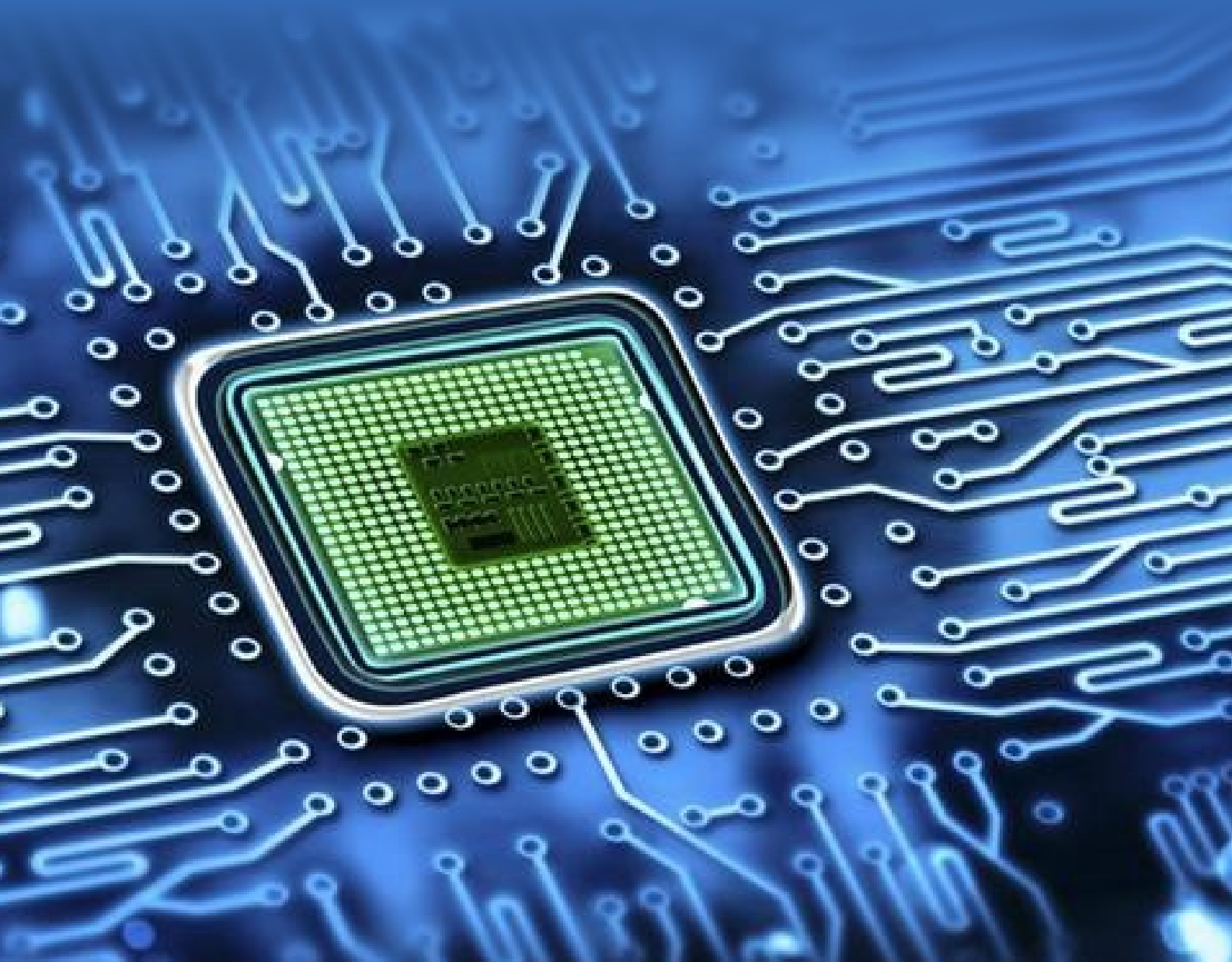
Computer System Architecture

Roland Barr



COMPUTER SYSTEM ARCHITECTURE

COMPUTER SYSTEM ARCHITECTURE

Roland Barr

BIBLIOTEX
Digital Library

Computer System Architecture
by Roland Barr

www.bibliotex.com

Ebook ISBN: 9781984664327

Published by:
Bibliotex
Canada
Website: www.bibliotex.com

Contents

1

Internet Architecture

INTRODUCTION

What is the *Internet* architecture? It is by definition a meta-network, a constantly changing collection of thousands of individual networks intercommunicating with a common protocol.

The Internet's architecture is described in its name, a short from of the compound word "inter-networking". This architecture is based in the very specification of the standard *TCP/IP* protocol, designed to connect any two networks which may be very different in internal hardware, software, and technical design. Once two networks are interconnected, communication with TCP/IP is enabled end-to-end, so that any node on the Internet has the near magical ability to communicate with any other no matter where they are. This openness of design has enabled the Internet architecture to

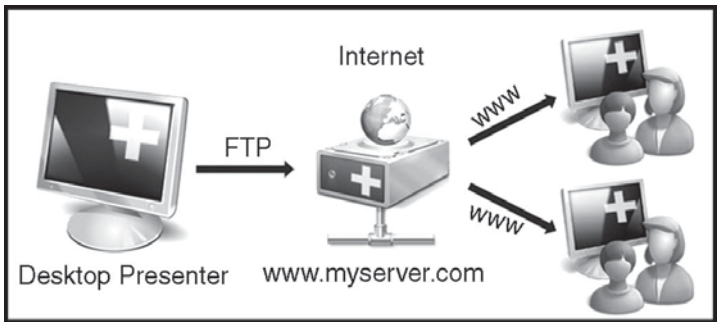
grow to a global scale. In practice, the Internet technical architecture looks a bit like a multi-dimensional river system, with small tributaries feeding medium-sized streams feeding large rivers. For example, an individual's access to the Internet is often from home over a modem to a local Internet service provider who connects to a regional network connected to a national network. At the office, a desktop computer might be connected to a local area network with a company connection to a corporate Intranet connected to several national Internet service providers.

In general, small local Internet service providers connect to medium-sized regional networks which connect to large national networks, which then connect to very large bandwidth networks on the Internet *backbone*. Most Internet service providers have several redundant network cross-connections to other providers in order to ensure continuous availability. The companies running the Internet backbone operate very high bandwidth networks relied on by governments, corporations, large organizations, and other Internet service providers. Their technical infrastructure often includes global connections through underwater cables and satellite links to enable communication between countries and continents. As always, a larger scale introduces new phenomena: the number of packets flowing through the switches on the backbone is so large that it exhibits the kind of complex non-linear patterns usually found in natural, analog systems like the flow of water or development of the rings of Saturn (RFC 3439, S2.2).

Each communication *packet* goes up the hierarchy of Internet networks as far as necessary to get to its destination

network where local *routing* takes over to deliver it to the addressee. In the same way, each level in the hierarchy pays the next level for the bandwidth they use, and then the large backbone companies settle up with each other. Bandwidth is priced by large Internet service providers by several methods, such as at a fixed rate for constant availability of a certain number of megabits per second, or by a variety of use methods that amount to a cost per gigabyte. Due to economies of scale and efficiencies in management, bandwidth cost drops dramatically at the higher levels of the architecture.

DEFINITION OF INTERNET



The Internet is a global network of computers. Every computer that is connected to the Internet is considered a part of that network. This means even your home computer. It's all a matter of degrees, you connect to your ISP's network, then your ISP connects to a larger network and so on. At the top of the tree is the high-capacity backbones, all of these interconnect at 'Network Access Points' 'NAPs' at important regions around the world. The entire Internet is based on agreements between these backbone providers who set in place all the fibre optics lines and other technical aspects of

the Internet. The first high speed backbone was created by the 'National Science Foundation' in 1987.

The Internet was first created by the Advanced Research Projects Agency (ARPA) of the U.S. government in 1960's, and was first known as the ARPANet. At this stage the Internet's first computers were at academic and government institutions. They were mainly used for accessing files and to send email. From 1983 onwards the Internet as we know it today started to form with the introduction of the communication protocol TCP/IP to ARPANet.

Since 1983 the Internet has accommodated alot of changes and continues to keep developing. The last two decades has seen the Internet accommodate such things as network LANs and ATM and frame switched services. The Internet continues to evolve with it becoming available on mobile phones and pagers and possibly on televisions in the future. The actual term "Internet" was finally defined in 1995 by FNC (The Federal Networking Council). The resolution created by the The Federal Networking Council (FNC) agrees that the following language reflects our definition of the term "Internet". "Internet" refers to the global information system that,

THE EVOLUTION OF THE INTERNET

The underpinnings of the Internet are formed by the global interconnection of hundreds of thousands of otherwise independent computers, communications entities and information systems. What makes this interconnection possible is the use of a set of communication standards, procedures and formats in common among the networks and

the various devices and computational facilities connected to them. The procedures by which computers communicate with each other are called "protocols." While this infrastructure is steadily evolving to include new capabilities, the protocols initially used by the Internet are called the "TCP/IP" protocols, named after the two protocols that formed the principal basis for Internet operation.

On top of this infrastructure is an emerging set of architectural concepts and data structures for heterogeneous information systems that renders the Internet a truly global information system.

In essence, the Internet is an architecture, although many people confuse it with its implementation. When the Internet is looked at as an architecture, it manifests two different abstractions. One abstraction deals with communications connectivity, packet delivery and a variety of end-end communication services. The other abstraction deals with the Internet as an information system, independent of its underlying communications infrastructure, which allows creation, storage and access to a wide range of information resources, including digital objects and related services at various levels of abstraction.

Interconnecting computers is an inherently digital problem. Computers process and exchange digital information, meaning that they use a discrete mathematical "binary" or "two-valued" language of 1s and 0s. For communication purposes, such information is mapped into continuous electrical or optical waveforms.

The use of digital signaling allows accurate regeneration and reliable recovery of the underlying bits. We use the terms

"computer," "computer resources" and "computation" to mean not only traditional computers, but also devices that can be controlled digitally over a network, information resources such as mobile programs and other computational capabilities.

The telephone network started out with operators who manually connected telephones to each other through "patch panels" that accepted patch cords from each telephone line and electrically connected them to one another through the panel, which operated, in effect, like a switch. The result was called circuit switching, since at its conclusion, an electrical circuit was made between the calling telephone and the called telephone. Conventional circuit switching, which was developed to handle telephone calls, is inappropriate for connecting computers because it makes limited use of the telecommunication facilities and takes too long to set up connections. Although reliable enough for voice communication, the circuit-switched voice network had difficulty delivering digital information without errors.

For digital communications, packet switching is a better choice, because it is far better suited to the typically "burst" communication style of computers. Computers that communicate typically send out brief but intense bursts of data, then remain silent for a while before sending out the next burst. These bursts are communicated as packets, which are very much like electronic postcards.

The postcards, in reality packets, are relayed from computer to computer until they reach their destination. The special computers that perform this forwarding function are called variously "packet switches" or "routers" and form the

equivalent of many bucket brigades spanning continents and oceans, moving buckets of electronic postcards from one computer to another. Together these routers and the communication links between them form the underpinnings of the Internet.

Without packet switching, the Internet would not exist, as we now know it. Going back to the postcard analogy, postcards can get lost. They can be delivered out of order, and they can be delayed by varying amounts. The same is true of Internet packets, which, on the Internet, can even be duplicated. The Internet Protocol is the postcard layer of the Internet. The next higher layer of protocol, TCP, takes care of re-sending the "postcards" to recover packets that might have been lost, and putting packets back in order if they have become disordered in transit.

Of course, packet switching is about a billion times faster than the postal service or a bucket brigade would be. It also has to operate over many different communications systems, or substrata. The authors designed the basic architecture to be so simple and undemanding that it could work with most communication services. Many organizations, including commercial ones, carried out research using the TCP/IP protocols in the 1970s. Email was steadily used over the nascent Internet during that time and to the present. It was not until 1994 that the general public began to be aware of the Internet by way of the World Wide Web application, particularly after Netscape Communications was formed and released its browser and associated server software.

Thus, the evolution of the Internet was based on two technologies and a research dream. The technologies were

packet switching and computer technology, which, in turn, drew upon the underlying technologies of digital communications and semiconductors. The research dream was to share information and computational resources. But that is simply the technical side of the story. Equally important in many ways were the other dimensions that enabled the Internet to come into existence and flourish. This aspect of the story starts with cooperation and far-sightedness in the U.S. Government, which is often derided for lack of foresight but is a real hero in this story.

It leads on to the enthusiasm of private sector interests to build upon the government funded developments to expand the Internet and make it available to the general public. Perhaps most important, it is fueled by the development of the personal computer industry and significant changes in the telecommunications industry in the 1980s, not the least of which was the decision to open the long distance market to competition.

