scenes: all done via textual information exchange via an SSH connection. This means that a user does not need to know anything about SSH or command line commands at all, they can just point and click options in a GUI, and then see the responses.

After entering a temperature, and clicking 'ok', Joe's application sends the command to set the heating to temperature 'X' and then requests a report back of system status. The remote computer sends the status report back, and the Heatcon app displays this data in some useful way.

This example of a simple distributed system, as illustrated in Figure 13.6, needs two bespoke components to operate – the user interface application program on the client-side, and a utility program running on the remote computer (the server-side) that can send heat instructions to the heating system from the commands it is sent.

What we have described is a simple **client-server** network application with a local component (the Heatcon Client App) and a remote system (the server app). This is a nice simple solution. Many remotely maintained systems use this kind of approach. For example, a remote 3G cellphone tower might have computer equipment that can be accessed by an engineer remotely via SSH, allowing it to be reconfigured rapidly if a fault occurs, regardless of its location or conditions such as severe weather. To make it more efficient, an app might be developed to act as a client and allow common SSH tasks to be automated, and allowing the engineer to deal with problems more quickly.

One drawback of this model is that both the client and server software components may have to be modified and reinstalled whenever a significant change is made to the software at either end of the connection. In a system with many users, this could be a significant planning and implementation effort, and may require a period of **downtime** when the system is not able to operate.

13.9 A web-based client-server system

Whereas Joe's Heatcon application operates as a distributed client-server system, where two tangible components exist on different computers, and

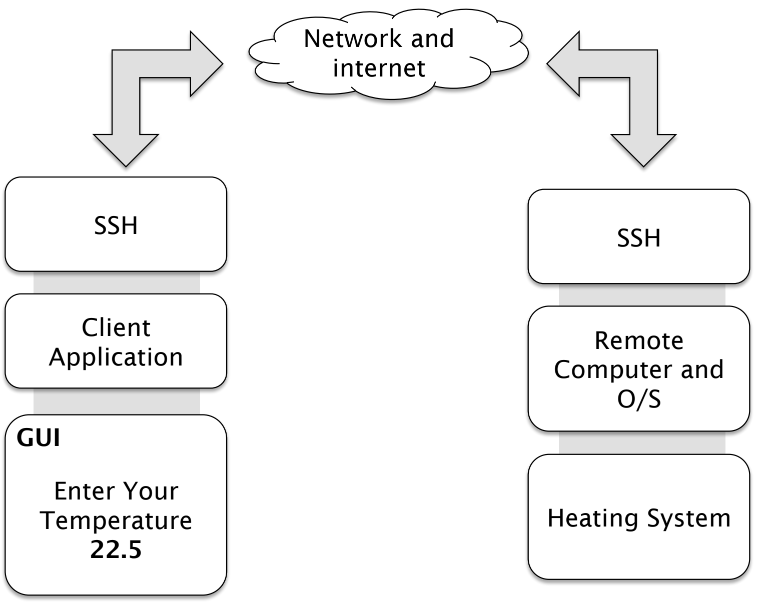


Figure 13.6: Joe's Heatcon remote central heating network application. Bespoke software is required on both the client and server machines.

cooperate to achieve the desired result, Joe could have implemented this system differently.

If the remote computer at Joe's house was set up as a web-server instead of accepting direct commands from the client application, it would be capable of receiving queries from Joe's work computer via a web-browser, and then reacting by sending back responses to be displayed and viewed as further web-pages.

At work Joe could type in a web-address and then see a web-page generated by the remote computer. It might say 'hello, the temperature at home is 20c, please select a new temperature'.

Joe can simply view this page and do nothing, or can select a new temperature via the webpage functionality and press send. The information is then sent back to the remote computer in the form of an **HTTP request**, and the remote machine decodes this message, performs the relevant tasks locally, and then sends a new web-page as a response to say it has completed the request. The new page might simply be a refreshed and updated version of the same page, so that it simply appears to update in response to the user action.