The role of workstations, the Unix operating system and local area networking (especially the Ethernet) are themes contributing to the spread of Internet technology in the 1980s into the research and academic community from which the Internet industry eventually emerged.

Many individuals have been involved in the development and evolution of the Internet covering a span of almost four decades if one goes back to the early writings on the subject of computer networking by Kleinrock, Licklider, Baran, Roberts, and Davies. The ARPANET, described below, was the first wide-area computer network. The NSFNET, which followed more than a decade later under the leadership of

Erich Bloch, Gordon Bell, Bill Wulf and Steve Wolff, brought computer networking into the mainstream of the research and education communities. It is not our intent here to attempt to attribute credit to all those whose contributions were central to this story, although we mention a few of the key players

COMPUTER NETWORK HIERARCHY

Every computer that is connected to the Internet is part of a network, even the one in your home. For example, you may use a modem and dial a local number to connect to an Internet Service Provider (ISP). At work, you may be part of a local area network (LAN), but you most likely still connect to the Internet using an ISP that your company has contracted with. When you connect to your ISP, you become part of their network. The ISP may then connect to a larger network and become part of their network. The Internet is simply a network of networks. Most large communications companies have their own dedicated backbones connecting various regions. In each region, the company has a Point of Presence (POP). The POP is a place for local users to access the company's network, often through a local phone number or dedicated line. The amazing thing here is that there is no overall controlling network. Instead, there are several high-level networks connecting to each other through Network Access Points or NAPs.

INTERNET NETWORK EXAMPLE

Here's an example. Imagine that Company A is a large ISP. In each major city, Company A has a POP. The POP in each

city is a rack full of modems that the ISP's customers dial into. Company A leases fibre optic lines from the phone company to connect the POPs together.

Imagine that Company B is a corporate ISP. Company B builds large buildings in major cities and corporations locate their Internet server machines in these buildings. Company B is such a large company that it runs its own fibre optic lines between its buildings so that they are all interconnected.

In this arrangement, all of Company A's customers can talk to each other, and all of Company B's customers can talk to each other, but there is no way for Company A's customers and Company B's customers to intercommunicate. Therefore, Company A and Company B both agree to connect to NAPs in various cities, and traffic between the two companies flows between the networks at the NAPs.

In the real Internet, dozens of large Internet providers interconnect at NAPs in various cities, and trillions of bytes of data flow between the individual networks at these points. The Internet is a collection of huge corporate networks that agree to all intercommunicate with each other at the NAPs. In this way, every computer on the Internet connects to every other.

THE FUNCTION OF AN INTERNET ROUTER

All of these networks rely on NAPs, backbones and routers to talk to each other. What is incredible about this process is that a message can leave one computer and travel halfway across the world through several different networks and arrive at another computer in a fraction of a second!

The routers determine where to send information from one computer to another.

Routers are specialized computers that send your messages and those of every other Internet user speeding to their destinations along thousands of pathways. A router has two separate, but related, jobs:

- It ensures that information doesn't go where it's not needed. This is crucial for keeping large volumes of data from clogging the connections of "innocent bystanders."
- It makes sure that information does make it to the intended destination.

In performing these two jobs, a router is extremely useful in dealing with two separate computer networks. It joins the two networks, passing information from one to the other. It also protects the networks from one another, preventing the traffic on one from unnecessarily spilling over to the other. Regardless of how many networks are attached, the basic operation and function of the router remains the same. Since the Internet is one huge network made up of tens of thousands of smaller networks, its use of routers is an absolute necessity.

INTERNET BACKBONE

The National Science Foundation (NSF) created the first high-speed backbone in 1987. Called NSFNET, it was a T1 line that connected 170 smaller networks together and operated at 1.544 Mbps (million bits per second). IBM, MCI and Merit worked with NSF to create the backbone and developed a T3 (45 Mbps) backbone the following year.

Backbones are typically fibre optic trunk lines. The trunk line has multiple fibre optic cables combined together to increase the capacity. Fibre optic cables are designated OC for optical carrier, such as OC-3, OC-12 or OC-48. An OC-3 line is capable of transmitting 155 Mbps while an OC-48 can transmit 2,488 Mbps (2.488 Gbps). Compare that to a typical 56K modem transmitting 56,000 bps and you see just how fast a modern backbone is.

Today there are many companies that operate their own high-capacity backbones, and all of them interconnect at various NAPs around the world. In this way, everyone on the Internet, no matter where they are and what company they use, is able to talk to everyone else on the planet. The entire Internet is a gigantic, sprawling agreement between companies to intercommunicate freely.

INTERNET PROTOCOL IP ADDRESSES

Every machine on the Internet has a unique identifying number, called an IP Address. The IP stands for Internet Protocol, which is the language that computers use to communicate over the Internet. A protocol is the pre-defined way that someone who wants to use a service talks with that service. The "someone" could be a person, but more often it is a computer program like a Web browser.

A typical IP address looks like this:

To make it easier for us humans to remember, IP addresses are normally expressed in decimal format as a *dotted decimal number* like the one above. But computers communicate in binary form. Look at the same IP address in binary:

The four numbers in an IP address are called octets, because they each have eight positions when viewed in binary form. If you add all the positions together, you get 32, which is why IP addresses are considered 32-bit numbers. Since each of the eight positions can have two different states (1 or zero), the total number of possible combinations per octet is 2^8 or 256. So each octet can contain any value between zero and 255. Combine the four octets and you get 2^{32} or a possible 4,294,967,296 unique values!

Out of the almost 4.3 billion possible combinations, certain values are restricted from use as typical IP addresses. For example, the IP address 0.0.0.0 is reserved for the default network and the address 255.255.255.255 is used for broadcasts.

The octets serve a purpose other than simply separating the numbers. They are used to create classes of IP addresses that can be assigned to a particular business, government or other entity based on size and need. The octets are split into two sections: Net and Host. The Net section always contains the first octet. It is used to identify the network that a computer belongs to. Host (sometimes referred to as Node) identifies the actual computer on the network. The Host section always contains the last octet. There are five IP classes plus certain special addresses.

DOMAIN NAME SYSTEM

When the Internet was in its infancy, it consisted of a small number of computers hooked together with modems and telephone lines. You could only make connections by providing the IP address of the computer you wanted to

establish a link with. For example, a typical IP address might be 216.27.22.162. This was fine when there were only a few hosts out there, but it became unwieldy as more and more systems came online.

The first solution to the problem was a simple text file maintained by the Network Information Centre that mapped names to IP addresses. Soon this text file became so large it was too cumbersome to manage. In 1983, the University of Wisconsin created the Domain Name System (DNS), which maps text names to IP addresses automatically.

URL: UNIFORM RESOURCE LOCATOR

When you use the Web or send an e-mail message, you use a domain name to do it. For example, the Uniform Resource Locator (URL) "http://www.yahoo.com" contains the domain name howstuffworks.com. So does this e-mail address: example@yahoo.com. Every time you use a domain name, you use the Internet's DNS servers to translate the human-readable domain name into the machine-readable IP address. Top-level domain names, also called first-level domain names, include.COM.ORG.NET.EDU and.GOV. Within every top-level domain there is a huge list of second-level domains.

Every name in the.COM top-level domain must be unique. The left-most word, like www, is the host name. It specifies the name of a specific machine (with a specific IP address) in a domain. A given domain can, potentially, contain millions of host names as long as they are all unique within that domain. DNS servers accept requests from programs and other name servers to convert domain names into IP addresses.

When a request comes in, the DNS server can do one of four things with it:

- It can answer the request with an IP address because it already knows the IP address for the requested domain.
- It can contact another DNS server and try to find the IP address for the name requested. It may have to do this multiple times.
- It can say, "I don't know the IP address for the domain you requested, but here's the IP address for a DNS server that knows more than I do."
- It can return an error message because the requested domain name is invalid or does not exist.

A DNS EXAMPLE

Let's say that you type the URL www.yahoo.com into your browser. The browser contacts a DNS server to get the IP address. A DNS server would start its search for an IP address by contacting one of the root DNS servers. The root servers know the IP addresses for all of the DNS servers that handle the top-level domains (.COM.NET.ORG, etc.). Your DNS server would ask the root for www.howstuffworks.com, and the root would say, "I don't know the IP address for www.howstuffworks.com, but here's the IP address for the.COM DNS server." Your name server then sends a query to the.COM DNS server asking it if it knows the IP address for www.howstuffworks.com. The DNS server for the COM domain knows the IP addresses for the name servers handling the www.yahoo.com domain, so it returns those.

Your name server then contacts the DNS server for www.yahoo.com and asks if it knows the IP address for www.yahoo.com. It actually does, so it returns the IP address to your DNS server, which returns it to the browser, which can then contact the server for www.yahoo.com to get a Web page.

One of the keys to making this work is redundancy. There are multiple DNS servers at every level, so that if one fails, there are others to handle the requests. The other key is caching. Once a DNS server resolves a request, it caches the IP address it receives.

Once it has made a request to a root DNS server for any.COM domain, it knows the IP address for a DNS server handling the.COM domain, so it doesn't have to bug the root DNS servers again for that information. DNS servers can do this for every request, and this caching helps to keep things from bogging down.

Even though it is totally invisible, DNS servers handle billions of requests every day and they are essential to the Internet's smooth functioning. The fact that this distributed database works so well and so invisibly day in and day out is a testimony to the design.

INTERNET SERVERS AND CLIENTS

Internet servers make the Internet possible. All of the machines on the Internet are either servers or clients. The machines that provide services to other machines are servers. And the machines that are used to connect to those services are clients. There are Web servers, e-mail servers, FTP servers and so on serving the needs of Internet users all over the

world. When you connect to www.yahoo.com to read a page, you are a user sitting at a client's machine. You are accessing the yahoo Web server. The server machine finds the page you requested and sends it to you. Clients that come to a server machine do so with a specific intent, so clients direct their requests to a specific software server running on the server machine. For example, if you are running a Web browser on your machine, it will want to talk to the Web server on the server machine, not the e-mail server.

A server has a static IP address that does not change very often. A home machine that is dialing up through a modem, on the other hand, typically has an IP address assigned by the ISP every time you dial in. That IP address is unique for your session — it may be different the next time you dial in. This way, an ISP only needs one IP address for each modem it supports, rather than one for each customer.

PORTS AND HTTP

Any server machine makes its services available using numbered ports — one for each service that is available on the server. For example, if a server machine is running a Web server and a file transfer protocol (FTP) server, the Web server would typically be available on port 80, and the FTP server would be available on port 21. Clients connect to a service at a specific IP address and on a specific port number.

Once a client has connected to a service on a particular port, it accesses the service using a specific protocol. Protocols are often text and simply describe how the client and server will have their conversation. Every Web server on the Internet conforms to the hypertext transfer protocol (HTTP). Networks, routers, NAPs, ISPs, DNS and powerful servers all make the

Internet possible. It is truly amazing when you realise that all this information is sent around the world in a matter of milliseconds! The components are extremely important in modern life— without them, there would be no Internet. And without the Internet, life would be very different indeed for many of us.

INTERNET AND WORLD WIDE WEB

The Internet and the World Wide Web have a whole-to-part relationship. The Internet is the large container, and the Web is a part within the container. It is common in daily conversation to abbreviate them as the "Net" and the "Web", and then swap the words interchangeably. But to be technically precise, the Net is the restaurant, and the Web is the most popular dish on the menu.

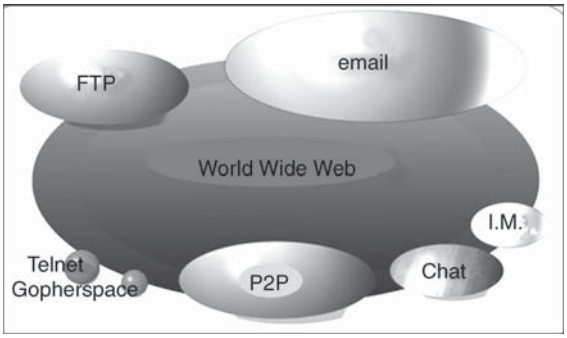
Here is the detailed explanation:

The Internet is a Big Collection of Computers and Cables. The Internet is named for "interconnection of computer networks". It is a massive hardware combination of millions of personal, business, and governmental computers, all connected like roads and highways. The Internet started in the 1960's under the original name "ARPAnet". ARPAnet was originally an experiment in how the US military could maintain communications in case of a possible nuclear strike. With time, ARPAnet became a civilian experiment, connecting university mainframe computers for academic purposes.

As personal computers became more mainstream in the 1980's and 1990's, the Internet grew exponentially as more users plugged their computers into the massive network. Today, the Internet has grown into a public spiderweb of

millions of personal, government, and commercial computers, all connected by cables and by wireless signals.

No single person owns the Internet. No single government has authority over its operations. Some technical rules and hardware/software standards enforce how people plug into the Internet, but for the most part, the Internet is a free and open broadcast medium of hardware networking.



THE WEB IS A BIG COLLECTION OF HTML PAGES ON THE INTERNET

The World Wide Web, or "Web" for short, is that large software subset of the Internet dedicated to broadcasting HTML pages. The Web is viewed by using free software called web browsers. Born in 1989, the Web is based on hypertext transfer protocol, the language which allows you and me to "jump" (hyperlink) to any other public web page. There are over 40 billion public web pages on the Web today.

The Internet is a worldwide network of computers that use common communication standards and interfaces to provide the physical backbone for a number of interesting applications. One of the most utilized of these Internet applications is the World Wide Web. What sets the Web apart is an easy-to-use interface to a complex network of computers and data.

2

Evolution of Network Operating System

Modern network devices are complex entities composed of both silicon and software. Thus, designing an efficient hardware platform is not, by itself, sufficient to achieve an effective, cost-efficient and operationally tenable product.

The control plane plays a critical role in the development of features and in ensuring device usability.

Although progress from the development of faster CPU boards and forwarding planes is visible, structural changes made in software are usually hidden, and while vendor collateral often offers a list of features in a carrier-class package, operational experiences may vary considerably. Products that have been through several generations of software releases provide the best examples of the difference made by the choice of OS. It is still not uncommon to find routers or switches that started life under older, monolithic

software and later migrated to more contemporary designs. The positive effect on stability and operational efficiency is easy to notice and appreciate.

However, migration from one network operating system to another can pose challenges from non-overlapping feature sets, noncontiguous operational experiences and inconsistent software quality. These potential challenges make it is very desirable to build a control plane that can power the hardware products and features supported in both current and future markets.

Developing a flexible, long-lasting and high-quality network OS provides a foundation that can gracefully evolve to support new needs in its height for up and down scaling, width for adoption across many platforms, and depth for rich integration of new features and functions. It takes time, significant investment and in-depth expertise.

Most of the engineers writing the early releases of Junos OS came from other companies where they had previously built network software. They had firsthand knowledge of what worked well, and what could be improved. These engineers found new ways to solve the limitations that they'd experienced in building the older operating systems.

Resulting innovations in Junos OS are significant and rooted in its earliest design stages. Still, to ensure that our products anticipate and fulfil the next generation of market requirements, Junos OS is periodically reevaluated to determine whether any changes are needed to ensure that it continues to provide the reliability, performance and resilience for which it is known. Contemporary network operating systems are mostly advanced and specialized

branches of POSIX-compliant software platforms and are rarely developed from scratch. The main reason for this situation is the high cost of developing a world-class operating system all the way from concept to finished product. By adopting a general purpose OS architecture, network vendors can focus on routing-specific code, decrease time to market, and benefit from years of technology and research that went into the design of the original (donor) products.

FIRST-GENERATION OS: MONOLITHIC ARCHITECTURE

Typically, first-generation network operating systems for routers and switches were proprietary images running in a flat memory space, often directly from flash memory or ROM. While supporting multiple processes for protocols, packet handling and management, they operated using a cooperative, multitasking model in which each process would run to completion or until it voluntarily relinquished the CPU.

All first-generation network operating systems shared one trait: They eliminated the risks of running full-size commercial operating systems on embedded hardware. Memory management, protection and context switching were either rudimentary or nonexistent, with the primary goals being a small footprint and speed of operation.

Nevertheless, first-generation network operating systems made networking commercially viable and were deployed on a wide range of products. The downside was that these systems were plagued with a host of problems associated with resource management and fault isolation; a single runaway process could easily consume the processor or cause the entire system to fail. Such failures were not uncommon

in the data networks controlled by older software and could be triggered by software errors, rogue traffic and operator errors.

Legacy platforms of the first generation are still seen in networks worldwide, although they are gradually being pushed into the lowest end of the telecom product lines.

SECOND-GENERATION OS: CONTROL PLANE MODULARITY

The mid-1990s were marked by a significant increase in the use of data networks worldwide, which quickly challenged the capacity of existing networks and routers. By this time, it had become evident that embedded platforms could run full-size commercial operating systems, at least on high-end hardware, but with one catch: They could not sustain packet forwarding with satisfactory data rates. A breakthrough solution was needed. It came in the concept of a hard separation between the control and forwarding plane—an approach that became widely accepted after the success of the industry's first application-specific integrated circuit (ASIC)-driven routing platform, the Juniper Networks M40. Forwarding packets entirely in silicon was proven to be viable, clearing the path for next generation network operating systems, led by Juniper with its Junos OS.

Today, the original M40 routers are mostly retired, but their legacy lives in many similar designs, and their blueprints are widely recognized in the industry as the second-generation reference architecture.

Second-generation network operating systems are free from packet switching and thus are focused on control plane functions. Unlike its first-generation counterparts, a second-

generation OS can fully use the potential of multitasking, multithreading, memory management and context manipulation, all making systemwide failures less common. Most core and edge routers installed in the past few years are running second-generation operating systems, and it is these systems that are currently responsible for moving the bulk of traffic on the Internet and in corporate networks. However, the lack of a software data plane in second-generation operating systems prevents them from powering low-end devices without a separate (hardware) forwarding plane. Also, some customers cannot migrate from their older software easily because of compatibility issues and legacy features still in use.

These restrictions led to the rise of transitional (generation 1.5) OS designs, in which a first-generation monolithic image would run as a process on top of the second-generation scheduler and kernel, thus bridging legacy features with newer software concepts. The idea behind "generation 1.5" was to introduce some headroom and gradually move the functionality into the new code, while retaining feature parity with the original code base. Although interesting engineering exercises, such designs were not as feature-rich as their predecessors, nor as effective as their successors, making them of questionable value in the long term.

THIRD-GENERATION OS: FLEXIBILITY, SCALABILITY AND CONTINUOUS OPERATION

Although second-generation designs were very successful, the past 10 years have brought new challenges.

Increased competition led to the need to lower operating expenses and a coherent case for network software flexible

enough to be redeployed in network devices across the larger part of the end-to-end packet path. From multiple terabit routers to Layer 2 switches and security appliances, the "best-in-class" catchphrase can no longer justify a splintered operational experience—true "network" operating systems are clearly needed. Such systems must also achieve continuous operation, so that software failures in the routing code, as well as system upgrades, do not affect the state of the network. Meeting this challenge requires availability and convergence characteristics that go far beyond the hardware redundancy available in second-generation routers.

Another key goal of third-generation operating systems is the capability to run with zero downtime (planned and unplanned). Drawing on the lesson learned from previous designs regarding the difficulty of moving from one OS to another, third-generation operating systems also should make the migration path completely transparent to customers. They must offer an evolutionary, rather than revolutionary upgrade experience typical to the retirement process of legacy software designs.

BASIC OS DESIGN CONSIDERATIONS

Choosing the right foundation (prototype) for an operating system is very important, as it has significant implications for the overall software design process and final product quality and serviceability. This importance is why OEM vendors sometimes migrate from one prototype platform to another midway through the development process, seeking a better fit. Generally, the most common transitions are from a proprietary to a commercial code base and from a commercial code base to an open-source software foundation.

Regardless of the initial choice, as networking vendors develop their own code, they get further and further away from the original port, not only in protocol-specific applications but also in the system area. Extensions such as control plane redundancy, in-service software upgrades and multi chassis operation require significant changes on all levels of the original design. However, it is highly desirable to continue borrowing content from the donor OS in areas that are not normally the primary focus of networking vendors, such as improvements in memory management, scheduling, multi core and symmetric multiprocessing (SMP) support, and host hardware drivers. With proper engineering discipline in place, the more active and peer-reviewed the donor OS is, the more quickly related network products can benefit from new code and technology.

This relationship generally explains another market trend—only two out of five network operating systems that emerged in the routing markets over the past 10 years used a commercial OS as a foundation.

Juniper's main operating system, Junos OS, is an excellent illustration of this industry trend. The basis of the Junos OS kernel comes from the FreeBSD UNIX OS, an open-source software system. The Junos OS kernel and infrastructure have since been heavily modified to accommodate advanced and unique features such as state replication, nonstop active routing and in-service software upgrades, all of which do not exist in the donor operating system. Nevertheless, the Junos OS tree can still be synchronized with the FreeBSD repository to pick the latest in system code, device drivers and development tool chains, which allows Juniper Networks engineers to concentrate on network-specific development.

COMMERCIAL VERSUS OPEN-SOURCE DONOR OS

The advantage of a more active and popular donor OS is not limited to just minor improvements—the cutting edge of technology creates new dimensions of product flexibility and usability. Not being locked into a single-vendor framework and roadmap enables greater control of product evolution as well as the potential to gain from progress made by independent developers.

This benefit is evident in Junos OS, which became a first commercial product to offer hard resource separation of the control plane and a real-time software data plane. Juniper-specific extension of the original BSD system architecture relies on multicore CPUs and makes Junos OS the only operating system that powers both low-end software-only systems and high-end multiple-terabit hardware platforms with images built from the same code tree. This technology and experience could not be created without support from the entire Internet-driven community. The powerful collaboration between leading individuals, universities and commercial organizations helps Junos OS stay on the very edge of operating system development. Further, this collaboration works both ways:

Juniper donates to the free software movement, one example being the Juniper Networks FreeBSD/MIPS port.

FUNCTIONAL SEPARATION AND PROCESS SCHEDULING

Multiprocessing, functional separation and scheduling are fundamental for almost any software design, including network software. Because CPU and memory are shared

resources, all running threads and processes have to access them in a serial and controlled fashion. Many design choices are available to achieve this goal, but the two most important are the memory model and the scheduling discipline.

MEMORY MODEL

The memory model defines whether processes (threads) run in a common memory space. If they do, the overhead for switching the threads is minimal, and the code in different threads can share data via direct memory pointers.

The downside is that a runaway process can cause damage in memory that does not belong to it.

In a more complex memory model, threads can run in their own virtual machines, and the operating system switches the context every time the next thread needs to run. Because of this context switching, direct communication between threads is no longer possible and requires special Inter Process Communication (IPC) structures such as pipes, files and shared memory pools.

SCHEDULING DISCIPLINE

Scheduling choices are primarily between cooperative and preemptive models, which define whether thread switching happens voluntarily. A cooperative multitasking model allows the thread to run to completion, and a preemptive design ensures that every thread gets access to the CPU regardless of the state of other threads.

VIRTUAL MEMORY/PREEMPTIVE SCHEDULING PROGRAMMING MODEL

Virtual memory with preemptive scheduling is a great design choice for properly constructed functional blocks,

where interaction between different modules is limited and well defined. This technique is one of the main benefits of the second-generation OS designs and underpins the stability and robustness of contemporary network operating systems. However, it has its own drawbacks.

Notwithstanding the overhead associated with context switching, consider the interaction between two threads, A and B, both relying on the common resource R. Because threads do not detect their relative scheduling in the preemptive model, they can actually access R in a different order and with varying intensity. For example, R can be accessed by A, then B, then A, then A and then B again. If thread B modifies resource R, thread A may get different results at different times—and without any predictability. For instance, if R is an interior gateway protocol (IGP) next hop, B is an IGP process, and A is a BGP process, then BGP route installation may fail because the underlying next hop was modified midway through routing table modification. This scenario would never happen in the cooperative multitasking model, because the IGP process would release the CPU only after it finishes the next-hop maintenance. This problem is well researched and understood within software design theory, and special solutions such as resource locks and synchronization primitives are easily available in nearly every operating system. However, the effectiveness of IPC depends greatly on the number of interactions between different processes. As the number of interacting processes increases, so does the number of IPC operations. In a carefully designed system, the number of IPC operations is proportional to the number of processes (N). In a system with extensive IPC

activity, this number can be proportional to N2. Exponential growth of an IPC map is a negative trend not only because of the associated overhead, but because of the increasing number of unexpected process interactions that may escape the attention of software engineers.

In practice, overgrown IPC maps result in systemwide "IPC meltdowns" when major events trigger intensive interactions. For instance, pulling a line card would normally affect interface management, IGP, exterior gateway protocol and traffic engineering processes, among others. When interprocess interactions are not well contained, this event may result in locks and tight loops, with multiple threads waiting on each other and vital system operations such as routing table maintenance and IGP computations temporarily suspended. Such defects are signatures of improper modularization, where similar or heavily interacting functional parts do not run as one process or one thread. The right question to ask is, "Can a system be too modular?" The conventional wisdom says, "Yes." Excessive modularity can bring long-term problems, with code complexity, mutual locks and unnecessary process interdependencies. Although none of these may be severe enough to halt development, feature velocity and scaling parameters can be affected. Complex process interactions make programming for such a network OS an increasingly difficult task.