What is interesting about this system, as shown in Figure 13.7 is that there is no custom client component. Any web-browser can act as the

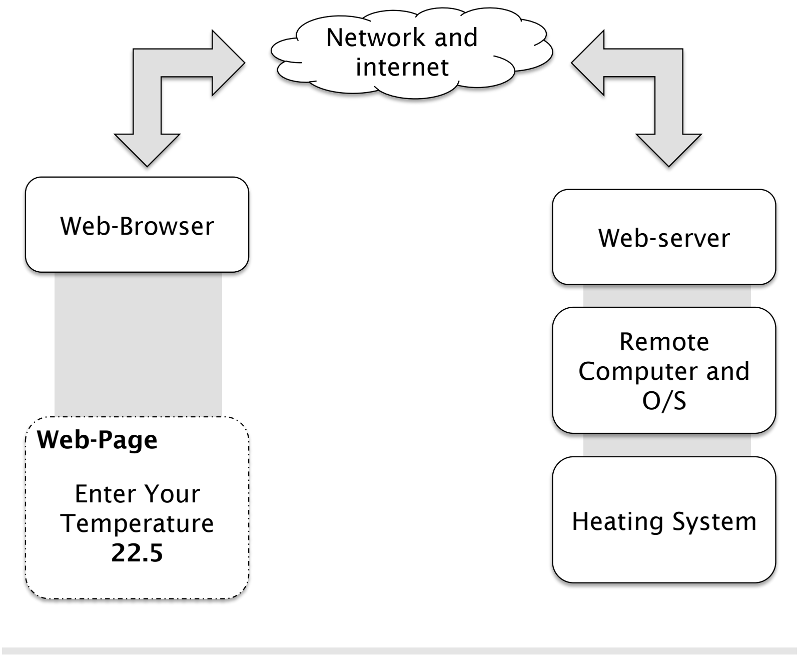


Figure 13.7: Joe's client-server Heatweb application. Only the server requires bespoke software. The client operates entirely via a standard web browser.

access point to communicate with the server, but all of the actual work is done on the remote computer: it serves up the web-pages, interprets the http requests, performs the local commands, and generates new pages as responses.

A consequence of adopting this model is that any maintenance can be done entirely on the server side, and that program version can be changed over in a matter of seconds to an updated version[180], meaning virtually no interruption of service is observed.

[180] Though it may not seem obvious, it is also (importantly) possible to just as quickly revert back to a previous server program version. If a problem arose after updating, then the system can quickly be **'rolled back'** to a known good previous state.

In the distributed system example, Heatcon, as shown in earlier Figure 13.6, the user application had some role in deciding how to respond to user choices (which commands to send etc), but with this new Heatweb system the client (the web-browser) has no knowledge of the remote system.

There are many programming languages specifically designed to deal with client-server scenarios. On the client-side, which is effectively just a web-page, the server could provide a web-page that contains Javascript[181] for example. This would allow the web-page to have more interactive features and be more user friendly. But note that the GUI aspect that this provides is still being generated by the server: the client needs no prior knowledge or locally installed code to operate, provided that it has

[181] Javascript is effectively a way of sending program code to a web-browser from a server to allow it to perform something more complex than point and click browsing.

javascript enabled in its browser settings.

Meanwhile, on the server side, a language such as PERL[182] could manage HTTP requests and generate appropriate responses, whilst being able to initiate running of additional programs such as the utility Joe created to allow the home computer to send commands to the heating controller.

[182] PERL is one of a number of languages that are flexible and convenient in providing CGI (Common Gateway Interface) program modules to manage web-server requests and responses. Such CGI-scripts form the backbone of many web-server systems.

13.10 Cloud computing

Cloud computing is a term used to refer to distributed computing systems, and in particular it relates to the use of remote computing resources by a user or users, to achieve a particular computational outcome. This may take several forms:

- **Cloud storage:** Primarily a data storage and retrieval provision, used to provide reliable storage capability and/or large scale data storage, without the need for local equipment and maintenance.
- **Cloud computing:** Primarily a computational service provision, used to provide always-available scalable computing resources at low cost, particularly where a customer has a highly variable workload demand model.
- **Hybrid:** a combination of storage and compute services used together according to specific need.

Understanding the concept of cloud computing has much to do with the economics and logistics of deploying computer systems and services, and the nature of computer system end-user demand.