On the other hand, the cooperative multitasking, shared memory paradigm becomes clearly suboptimal if unrelated processes are influencing each other via the shared memory pool and collective restartability. A classic problem of first-generation operating systems was systemwide failure due to

a minor bug in a nonvital process such as SNMP or network statistics. Should such an error occur in a protected and independently restartable section of system code, the defect could easily be contained within its respective code section.

This brings us to an important conclusion. No fixed principle in software design fits all possible situations. Ideally, code design should follow the most efficient paradigm and apply different strategies in different parts of the network OS to achieve the best marriage of architecture and function. This approach is evident in Junos OS, where functional separation is maintained so that cooperative multitasking and preemptive scheduling can both be used effectively, depending on the degree of IPC containment between functional modules.

GENERIC KERNEL DESIGN

Kernels normally do not provide any immediately perceived or revenue-generating functionality. Instead, they perform housekeeping activities such as memory allocation and hardware management and other system-level tasks. Kernel threads are likely the most often run tasks in the entire system. Consequently, they have to be robust and run with minimal impact on other processes.

In the past, kernel architecture largely defined the operating structure of the entire system with respect to memory management and process scheduling. Hence, kernels were considered important differentiators among competing designs.

Historically, the disputes between the proponents and opponents of lightweight versus complex kernel architectures

came to a practical end when most operating systems became functionally decoupled from their respective kernels.

Once software distributions became available with alternate kernel configurations, researchers and commercial developers were free to experiment with different designs.

For example, the original Carnegie-Mellon Mach microkernel was originally intended to be a drop-in replacement for the kernel in BSD UNIX and was later used in various operating systems, including mkLinux and GNU FSF projects. Similarly, some software projects that started life as purely microkernel-based systems later adopted portions of monolithic designs.

Over time, the radical approach of having a small kernel and moving system functions into the user-space processes did not prevail. A key reason for this was the overhead associated with extra context switches between frequently executed system tasks running in separate memory spaces.

Furthermore, the benefits associated with restart ability of essentially all system processes proved to be of limited value, especially in embedded systems. With the system code being very well tested and limited to scheduling, memory management and a handful of device drivers, the potential errors in kernel subsystems are more likely to be related to hardware failures than to software bugs.

This means, for example, that simply restarting a faulty disk driver is unlikely to help the routing engine stay up and running, as the problem with storage is likely related to a hardware failure (for example, uncorrectable fault in a mass storage device or system memory bank).

Another interesting point is that although both monolithic and lightweight kernels were widely studied by almost all

operating system vendors, few have settled on purist implementations. For example, Apple's Mac OS X was originally based on microkernel architecture, but now runs system processes, drivers and the operating environment in BSD-like subsystems. Microsoft NT and derivative operating systems also went through multiple changes, moving critical performance components such as graphical and I/O subsystems in and out of the system kernel to find the right balance of stability, performance and predictability. These changes make NT a hybrid operating system. On the other hand, freeware development communities such as FSF, FreeBSD and NetBSD have mostly adopted monolithic designs (for example, Linux kernel) and have gradually introduced modularity into selected kernel sections (for example, device drivers).

So what difference does kernel architecture make to routing and control?

MONOLITHIC VERSUS MICROKERNEL NETWORK OPERATING SYSTEM DESIGNS

In the network world, both monolithic and microkernel designs can be used with success.

However, the ever-growing requirements for a system kernel quickly turn any classic implementation into a compromise. Most notably, the capability to support a real-time forwarding plane along with stateful and stateless forwarding models and extensive state replication requires a mix of features not available from any existing monolithic or microkernel OS implementation.

This lack can be overcome in two ways.

First, a network OS can be constrained to a limited class of products by design. For instance, if the OS is not intended for mid- to low-level routing platforms, some requirements can be lifted. The same can be done for flow-based forwarding devices, such as security appliances. This artificial restriction allows the network operating systems to stay closer to their general-purpose siblings—at the cost of fracturing the product lineup. Different network element classes will now have to maintain their own operating systems, along with unique code bases and protocol stacks, which may negatively affect code maturity and customer experience.

Second, the network OS can evolve into a specialized design that combines the architecture and advantages of multiple classic implementations.

This custom kernel architecture is a more ambitious development goal because the network OS gets further away from the donor OS, but the end result can offer the benefits of feature consistency, code maturity, and operating experience. This is the design path that Juniper selected for Junos OS.

JUNOS OS KERNEL

According to the formal criteria, the Junos OS kernel is fully customizable. At the very top is a portion of code that can be considered a microkernel. It is responsible for real-time packet operations and memory management, as well as interrupts and CPU resources. One level below it is a more conventional kernel that contains a scheduler, memory manager and device drivers in a package that looks more like a monolithic design.

Finally, there are user-level (POSIX) processes that actually serve the kernel and implement functions normally residing inside the kernels of classic monolithic router operating systems. Some of these processes can be compound or run on external CPUs (or packet forwarding engines). In Junos OS, examples include periodic hello management, kernel state replication, and protected system domains (PSDs).

The entire structure is strictly hierarchical, with no underlying layers dependent on the operations of the top layers.

This high degree of virtualization allows the Junos OS kernel to be both fast and flexible.

However, even the most advanced kernel structure is not a revenue-generating asset of the network element.

Uptime is the only measurable metric of system stability and quality. This is why the fundamental difference between the Junos OS kernel and competing designs lies in the focus on reliability.

Coupled with Juniper's industry-leading nonstop active routing and system upgrade implementation, kernel state replication acts as the cornerstone for continuous operation. In fact, the Junos OS redundancy scheme is designed to protect data plane stability and routing protocol adjacencies at the same time. With in-service software upgrade, networks powered by Junos OS are becoming immune to the downtime related to the introduction of new features or bug fixes, enabling them to approach true continuous operation. Continuous operation demands that the integrity of the control and forwarding planes remains intact in the event of failover or system upgrades, including minor and major

release changes. Devices running Junos OS will not miss or delay any routing updates when either a failure or a planned upgrade event occurs.

This goal of continuous operation under all circumstances and during maintenance tasks is ambitious, and it reflects Juniper's innovation and network expertise, which is unique among network vendors.

PROCESS SCHEDULING IN JUNOS OS

Innovation in Junos OS does not stop at the kernel level; rather, it extends to all aspects of system operation.

As mentioned before, there are two tiers of schedulers in Junos OS, the topmost becoming active in systems with a software data plane to ensure the real-time handling of incoming packets. It operates in real time and ensures that quality of service (QoS) requirements are met in the forwarding path.

The second-tier (non-real-time) scheduler resides in the base Junos OS kernel and is similar to its FreeBSD counterpart. It is responsible for scheduling system and user processes in a system to enable preemptive multitasking.

In addition, a third-tier scheduler exists within some multithreaded user-level processes, where threads operate in a cooperative, multitasking model. When a compound process gets the CPU share, it may treat it like a virtual CPU, with threads taking and leaving the processor according to their execution flow and the sequence of atomic operations. This approach allows closely coupled threads to run in a cooperatively multitasking environment and avoid being entangled in extensive IPC and resource-locking activities.

ARCHITECTURE AND INFRASTRUCTURE PARALLELISM

Advances in multicore CPU development and the capability to run several routing processors in a system constitute the basis for increased efficiency in a router control plane. However, finding the right balance of price and performance can also be very difficult. Unlike the data mining and computational tasks of supercomputers, processing of network updates is not a static job. A block of topology changes cannot be prescheduled and then sliced across multiple CPUs. In routers and switches, network state changes asynchronously (as events happen), thus rendering time-based load sharing irrelevant. Sometimes vendors try to solve this dilemma by statically sharing the load in functional, rather than temporal, domains. In other words, they claim that if the router OS can use separate routing processors for different tasks (for example, OSPF or BGP), it can also distribute the bulk of data processing across multiple CPUs. To understand whether this is a valid assumption, let's consider a typical CPU utilization capture. What is interesting here is that the different processes are not computationally active at the same time—OSPF and BGP do not compete for CPU cycles. Unless the router runs multiple same-level protocols simultaneously, the well-designed network protocol stack stays fairly orthogonal. Different protocols serve different needs and seldom converge at the same time.

Show Processes CPU Seconds Unicast

Date	*Average*	*RIP*	*OSPF*	*BGP*	*RIPng*	*OSPF6*	*BGP4+*	*RA*	*ISIS*
01/22 15:48:19	3	0	0	0	0	0	0	0	
01:22 15:48:20	3	0	1	0	0	0	0	0	
01/22 15:49:18	3	0	0	1	0	0	0	0	

Typical CPU times capture (from NEC 8800 product documentation).

For instance, an IGP topology change may trigger a Dijkstra algorithm computation; until it is complete, BGP nexthop updates do not make much sense. At the same time, all protected MPLS LSPs should fall on precomputed alternate paths and not cause major RSVP activities.

Thus, the gain from placing different processes of a single control plane onto physically separate CPUs may be limited, while the loss from the overhead functions such as synchronization and distributed memory unification may be significant.

Does this mean that the concept of parallelism is not applicable to the routing processors? Not at all. Good coding practice and modern compilers can make excellent use of multicore and SMP hardware, while clustered routing engines are indispensable when building multichasis (single control and data plane spanning multiple chassis) or segmented (multiple control and data planes within a single physical chassis) network devices. Furthermore, high-end designs may allow for independent scaling of control and forwarding planes, as implemented in the highly acclaimed Juniper Networks JCS1200 Control System.

With immediate access to state-of-the art processor technology, Juniper Networks engineers heavily employ parallelism in the Junos OS control plane design, targeting both elegance and functionality.

A functional solution is the one that speeds up the control plane without unwanted side effects such as limitations in forwarding capacity. When deployed in a JCS1200, Junos OS can power multiple control plane instances (system domains) at the same time without consuming revenue-

generating slots in the router chassis. Moreover, the Junos OS architecture can run multiple routing systems (including third-party code) from a single rack of routing engines, allowing an arbitrary mix-and-match of control plane and data plane resources within a point of presence (POP).

These unique capabilities translate into immediate CAPEX savings, because a massively parallel control plane can be built independent of the forwarding plane and will never confront a limited common resource (such as the number of physical routers or a number of slots in each chassis).

Elegance means the design should also bring other technical advantages: for instance, bypassing space and power requirements associated with the embedded chassis and thus enabling use of faster silicon and speeding up the control plane. Higher CPU speed and memory limits can substantially improve the convergence and scaling characteristics of the entire routing domain.

The goal of Juniper design philosophy is tangible benefits to our customers—without cutting corners.

FLEXIBILITY AND PORTABILITY

A sign of a good OS design is the capability to adapt the common software platform to various needs. In the network world, this equates to the adoption of new hardware and markets under the same operating system.

The capability to extend the common operating system over several products brings the following important benefits to customers:

- Reduced OPEX from consistent UI experience and common management interface

- Same code for all protocols; no unique defects and interoperability issues
- Common schedule for software releases; a unified feature set in the control plane
- Accelerated technology introduction; once developed, the feature ships on many platforms.

Technology companies are in constant search of innovation both internally and externally. New network products can be developed in-house or within partnerships or acquired. Ideally, a modern network OS should be able to absorb domestic (internal) hardware platforms as well as foreign (acquired) products, with the latter being gradually folded into the mainstream software line.

The capability to absorb in-house and foreign innovations in this way is a function of both software engineering discipline and a flexible, well-designed OS that can be adapted to a wide range of applications.

On the contrary, the continuous emergence of internally developed platforms from the same vendor featuring different software trains and versions can signify the lack of a flexible and stable software foundation.

For example, when the same company develops a core router with one OS, an Ethernet switch with another and a data centre switch with a third, this likely means that in-house R&D groups considered and rejected readily available OS designs as impractical or unfit.

Although partial integration may still exist through a unified command-line interface (CLI) and shared code and features, the main message is that the existing software designs were not flexible enough to be easily adapted to new

markets and possibilities. As a result, customers end up with a fractured software lineup, having to learn and maintain loosely related or completely unrelated software trains and develop expertise in all of them—an operationally suboptimal approach.

USE IN ROUTERS

Network Operating Systems (NOS) are embedded in a router or hardware firewall that operates the functions in the network layer (layer 3) of the OSI model.

Examples:

- JUNOS, used in routers and switches from Juniper Networks,
- Cisco IOS (formerly "Cisco Internetwork Operating System").
- TiMOS, used in routers from Alcatel-Lucent
- Huawei VRP (Versatile Routing Platform), used in routers from Huawei
- MikroTik RouterOS™ (is a router operating system and software which turns a regular Intel PC or MikroTik RouterBOARD™ hardware into a dedicated router.)
- ZyNOS, used in network devices made by ZyXEL.

PEER-TO-PEER

In a Peer-to-peer network operating system users are allowed to share resources and files located on their computers and access shared resources from others. This system is not based with having a file server or centralized management source. A peer-to-peer network sets all

connected computers equal; they all share the same abilities to utilize resources available on the network.

Examples:

- AppleShare used for networking connecting Apple products.
- Windows for Workgroups used for networking peer-to-peer windows computers.
- Lantastic.

ADVANTAGES

- Ease of setup
- Less hardware needed, no server needs to be purchased.

DISADVANTAGES

- No central location for storage.
- Lack of security that a client/server type offers.

CLIENT/SERVER

Client/server network operating systems allow the network to centralize functions and applications in one or more dedicated file servers. The server is the centre of the system, allowing access to resources and instituting security. The network operating system provides the mechanism to integrate all the components on a network to allow multiple users to simultaneously share the same resources regardless of physical location.

Examples:

- Novell Netware
- Windows Server.

ADVANTAGES

- Centralized servers are more stable.
- Security is provided through the server.
- New technology and hardware can be easily integrated into the system.
- Servers are able to be accessed remotely from different locations and types of systems.

DISADVANTAGES

- Cost of buying and running a server are high.
- Dependence on a central location for operation.
- Requires regular maintenance and updates.

SECURITY ISSUES INVOLVED IN USING A CLIENT/SERVER NETWORK

In a client/server network security issues may evolve at three different locations: the client, the network, and the server. All three points need to be monitored for unauthorized activity and need to be secured against hackers or eavesdroppers.

THE CLIENT

The client is the end user of the network and needs to be secured the most. The client end usually exposes data through the screen of the computer. Client connections to server should be secured through passwords and upon leaving their workstations clients should make sure that their connection to the server is securely cut off in order to make sure that no hackers or intruders are able to reach the server data. Not only securing the workstations connection to the

server is important but also securing the files on the workstation (client) is important as it ensures that no hackers are able to reach the system. Another possibility is that of introducing a virus or running unauthorized software on the client workstation thus threatening the entire information bank at the server.

The users themselves could also be a security threat if they purposely leave their IDs logged in or use easy IDs and passwords to enable hacking. Users may also be sharing their passwords in order to give the hackers access to confidential data. This can be overcome by giving passwords to each client and regularly asking clients to change their passwords. Also passwords should be checked for guess ability and for their strength and uniqueness.

THE NETWORK

The network allows transmission of data from the clients to the server. There are several points on the network where a hacker could eavesdrop or steal important packets of information. These packets may contain important confidential data such as passwords or company details. It is important that these networks are secured properly to keep unauthorized professionals away from all the data stored on the server. This can be done by encrypting important data being sent on the network. However, encryption may not be the only possible way of protecting networks as hackers can work their way around encryption. Another method could be conducting security audits regularly and ensuring identification and authorisation of individuals at all points along the network. This should discourage potential hackers. Making the entire environment difficult to impersonate also

makes sure that the clients are reaching the true files and applications on the server and that the server is providing information to authorized personnel only.

THE SERVER

The server can be secured by placing all the data in a secure, centralized location that is protected through permitting access to authorized personnel only. Virus protection should also be available on server computers as huge tons of data can be infected. Regular upgrades should be provided to the servers as the software and the applications need to be updated. Even the entire data on a server could be encrypted in order to make sure that getting through to the data takes a lot of effort and time.

3

Network Architecture

Network architecture refers to the layout of the network, consisting of the hardware, software, connectivity, communication protocols and mode of transmission, such as wired or wireless. Know about the types of network classified according to the areas covered such as LAN, MAN and WAN. Learn about the network topologies categorized according to the layout of equipments and computers such as star, loop, bus, or mesh topologies. There are many communication protocols used in the networking technology. It is important to know about the network architecture as networks play a very important role in today's world.

DEFINITION

Network architecture, is the logical and structural layout of the network consisting of transmission equipment, software and communication protocols and infrastructure (wired or wireless) transmission of data and connectivity between components.

TOPOLOGY

There are 4 different network topologies: star network, a bus or line network, a loop or ring network, and a mesh network.

TYPES OF NETWORKS

The different topologies can be arranged in different ways described as LAN (Local Area Network), MAN (Metropolitan Area Network) and WAN (Wide Area Network) where the network extends over a local area (<1 km), metropolitan (<100 km) and long distance.

NETWORK ARCHITECTURE (OSI)

Open Systems Interconnection (OSI) network architecture, developed by International Organization for Standardization, is an open standard for communication in the network across different equipment and applications by different vendors. Though not widely deployed, the OSI 7 layer model is considered the primary network architectural model for inter-computing and inter-networking communica-tions. In addition to the OSI network architecture model, there exist other network architecture models by many vendors, such as IBM SNA (Systems Network Architecture), Digital Equipment Corporation (DEC; now part of HP) DNA (Digital Network Architecture), Apple computer's AppleTalk, and Novell's NetWare. Network architecture provides only a conceptual framework for communications between computers. The model itself does not provide specific methods of communication. Actual communication is defined by various communication protocols.

ENHANCE NETWORK ARCHITECTURE WITH A SONA APPROACH

Good network architecture helps ensure that business strategy and IT investments are aligned. As the backbone for IT communications, the network element of enterprise architecture is increasingly critical. Service Oriented Network Architecture (SONA) is Cisco's architectural approach to designing advanced network capabilities into your infrastructure. SONA provides guidance, best practices, and blueprints for connecting network services and applications to enable business solutions. SONA is an open framework for network-based services used by enterprise applications to drive business results. This framework includes three interconnected layers.

Primary is the Core Common Services Layer, comprising an extensive library of network-based service categories working together to create functionality that can be used by the Applications Layer, which contains all types of business applications used across the enterprise. At the Physical Infrastructure Layer, Cisco designs, tests, and validates sets of modular, connected infrastructure elements organized by places in the network (PINs).

These branch, campus, and data centre reference solutions form a quick starting point for understanding how network-based services can be deployed with business applications in a variety of industries. The Core Common Services layer is distinctive to Cisco in that no other vendor offers the breadth and depth of integrated services throughout an enterprise-class network architecture.

This layer comprises seven major core service groups, which deliver consistent and robust capabilities throughout the network:

- Real-Time Communication Services offer session and media management capabilities, contact centre services, as well as presence functions.
- Mobility Services provide location information, as well as device dependent functionality.
- Application Delivery Services use application awareness to optimize performance.
- Security Services help protect the infrastructure, data, and application layers from constantly evolving threats, and also offer access-control and identity functions.
- Management Services offer configuration and reporting capabilities.
- Virtualization Services deliver abstraction between physical and functional elements in the infrastructure, allowing for more flexible and reliable service operations and management.
- Transport Services help with resource allocation and deliver on the overall QoS requirements of the application, as well as routing and topology functions.

Cisco Core Common Services are centered on two principles: application-focus and reusability. These services use a variety of open protocols (such as SIP and XML) and published APIs that allow IT developers as well as an innovative community of global development partners to improve reliability and performance and deliver new capabilities.

NETWORK LAYER

In the seven-layer OSI model of computer networking, the network layer is layer 3.

The network layer is responsible for packet forwarding including routing through intermediate routers, whereas the data link layer is responsible for media access control, flow control and error checking. The network layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination host via one or more networks while maintaining the quality of service functions.

Functions of the network layer include:

- *Connection model*: *connectionless communication*. For example, IP is connectionless, in that a datagram can travel from a sender to a recipient without the recipient having to send an acknowledgement. Connection-oriented protocols exist at other, higher layers of that model.
- *Host addressing*. Every host in the network must have a unique address that determines where it is. This address is normally assigned from a hierarchical system, so you can be "Fred Murphy" to people in your house, "Fred Murphy, 1 Main Street" to Dubliners, or "Fred Murphy, 1 Main Street, Dublin" to people in Ireland, or "Fred Murphy, 1 Main Street, Dublin, Ireland" to people anywhere in the world. On the Internet, addresses are known as Internet Protocol (IP) addresses.
- *Message forwarding*. Since many networks are partitioned into subnetworks and connect to other

networks for wide-area communications, networks use specialized hosts, called gateways or routers to forward packets between networks. This is also of interest to mobile applications, where a user may move from one location to another, and it must be arranged that his messages follow him. Version 4 of the Internet Protocol (IPv4) was not designed with this feature in mind, although mobility extensions exist. IPv6 has a better designed solution.

Within the service layering semantics of the OSI network architecture the network layer responds to service requests from the transport layer and issues service requests to the data link layer.

NETWORK LAYER FUNCTIONS

Some of the specific jobs normally performed by the network layer include:

- *Logical Addressing*: Every device that communicates over a network has associated with it a logical address, sometimes called a *layer three* address. For example, on the Internet, the Internet Protocol (IP) is the network layer protocol and every machine has an IP address. Note that addressing is done at the data link layer as well, but those addresses refer to local physical devices. In contrast, logical addresses are independent of particular hardware and must be unique across an entire internetwork.
- *Routing*: Moving data across a series of interconnected networks is probably the defining function of the network layer. It is the job of the

devices and software routines that function at the network layer to handle incoming packets from various sources, determine their final destination, and then figure out where they need to be sent to get them where they are supposed to go. The OSI model more completely in this topic on the topic on indirect device connection, and show how it works by way of an OSI model analogy.

- *Datagram Encapsulation*: The network layer normally encapsulates messages received from higher layers by placing them into *datagrams* (also called *packets*) with a network layer header.
- *Fragmentation and Reassembly*: The network layer must send messages down to the data link layer for transmission. Some data link layer technologies have limits on the length of any message that can be sent. If the packet that the network layer wants to send is too large, the network layer must split the packet up, send each piece to the data link layer, and then have pieces reassembled once they arrive at the network layer on the destination machine. A good example is how this is done by the Internet Protocol.
- *Error Handling and Diagnostics*: Special protocols are used at the network layer to allow devices that are logically connected, or that are trying to route traffic, to exchange information about the status of hosts on the network or the devices themselves.

NETWORK PROTOCOL

A protocol is a set of rules that governs the communications between computers on a network. These rules include

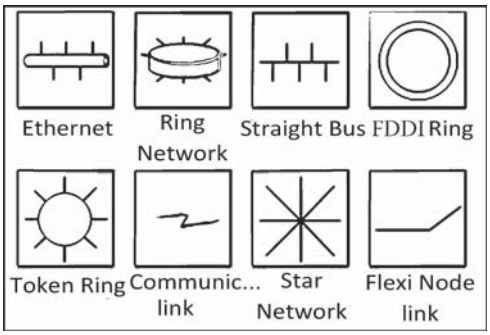
guidelines that regulate the following characteristics of a network: access method, allowed physical topologies, types of cabling, and speed of data transfer.

TYPES OF NETWORK PROTOCOLS

The most common network protocols are:

- Ethernet
- Local Talk
- Token Ring
- FDDI
- ATM.

The following is some common-used network symbols to draw different kinds of network protocols.



ETHERNET

The Ethernet protocol is by far the most widely used. Ethernet uses an access method called CSMA/CD (Carrier Sense Multiple Access/Collision Detection). This is a system where each computer listens to the cable before sending anything through the network. If the network is clear, the computer will transmit. If some other node is already transmitting on the cable, the computer will wait and try again when the line is clear. Sometimes, two computers

attempt to transmit at the same instant. When this happens a collision occurs. Each computer then backs off and waits a random amount of time before attempting to retransmit. With this access method, it is normal to have collisions. However, the delay caused by collisions and retransmitting is very small and does not normally effect the speed of transmission on the network. The Ethernet protocol allows for linear bus, star, or tree topologies. Data can be transmitted over wireless access points, twisted pair, coaxial, or fibre optic cable at a speed of 10 Mbps up to 1000 Mbps.

FAST ETHERNET

To allow for an increased speed of transmission, the Ethernet protocol has developed a new standard that supports 100 Mbps. This is commonly called Fast Ethernet. Fast Ethernet requires the use of different, more expensive network concentrators/hubs and network interface cards. In addition, category 5 twisted pair or fibre optic cable is necessary. Fast Ethernet is becoming common in schools that have been recently wired.

LOCAL TALK

Local Talk is a network protocol that was developed by Apple Computer, Inc. for Macintosh computers. The method used by Local Talk is called CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance). It is similar to CSMA/CD except that a computer signals its intent to transmit before it actually does so. Local Talk adapters and special twisted pair cable can be used to connect a series of computers through the serial port. The Macintosh operating system allows the establishment of a peer-to-peer network without

the need for additional software. With the addition of the server version of AppleShare software, a client/server network can be established.

The Local Talk protocol allows for linear bus, star, or tree topologies using twisted pair cable. A primary disadvantage of Local Talk is speed. Its speed of transmission is only 230 Kbps.