Consider, for example, a company that has a small number of staff but has a large data storage requirement, which is likely to grow over time. Let us suppose this is a medical image analysis and archiving service provider. What are the options for this company to provide for its resource requirements?

13.10.1 Traditional in-house infrastructure

One option available would be to have a server installed in their building (the in-house infrastructure option). However, there are drawbacks for this approach:

- **Cost**: The initial cost of the server might be quite high, skewing the costs of the business toward resource investment rather than day to day running costs.
- **Staff:** The company must now have a member of staff responsible for maintaining the server, upgrading the hard disks from time to time, to increase storage capacity. This may mean discarding perfectly good resources because they are upgraded.
- **Data backups:** The company must make regular data backups, and must also have a contingency to ensure that data is always available even if the server fails. Should they have two servers just in case?
- **Data resilience:** Perhaps it is not enough to simply backup the data? If the data is important enough, it must be backed up onto storage media and stored in a second location to guard against a fire at the main premises for example.
- **Data security:** Both the local server and the remote backups may need high security. Server security requires specialist IT skills: more staff costs, and remote backup storage may require a secure storage location and incur costs to lease.
- **Location:** Perhaps the company is one of the new breed of companies where staff are scattered around the country, working from home or at multiple geographical locations. A single server doesn't make much sense without a main office.

If a local infrastructure is preferred, then these are the issues that must be considered. However, there are other options.

13.10.2 The cloud computing option

Another option is to lease remote computer services. (the cloud computing option). Rather than buying and installing a server in the head office, the company leases cloud computing resources. How does this help?

- **Cost**: The initial cost of the server is zero, the company simply rents services (server time and capacity). It may be slightly more expensive day to day, but up-front costs are much lower.
- **Staff:** The company does not need any significant IT staff: the maintenance is all done remotely by the cloud service provider, and is factored into the cost.
- **Data backups:** Regular backups are standard for many cloud computing platforms, and specific backup options can be subscribed to when needed. Major cloud service providers will have distributed

backup capabilities, and that data may be backed up to multiple locations, perhaps not even in the same country. Loss of data is then almost impossible.

- **Data security:** No system is entirely secure, but a reputable cloud service provider will have security arrangements at least as good as any private company can manage via their own IT staff, and typically much better.
- **Resilience:** A cloud server could fail, but there are always second, third, and fourth machines waiting to take over, meaning system failures will be rarely noticed and have a short-lived impact.
- **Location:** The cloud platform can be anywhere, and so can the employees of the company accessing the resources: there is no physical location constraint to consider.
- **Upgrades:** The company only pays for what it uses. If its storage capacity increases, it will pay more, but it will not have to pay repeatedly for old hard disks to be scrapped and upgraded.

So there are many benefits from using a cloud-based infrastructure, much of them surrounding the consistency and reliability of the service provision, but also in terms of the cost, or the spreading of that cost over a time period.

13.11 Compute servers and virtual machines

The example described in previous sections was based upon the idea that a company needed reliable access to securely stored and backed up images in large volumes. There the main requirement was storage and retrieval.

Suppose that the company now analyses its working practices (its workflow model) and it is observed that each employee's main working activity (repeated many times per day) consists of:

(a) loading an image from the server,
(b) applying image processing on a local computer,
(c) saving the new image and results back to the server.

Clearly, the storage and retrieval is an important aspect of this work, and is catered for by the cloud computing data storage and retrieval capability. However, it is also noted that the analysis techniques are becoming gradually more sophisticated, and the image sizes are growing larger as medical scanners acquire higher imaging resolutions. As a result, the computers used by the staff, and the network they are connected to, are increasingly becoming a bottleneck: they are becoming too slow to handle the new requirements.

An option here would be to scrap all of the computers and buy new, more powerful ones. But perhaps this will simply mean the problem recurs in a year or two. What should the company do? Should they accept the cost of regular machine upgrades? Or buy very expensive high performance machines to stay ahead of the game for longer periods? Again this suggests all sorts of costs, and maintenance implications. And of course those computers will sit idle for 16 hours a day, when the offices are closed.

Again, cloud computing can provide a different kind of answer. Rather than attempting to perform more and more computation on local machines, the company can instead maintain a set of modestly specified computers for its staff, on their desks. Then, when an image processing task is to be performed, the staff member can run that image analysis task on a remote server (sometimes referred to as a **compute server**). In a sense, the staff desktop computers have become modern 'dumb terminals', like the VT100 terminal we mentioned earlier. There are some valuable advantages of doing this. For example, the remote server can make available many processors for a short period when needed. Instead of one CPU taking 5 minutes, 5 CPUs can do the work in one minute. The working efficiency of the staff is improved instantly as a result.

If the company recognised that cloud computing and cloud storage were both good solutions to their requirements, and even better when working together, they may even be able to organise an integrated solution with a service provider, such that the images are processed remotely without ever being transferred to the staff member, and only the results are viewed. This would result in a lower data traffic requirement: speeding things up further and potentially saving money once again on broadband connection bandwidth costs.

What we have explored in this case is an imaginary company with particular requirements, and the potential to optimise their costs and infrastructure, and reduce risks, by using cloud computing. We can imagine many cases where this could be applied, and of course a few where it may not be the best option.

For more on this topic, see CASE-STUDY **??**.

13.12 Virtualisation

In the previous section we explored the idea of a server as a machine that can provide resources to multiple users apparently simultaneously, either as a local resource or a service delivered via the cloud infrastructure route. An important question here is how the user is provided with this service. If 10 employees all use the same server at the same time, how does that work?

In early computer systems, the multi-user paradigm was often managed by having time-slicing between users. Each user would get frequently switched portions of CPU and IO time. Where the operating system and the use of resources was very limited, this could be done with relatively low task-switching overheads. Another approach was for users to submit jobs to a queue and they would then be processed in some order of priority. However, that system does not allow for interactive use of the computer resource (whereas user time-slicing does).

In a modern computer system things are a little different. It is still possible for a server to be set up to accept jobs sent to a queue: for example, image processing requests generated by staff members in the company scenario we discussed. A computer server could be configured to do this quite successfully if there is frequent demand for repetitive processes, and the users are happy to wait for a response for a certain period.

On the other hand, a user may want to have an interactive desktop, running some particular operating system and applications, and an extended level of control over what it is doing. Likewise, cloud computing providers do not want to have a separate physical machine set up in their premises for every customer, especially as their use might be intermittent. In these scenarios, the idea of **virtualisation** becomes quite important as a solution.

Virtualisation is a concept whereby an operating system and user applications can run on an emulation of a real machine. Because a computer can potentially run several instances of these virtual machines (VMs), then one well-specified computer system (i.e. a server) can emulate a number of virtual machines at the same time. A server might support 8 virtual machines for example. There is of course a limit, based upon the performance characteristics of that server. With a particular amount of physical memory, and a particular processor technology, a particular set of disk units, there will be a limit to the number of virtual machines a server can support without noticeably poorer performance for the individual users of those virtual machines.

A virtual machine therefore looks just like a normal operating system including the windowed desktop if required. Here, there are two options:

- A user can connect to a virtual machine via SSH to connect to the command line of that machine and run suitable commands.
- A user can connect to the virtual machine via X-Windows (or an equivalent), which duplicates the desktop of that machine on the screen of the user's local machine.

In the case of SSH, the remote virtual machine looks exactly like a local command line computing resource. This might well be all that is needed for running long, complex computational jobs (simply run a script and wait), and this could be the basis of a job-queue. Equally, if the virtual machine is set up to run as a web-server, making web-pages available to internet users, then SSH command line access may be all that is needed for an administrator to alter file content and web-page configurations.

In the case of X11 X-windows and similar, the user will instead see a desktop window on their local machine, containing a live image of the entire desktop of the remote virtual machine. It will look and behave exactly like a normal desktop, though there may be speed and lag issues in the behaviour of the system if the local network and internet connectivity is not suitably high in performance. One particular advantage of this approach is that the user can access a much more powerful machine to occasionally run specialised applications, where a dedicated local machine of similar capability would be just too expensive to justify.

There are several ways in which virtualisation can be supported. Figure 13.8 shows some examples of these virtual machine concepts. Each concept relies upon an intermediary software agent called a hypervisor[183] , which coordinates the activities needed to share resources, emulate the virtual machines, etc.