TOKEN RING

The Token Ring protocol was developed by IBM in the mid-1980s. The access method used involves token-passing. In Token Ring, the computers are connected so that the signal travels around the network from one computer to another in a logical ring. A single electronic token moves around the ring from one computer to the next. If a computer does not have information to transmit, it simply passes the token on to the next workstation. If a computer wishes to transmit and receives an empty token, it attaches data to the token. The token then proceeds around the ring until it comes to the computer for which the data is meant. At this point, the data is captured by the receiving computer. The Token Ring protocol requires a star-wired ring using twisted pair or fibre optic cable. It can operate at transmission speeds of 4 Mbps or 16 Mbps. Due to the increasing popularity of Ethernet, the use of Token Ring in school environments has decreased.

FDDI

Fiber Distributed Data Interface (FDDI) is a network protocol that is used primarily to interconnect two or more local area networks, often over large distances. The access method used by FDDI involves token-passing. FDDI uses a

dual ring physical topology. Transmission normally occurs on one of the rings; however, if a break occurs, the system keeps information moving by automatically using portions of the second ring to create a new complete ring. A major advantage of FDDI is speed. It operates over fibre optic cable at 100 Mbps.

ATM

Asynchronous Transfer Mode (ATM) is a network protocol that transmits data at a speed of 155 Mbps and higher. ATM works by transmitting all data in small packets of a fixed size; whereas, other protocols transfer variable length packets. ATM supports a variety of media such as video, CD-quality audio, and imaging. ATM employs a star topology, which can work with fibre optic as well as twisted pair cable. ATM is most often used to interconnect two or more local area networks. It is also frequently used by Internet Service Providers to utilize high-speed access to the Internet for their clients. As ATM technology becomes more cost-effective, it will provide another solution for constructing faster local area networks.

GIGABIT ETHERNET

The most recent development in the Ethernet standard is a protocol that has a transmission speed of 1 Gbps. Gigabit Ethernet is primarily used for backbones on a network at this time.

In the future, it will probably be used for workstation and server connections also. It can be used with both fibre optic cabling and copper. The 1000BaseTX, the copper cable used for Gigabit Ethernet, is expected to become the formal standard in 1999.

Compare the Network Protocols

Protocol	Cable	Speed	Topology
Ethernet	Twisted Pair, Coaxial, Fiber	10 Mbps	Linear Bus, Star, Tree
Fast Ethernet	Twisted Pair, Fiber	100 Mbps	Star
LocalTalk	Twisted Pair	.23 Mbps	Linear Bus or Star
Token Ring	Twisted Pair	4 Mbps - 16 Mbps	Star-Wired Ring
FDDI	Fiber	100 Mbps	Dual ring
ATM	Twisted Pair, Fiber	155-2488 Mbps	Linear Bus, Star, Tree

NETWORK DIAGRAMMING SOFTWARE

Edraw Network Diagrammer is a new, rapid and powerful network design software for network drawings with rich examples and templates. Easy to draw network topology, Cisco network design diagram, LAN/WAN diagram, network cabling diagrams, active directory, network planform and physical network diagram.

TCP/IP PROTOCOL ARCHITECTURE

TCP/IP protocols map to a four-layer conceptual model known as the *DARPA model*, named after the U.S. government agency that initially developed TCP/IP. The four layers of the DARPA model are: Application, Transport, Internet, and Network Interface. Each layer in the DARPA model corresponds to one or more layers of the seven-layer Open Systems Interconnection (OSI) model.

Figure shows the TCP/IP protocol architecture.

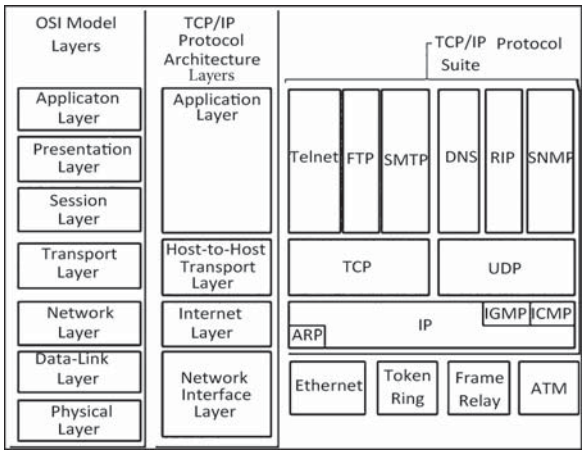


Fig. TCP/IP Protocol Architecture

NETWORK INTERFACE LAYER

The *Network Interface layer* (also called the Network Access layer) is responsible for placing TCP/IP packets on the network medium and receiving TCP/IP packets off the network medium. TCP/IP was designed to be independent of the network access method, frame format, and medium. In this way, TCP/IP can be used to connect differing network types. These include LAN technologies such as Ethernet and Token Ring and WAN technologies such as X.25 and Frame Relay. Independence from any specific network technology gives TCP/IP the ability to be adapted to new technologies such as Asynchronous Transfer Mode (ATM).

The Network Interface layer encompasses the Data Link and Physical layers of the OSI model. Note that the Internet layer does not take advantage of sequencing and acknowledgment services that might be present in the Data-Link layer. An unreliable Network Interface layer is assumed, and reliable communications through session establishment

and the sequencing and acknowledgment of packets is the responsibility of the Transport layer.

INTERNET LAYER

The *Internet layer* is responsible for addressing, packaging, and routing functions. The core protocols of the Internet layer are IP, ARP, ICMP, and IGMP.

- The *Internet Protocol* (IP) is a routable protocol responsible for IP addressing, routing, and the fragmentation and reassembly of packets.
- The *Address Resolution Protocol* (ARP) is responsible for the resolution of the Internet layer address to the Network Interface layer address such as a hardware address.
- The *Internet Control Message Protocol* (ICMP) is responsible for providing diagnostic functions and reporting errors due to the unsuccessful delivery of IP packets.
- The *Internet Group Management Protocol* (IGMP) is responsible for the management of IP multicast groups.

The Internet layer is analogous to the Network layer of the OSI model.

TRANSPORT LAYER

The *Transport layer* (also known as the Host-to-Host Transport layer) is responsible for providing the Application layer with session and datagram communication services. The core protocols of the Transport layer are *Transmission Control Protocol* (TCP) and the *User Datagram Protocol* (UDP).

- TCP provides a one-to-one, connection-oriented, reliable communications service. TCP is responsible

for the establishment of a TCP connection, the sequencing and acknowledgment of packets sent, and the recovery of packets lost during transmission.

- UDP provides a one-to-one or one-to-many, connectionless, unreliable communications service. UDP is used when the amount of data to be transferred is small (such as the data that would fit into a single packet), when the overhead of establishing a TCP connection is not desired or when the applications or upper layer protocols provide reliable delivery.

The Transport layer encompasses the responsibilities of the OSI Transport layer and some of the responsibilities of the OSI Session layer.

APPLICATION LAYER

The *Application layer* provides applications the ability to access the services of the other layers and defines the protocols that applications use to exchange data. There are many Application layer protocols and new protocols are always being developed.

The most widely-known Application layer protocols are those used for the exchange of user information:

- The Hypertext Transfer Protocol (HTTP) is used to transfer files that make up the Web pages of the World Wide Web.
- The File Transfer Protocol (FTP) is used for interactive file transfer.
- The Simple Mail Transfer Protocol (SMTP) is used for the transfer of mail messages and attachments.

- Telnet, a terminal emulation protocol, is used for logging on remotely to network hosts.

Additionally, the following Application layer protocols help facilitate the use and management of TCP/IP networks:

- The Domain Name System (DNS) is used to resolve a host name to an IP address.
- The Routing Information Protocol (RIP) is a routing protocol that routers use to exchange routing information on an IP internetwork.
- The Simple Network Management Protocol (SNMP) is used between a network management console and network devices (routers, bridges, intelligent hubs) to collect and exchange network management information.

Examples of Application layer interfaces for TCP/IP applications are Windows Sockets and NetBIOS. Windows Sockets provides a standard application programming interface (API) under Windows 2000. NetBIOS is an industry standard interface for accessing protocol services such as sessions, datagrams, and name resolution.

NETWORK ARCHITECTURES: LAYERS OF OSI MODEL AND TCP/IP MODEL

NETWORK ARCHITECTURE

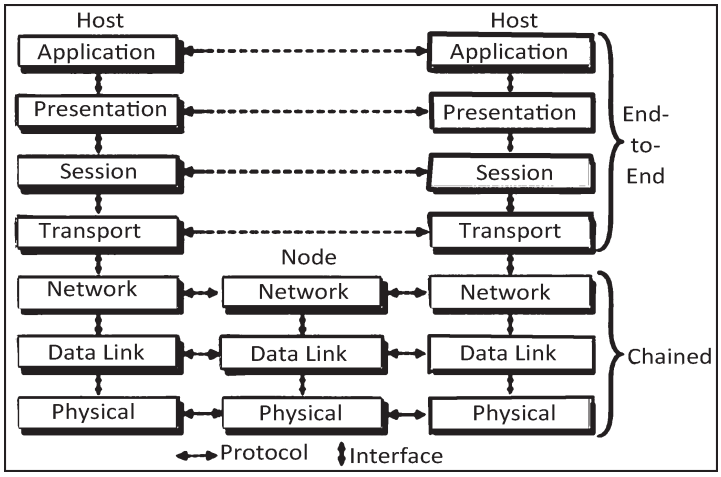
A Network is a conceptual framework that describes how data and network information are communicated from an application on one computer through network media to an application on other computers in terms of different layers. Network architecture is also known as Reference model. There

mainly two classifications of Reference models and are Open and closed. Open model is one which is open for everyone and no secrecy is there. In a closed model, also known as *proprietary* system the architecture is kept secret from users. OSI model is an open model while IBM's SNA 7 layer model is a closed system. Here Let's discuss two main reference models OSI model and TCP/IP model.

OSI ARCHITECTURE

International Organization for Standardization (ISO) developed OSI (Open Systems Interconnection) model in late 1970s. In this data communication functions are defined in terms of 7 layers which are organized in the chronological order of events occurring during a communications session. The top 3 layers explains how applications with in end stations communicate each other and also with users. The bottom 4 layers defines end to end data transmission.

OSI MODEL LAYERS



Application Layer - Layer 7: This is the layer where users communicate with computer. The main functions are file

transfers, e-mail, enabling remote access and network mangement activities. The common protocols in this layer are HTTP, FTP, TFTP, Telnet etc.

Presentation Layer - Layer 6: Also known as *'Syntax layer'*. This layer presents data to application layer in format that can be processed by an end user. This is actually a translator that provides coding and conversion functions. Main tasks are data compression, decompression, encryption and decryption.

Session Layer - Layer 5: This layer is responsible for setting up, managing and then tearing down session between two computers. The main function is to manage flow of data communication during a connection between 2 computers. Main services offered are Dialogue control, Token management and Synchronization. Dialogue control refers to organizing a data communication session between two computers via 3 modes namely simplex, half duplex and full duplex. In token management during a session between 2 computers if a critical operation is to be done they passes tokens and the one holding the token will do that operation. Synchronization is used to manage timing signal between session for the functions like insertion of check points.

Transport Layer - Layer 4: Provides end to end data transfer and establishes logic connections between sending host and destination host. Main functions are Flow control, Sequencing, error detection and recovery. Important protocols in this layer are TCP and UDP

Network Layer - Layer 3: This layer is the domain of WAN. Switches and Routers are devices of Network layer. Functions

are Logical addressing and Routing. The common protocols used by routers are RIP, EIGRP, OSPF etc.

Data-Link Layer - Layer 2: This functions the transfer of data between the ends of a link. It has two sub-layers namely Media Access Control Layer and Logical Link Control Layer. Packets received from network layer are transformed into frames. Devices in this layer are switches and bridges. Functions are Framing packets, Sequence control, Error control, Flow control, Physical addressing of devices in network, switching of LAN. Main protocols in this layer are HDLC, SDLC, PPP.

Physical Layer - Layer 1: Physical layer specifies electrical, mechanical, procedural and functional requirements for activating, maintaining and deactivating a physical link between end systems. The main function is to send bits and receive bits. Hub is the device in this layer. It also includes hardware interfaces to connect physical media.

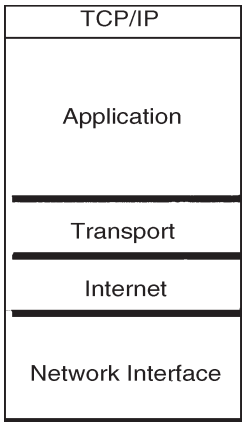
TCP/IP MODEL (INTERNET ARCHITECTURE)

TCP/IP (Transmission Control Protocol/Internet Protocol) defines a large collection of protocols that allow computers to communicate. It has a 4 layer architecture. TCP/ IP defines each of these protocols inside document called Requests For Comments (RFCs). By implementing the required protocols in TCP/IP RFCs, a computer can be relatively confident that it can communicate with other computers that also implement TCP/IP.