[183] Not to be confused with hyperthreading, which is a technique that allows multiple threads to co-execute on a processor core simultaneously: See Section 10.8.3.

Native Virtualisation: This allows a machine with a specific natively executed **hypervisor** component to provide multiple virtual machines, each appearing to be duplicates of the underlying system, but sharing the machine resources as if each user is the sole user of the machine. The VMs can each have different customisations such as user privileges and installed options etc, but are effectively clones of the same base system.

Hosted Virtualisation: This uses a model whereby a native operating system has a hypervisor installed as an application, and this can be any third-party virtualisation software. This hosted hypervisor can then, in theory, support any virtual machine model, so a MICROSOFT Windows

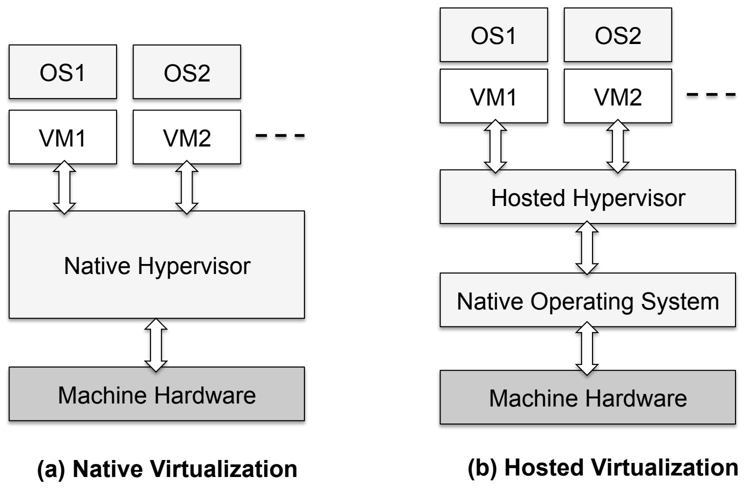


Figure 13.8: Typical virtualisation models. Showing (a) Native virtualisation, and (b) Hosted Virtualisation, where a virtualisation system is hosted as a layer on top of another operating system.

native operating system could host a hypervisor running a Linux virtual machine for example, or vice versa. Indeed, multiple virtual machines can coexist if the hypervisor supports this, and each VM can be identical or entirely different from the others. One of the consequences of this is that a server can be set up to support multiple users (different customers accessing a cloud resource for example) and provide a different resource to each user: different versions of the same operating system, or entirely different operating systems.

Local Virtualisation: Virtualisation can also be used locally, and not just to access resources on remote servers. For example, an APPLE Mac user may need to use Linux OS or Windows OS from time to time, and could do so by using a virtual machine hypervisor tool to allow these operating systems to appear to run on that platform.

This is frequently used where software engineers have to maintain software for multiple platforms, or where there is some legacy requirement to maintain software or to access a particular resource that can only be run under a given operating system. This is far more efficient than maintaining a physical machine configured with that setup just for those occasional cases of use, and removes the need to keep rebooting a machine with a different boot option each time in order to choose the particular setup required.

13.13 Summary

Networking is a topic that is often presented with a high degree of technical knowledge. However, the basic principles are fairly straightforward. Understanding how these relate to hardware certainly helps, however.

In this course material we have sought to explore the idea of a network as a hardware infrastructure, as well as a set of software capabilities and services. Hopefully, with the cases we have highlighted, and the performance issues we have explored, the capabilities and limitations of networked systems will be a little clearer.

13.14 Terminology introduced in this chapter

Client-server	DCHP
Destination node	Distributed system
Domain Name	Domain Name Server
Dynamically adaptive routing	Graphical User Interface
Hosted Virtualisation	HTTP Request
IP Address	LAN
Local area network	MAC address
Multi-port bridge	Native Virtualisation
Network hub	Network layer (OSI)
Network node	Network routing
Network scalability	Network segment
Network subnet	Network switch
Reliable service provision	Session layer
Socket Interface	Static routing algorithm
Subdomain	Transport layer (OSI)
WAN	Wide area network

These terms are defined in the glossary, Appendix A.1.