Application Layer: Provide services to the application software running on a computer. Application layer provides an interface between software running on a computer and

the network itself. Example for TCP/IP application is web browser. Example protocols are HTTP, POP3, SMTP etc.

Transport Layer: Consists of mainly two protocol options. Transmission control protocol (TCP) and User datagram protocol (UDP). Each layer provides a service to the layer above it. In same layer interaction on different computes, the two computers use protocol to communicate with the same layer on another computer. The protocol defined by each layer uses a header that is transmitted between the computers to communicate what each computer wants to do. While In Adjacent layer interaction on the same computer, one layer provides a service to a higher layer. The software or hardware that implements the higher layer requests that the next lower layer perform the needed function.

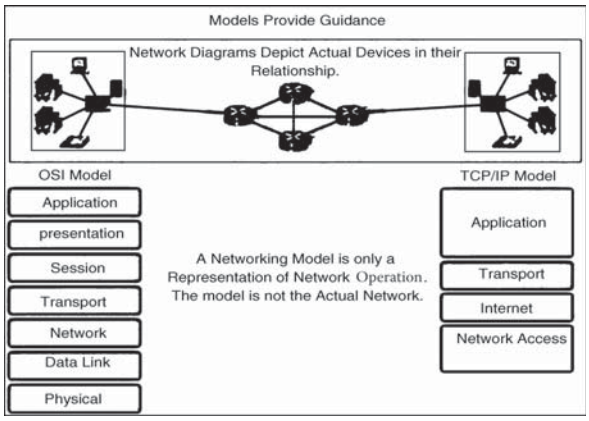


Internetwork Layer: Internet protocol (IP), works much like the postal service. IP defines logical addressing so that each host computer can have a different IP address. Similarly, IP defines the process of routing so that routers can choose where to send data correctly.

Network Interface Layer: Defines the protocols and hardware required to deliver data across some physical network. The term network interface refers to the fact that

this layer defines how to connect the host computer, which is not part of the network, to the network. It is the interface between the computer and network. Ethernet is one example protocol at the TCP/IP network interface layer. Ethernet defines the required cabling, addressing and protocols used to create an Ethernet. IP relies on the network interface layer to deliver IP packets across each physical network. IP understands the overall network topology, things such as which routers are connected to which networks, and what the IP addressing schemes looks like.

COMPARISON OF OSI AND TCP/IP MODELS



SIMILARITIES

- Both of them use a layered architecture to explain data communication process in computer networks.
- Each layer performs well-defined functions in both models.
- Similar types of protocols are used in both models.
- OSI and TCP/IP reference models are open in nature.
- Both models give a good explanation on how various types of network hardware and software interact during a data communication process.

- Data hiding principle is well maintained on each layer in the two models. The core level functional details of each layer are not revealed to other layers.
- Transport layer defines end-end data communication process and error-correction techniques in both the models.
- OSI and TCP/IP reference models process data in the form of packets to perform routing.

GSM NETWORK ARCHITECTURE

A GSM network is made up of multiple components and interfaces that facilitate sending and receiving of signalling and traffic messages. It is a collection of transceivers, controllers, switches, routers, and registers. A Public Land Mobile Network (PLMN) is a network that is owned and operated by one GSM service provider or administration, which includes all of the components and equipment. For example, all of the equipment and network resources that is owned and operated by Cingular is considered a PLMN.

MOBILE STATION (MS)

The Mobile Station (MS) is made up of two components:

Mobile Equipment (ME)—This refers to the physical phone itself. The phone must be able to operate on a GSM network. Older phones operated on a single band only. Newer phones are dual-band, triple-band, and even quad-band capable. A quad-band phone has the technical capability to operate on any GSM network worldwide. Each phone is uniquely identified by the *International Mobile Equipment Identity* (IMEI) number. This number is burned into the phone by the

manufacturer. The IMEI can usually be found by removing the battery of the phone and reading the panel in the battery well.

It is possible to change the IMEI on a phone to reflect a different IMEI. This is known as IMEI spoofing or IMEI cloning. This is usually done on stolen phones. The average user does not have the technical ability to change a phone's IMEI.

Subscriber Identity Module (SIM)—The SIM is a small smart card that is inserted into the phone and carries information specific to the subscriber, such as *IMSI, TMSI, Ki* (used for encryption), Service Provider Name (SPN), and *Local Area Identity* (LAI). The SIM can also store phone numbers (MSISDN) dialed and received, the *Kc* (used for encryption), phone books, and data for other applications. A SIM card can be removed from one phone, inserted into another GSM capable phone and the subscriber will get the same service as always.

Each SIM card is protected by a 4-digit Personal Identification Number (PIN). In order to unlock a card, the user must enter the PIN. If a PIN is entered incorrectly three times in a row, the card blocks itself and can not be used. It can only be unblocked with an 8-digit Personal Unblocking Key (PUK), which is also stored on the SIM card.

BASE TRANSCEIVER STATION (BTS)

The BTS is the Mobile Station's access point to the network. It is responsible for carrying out radio communications between the network and the MS. It handles speech encoding, encryption, multiplexing (*TDMA*), and modulation/ demodulation of the radio signals. It is also capable of frequency hopping. A BTS will have between 1 and 16 Transceivers (TRX), depending on the geography and user demand of an area. Each TRX represents one ARFCN. One BTS usually covers a single 120 degree sector of an area. Usually a tower with 3 BTSs will accommodate all 360 degrees around the tower. However, depending on geography and user demand of an area, a cell may be divided up into one or two sectors, or a cell may be serviced by several BTSs with redundant sector coverage.

A BTS is assigned a *Cell Identity*. The cell identity is 16-bit number (double octet) that identifies that cell in a particular *Location Area*. The cell identity is part of the Cell Global Identification (CGI), which is discussed in the section about the Visitor Location Register (VLR).

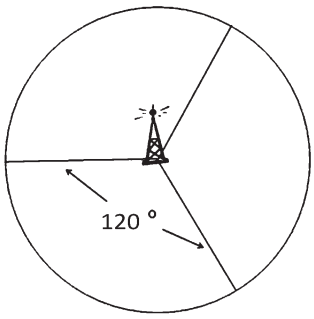


Fig. 120 ° Sector

The interface between the MS and the BTS is known as the *Um Interface* or the *Air Interface*.

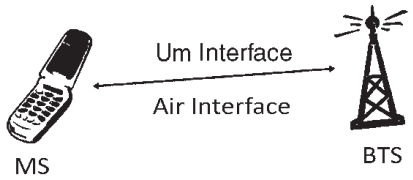


Fig. Um Interface

BASE STATION CONTROLLER (BSC)

The BSC controls multiple BTSs. It handles allocation of radio channels, frequency administration, power and signal measurements from the MS, and handovers from one BTS to another (if both BTSs are controlled by the same BSC). A BSC also functions as a "funneler". It reduces the number of connections to the *Mobile Switching Center* (MSC) and allows for higher capacity connections to the MSC. A BSC my be collocated with a BTS or it may be geographically separate. It may even be collocated with the Mobile Switching Center (MSC).

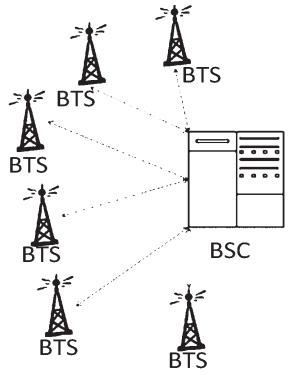


Fig. Base Station Controller

The interface between the BTS and the BSC is known as the *Abis Interface.*

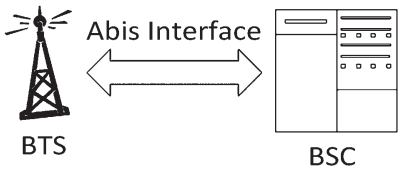


Fig. Abis Interface

The Base Transceiver Station (BTS) and the Base Station Controller (BSC) together make up the *Base Station System* (BSS).

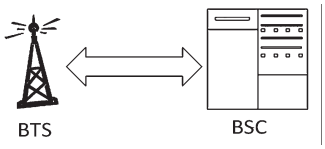


Fig. Base Station System

MOBILE SWITCHING CENTER (MSC)

The MSC is the heart of the GSM network. It handles call routing, call setup, and basic switching functions. An MSC handles multiple BSCs and also interfaces with other MSC's and registers. It also handles inter-BSC handoffs as well as coordinates with other MSC's for inter-MSC handoffs.

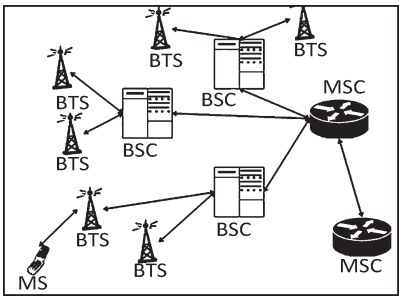


Fig. Mobile Switching Center

The interface between the BSC and the MSC is known as the *A Interface.*

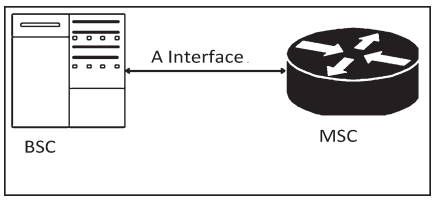


Fig. A Interface

GATEWAY MOBILE SWITCHING CENTER (GMSC)

There is another important type of MSC, called a Gateway Mobile Switching Center (GMSC).

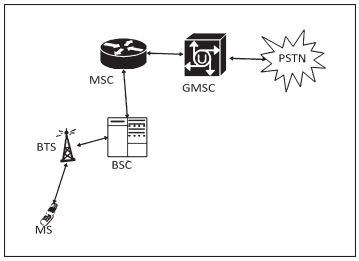


Fig. Gateway Mobile Switching Center

The GMSC functions as a gateway between two networks. If a mobile subscriber wants to place a call to a regular land line, then the call would have to go through a GMSC in order to switch to the Public Switched Telephone Network (PSTN). For example, if a subscriber on the Cingular network wants to call a subscriber on a T-Mobile network, the call would have to go through a GMSC.

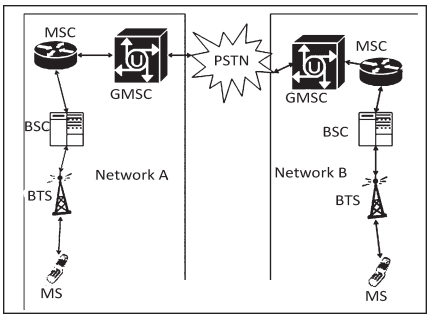


Fig. Connections between Two Networks

The interface between two Mobile Switching Centers (MSC) is called the *E Interface.*

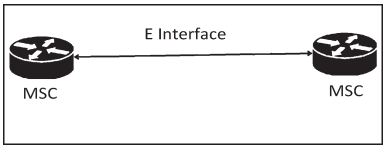


Fig. E Interface

HOME LOCATION REGISTER (HLR)

The HLR is a large database that permanently stores data about subscribers. The HLR maintains subscriber-specific

information such as the MSISDN, IMSI, current location of the MS, roaming restrictions, and subscriber supplemental features. There is logically only one HLR in any given network, but generally speaking each network has multiple physical HLRs spread out across its network.

VISITOR LOCATION REGISTER (VLR)

The VLR is a database that contains a subset of the information located on the HLR. It contains similar information as the HLR, but only for subscribers currently in its Location Area. There is a VLR for every Location Area. The VLR reduces the overall number of queries to the HLR and thus reduces network traffic. VLRs are often identified by the Location Area Code (LAC) for the area they service.

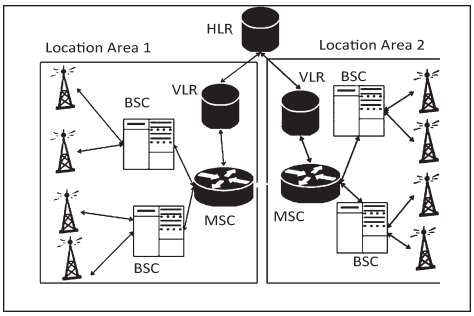


Fig. Visitor Location Register

LOCATION AREA CODE (LAC)

A LAC is a fixed-length code (two octets) that identifies a location area within the network. Each Location Area is serviced by a VLR, so we can think of a Location Area Code (LAC) being assigned to a VLR.

LOCATION AREA IDENTITY (LAI)

An LAI is a globally unique number that identifies the country, network provider, and LAC of any given Location

Area, which coincides with a VLR. It is composed of the Mobile Country Code (MCC), the Mobile Network Code (MNC), and the Location Area Code (LAC). The MCC and the MNC are the same numbers used when forming the IMSI.

CELL GLOBAL IDENTIFICATION (CGI)

The CGI is a number that uniquely identifies a specific cell within its location area, network, and country. The CGI is composed of the MCC, MNC, LAI, and Cell Identity (CI).

Cell Global Identity

CGI			
MCC	**MNC**	**LAC**	**Cell ID**
3 digits	2-3 digits	up to 5 digits	up to 5 digits

The VLR also has one other very important function: the assignment of a Temporary Mobile Subscriber Identity (TMSI). TMSIs are assigned by the VLR to a MS as it comes into its Location Area. TMSIs are only allocated when in cipher mode.

The interface between the MSC and the VLR is known as the *B Interface* and the interface between the VLR and the HLR is known as the *D Interface*. The interface between two VLRs is called the *G Interface.*

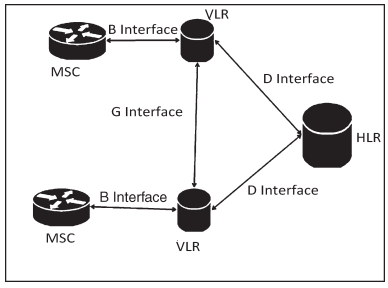


Fig. GSM Interfaces

EQUIPMENT IDENTITY REGISTER (EIR)

The EIR is a database that keeps tracks of handsets on the network using the IMEI. There is only one EIR per network. It is composed of three lists. The white list, the gray list, and the black list. The black list is a list if IMEIs that are to be denied service by the network for some reason. Reasons include the IMEI being listed as stolen or cloned or if the handset is malfunctioning or doesnt have the technical capabilities to operate on the network.

The gray list is a list of IMEIs that are to be monitored for suspicious activity. This could include handsets that are behaving oddly or not performing as the network expects it to. The white list is an unpopulated list. That means if an IMEI is not on the black list or on the gray list, then it is considered good and is "on the white list".

The interface between the MSC and the EIR is called the *F Interface*.

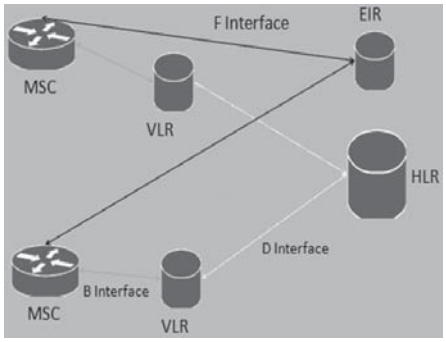


Fig. Equipment Identity Register

AUTHENTICATION CENTER (AUC)

The AuC is responsible for generating the necessary cryptovariables for authentication and encryption on the network. These variables are the RAND, SRES, and Kc. The Auc also stores the Ki for each IMSI on the network.

Although it is not required, the Auc is normally physically collocated with the HLR.

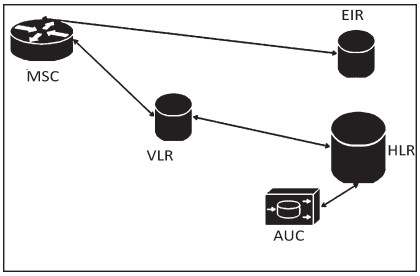


Fig. Authentication Center

There is one last interface that we haven't discussed. The interface between the HLR and a GMSC is called the *C Interface*. This completes the introduction to the network architecture of a GSM network. Below you will find a network diagram with all of the components as well as the names of all of the interfaces.

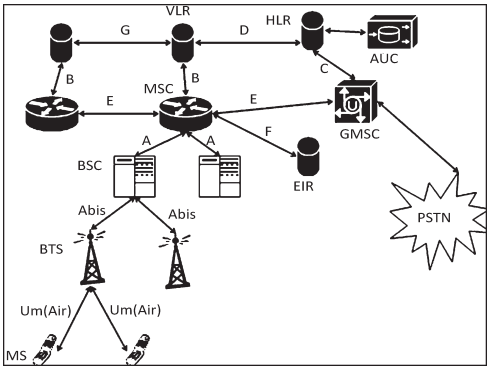


Fig. Full GSM Network

4

Operating Systems Structure

SYSTEM COMPONENTS

Even though, not all systems have the same structure many modern operating systems share the same goal of supporting the following types of system components.

PROCESS MANAGEMENT

The operating system manages many kinds of activities ranging from user programs to system programs like printer spooler, name servers, file server etc. Each of these activities is encapsulated in a process. A process includes the complete execution context (code, data, PC, registers, OS resources in use etc.).

It is important to note that a process is not a program. A process is only ONE instant of a program in execution. There are many processes can be running the same program.

The five major activities of an operating system in regard to process management are

- Creation and deletion of user and system processes.
- Suspension and resumption of processes.
- A mechanism for process synchronization.
- A mechanism for process communication.
- A mechanism for deadlock handling.

MAIN-MEMORY MANAGEMENT

Primary-Memory or Main-Memory is a large array of words or bytes. Each word or byte has its own address. Main-memory provides storage that can be access directly by the CPU. That is to say for a program to be executed, it must in the main memory.

The major activities of an operating in regard to memory-management are:

- Keep track of which part of memory are currently being used and by whom.
- Decide which processes are loaded into memory when memory space becomes available.
- Allocate and de allocate memory space as needed.

FILE MANAGEMENT

A file is a collected of related information defined by its creator. Computer can store files on the disk (secondary storage), which provide long term storage. Some examples of storage media are magnetic tape, magnetic disk and optical disk. Each of these media has its own properties like speed, capacity, data transfer rate and access methods.

A file systems normally organized into directories to ease their use. These directories may contain files and other directions.

The five main major activities of an operating system in regard to file management are

- The creation and deletion of files.
- The creation and deletion of directions.
- The support of primitives for manipulating files and directions.
- The mapping of files onto secondary storage.
- The back up of files on stable storage media.

I/O SYSTEM MANAGEMENT

I/O subsystem hides the peculiarities of specific hardware devices from the user. Only the device driver knows the peculiarities of the specific device to whom it is assigned.

SECONDARY-STORAGE MANAGEMENT

Generally speaking, systems have several levels of storage, including primary storage, secondary storage and cache storage. Instructions and data must be placed in primary storage or cache to be referenced by a running program. Because main memory is too small to accommodate all data and programs, and its data are lost when power is lost, the computer system must provide secondary storage to back up main memory. Secondary storage consists of tapes, disks, and other media designed to hold information that will eventually be accessed in primary storage (primary, secondary, cache) is ordinarily divided into bytes or words consisting of a fixed number of bytes. Each location in

storage has an address; the set of all addresses available to a program is called an address space. The three major activities of an operating system in regard to secondary storage management are:

- Managing the free space available on the secondary-storage device.
- Allocation of storage space when new files have to be written.
- Scheduling the requests for memory access.

NETWORKING

A distributed systems is a collection of processors that do not share memory, peripheral devices, or a clock. The processors communicate with one another through communication lines called network.

The communication-network design must consider routing and connection strategies, and the problems of contention and security.

PROTECTION SYSTEM

If a computer systems has multiple users and allows the concurrent execution of multiple processes, then the various processes must be protected from one another's activities. Protection refers to mechanism for controlling the access of programs, processes, or users to the resources defined by a computer systems.

COMMAND INTERPRETER SYSTEM

A command interpreter is an interface of the operating system with the user. The user gives commands with are

executed by operating system (usually by turning them into system calls). The main function of a command interpreter is to get and execute the next user specified command. Command-Interpreter is usually not part of the kernel, since multiple command interpreters (shell, in UNIX terminology) may be support by an operating system, and they do not really need to run in kernel mode. There are two main advantages to separating the command interpreter from the kernel.

- If we want to change the way the command interpreter looks, i.e., I want to change the interface of command interpreter, I am able to do that if the command interpreter is separate from the kernel. I cannot change the code of the kernel so I cannot modify the interface.
- If the command interpreter is a part of the kernel it is possible for a malicious process to gain access to certain part of the kernel that it showed not have to avoid this ugly scenario it is advantageous to have the command interpreter separate from kernel

OPERATING SYSTEMS SERVICES

Following are the five services provided by an operating systems to the convenience of the users.

PROGRAM EXECUTION

The purpose of a computer systems is to allow the user to execute programs. So the operating systems provides an environment where the user can conveniently run programs. The user does not have to worry about the memory allocation or multitasking or anything. These things are taken care

of by the operating systems. Running a program involves the allocating and deallocating memory, CPU scheduling in case of multiprocess. These functions cannot be given to the user-level programs. So user-level programs cannot help the user to run programs independently without the help from operating systems.

I/O OPERATIONS

Each program requires an input and produces output. This involves the use of I/O. The operating systems hides the user the details of underlying hardware for the I/O. All the user sees is that the I/O has been performed without any details. So the operating systems by providing I/O makes it convenient for the users to run programs.

For efficiently and protection users cannot control I/O so this service cannot be provided by user-level programs.

FILE SYSTEM MANIPULATION

The output of a program may need to be written into new files or input taken from some files. The operating systems provides this service. The user does not have to worry about secondary storage management. User gives a command for reading or writing to a file and sees his her task accomplished. Thus operating systems makes it easier for user programs to accomplished their task.

This service involves secondary storage management. The speed of I/O that depends on secondary storage management is critical to the speed of many programs and hence I think it is best relegated to the operating systems to manage it than giving individual users the control of it.

It is not difficult for the user-level programs to provide these services but for above mentioned reasons it is best if this service s left with operating system.

COMMUNICATIONS

There are instances where processes need to communicate with each other to exchange information. It may be between processes running on the same computer or running on the different computers. By providing this service the operating system relieves the user of the worry of passing messages between processes. In case where the messages need to be passed to processes on the other computers through a network it can be done by the user programs. The user program may be customized to the specifics of the hardware through which the message transits and provides the service interface to the operating system.

ERROR DETECTION

An error is one part of the system may cause malfunctioning of the complete system. To avoid such a situation the operating system constantly monitors the system for detecting the errors. This relieves the user of the worry of errors propagating to various part of the system and causing malfunctioning.

This service cannot allowed to be handled by user programs because it involves monitoring and in cases altering area of memory or deallocation of memory for a faulty process. Or may be relinquishing the CPU of a process that goes into an infinite loop. These tasks are too critical to be handed over to the user programs. A user program if given

these privileges can interfere with the correct (normal) operation of the operating systems.

SYSTEM CALLS AND SYSTEM PROGRAMS

System calls provide an interface between the process an the operating system. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do. In handling the trap, the operating system will enter in the kernel mode, where it has access to privileged instructions, and can perform the desired service on the behalf of user-level process. It is because of the critical nature of operations that the operating system itself does them every time they are needed. For example, for I/O a process involves a system call telling the operating system to read or write particular area and this request is satisfied by the operating system.

System programs provide basic functioning to users so that they do not need to write their own environment for program development (editors, compilers) and program execution (shells). In some sense, they are bundles of useful system calls.

LAYERED APPROACH DESIGN

In this case the system is easier to debug and modify, because changes affect only limited portions of the code, and programmer does not have to know the details of the other layers. Information is also kept only where it is needed and is accessible only in certain ways, so bugs affecting that data are limited to a specific module or layer.

MECHANISMS AND POLICIES

The policies what is to be done while the mechanism specifies how it is to be done. For instance, the timer construct for ensuring CPU protection is mechanism. On the other hand, the decision of how long the timer is set for a particular user is a policy decision.

The separation of mechanism and policy is important to provide flexibility to a system. If the interface between mechanism and policy is well defined, the change of policy may affect only a few parameters. On the other hand, if interface between these two is vague or not well defined, it might involve much deeper change to the system.

Once the policy has been decided it gives the programmer the choice of using his/her own implementation. Also, the underlying implementation may be changed for a more efficient one without much trouble if the mechanism and policy are well defined. Specifically, separating these two provides flexibility in a variety of ways. First, the same mechanism can be used to implement a variety of policies, so changing the policy might not require the development of a new mechanism, but just a change in parameters for that mechanism, but just a change in parameters for that mechanism from a library of mechanisms. Second, the mechanism can be changed for example, to increase its efficiency or to move to a new platform, without changing the overall policy.

TYPES OF OPERATING SYSTEMS

Operating system can be classified into various categories on the basic of several criteria, viz. a number of

simultaneously active programs, the number of users working simultaneously, and the number of processors in the computer system, etc. The main operating systems are as follows:

MULTI-USER

A multi-user operating system allows multiple users to access a computer system at the same time. Time-sharing systems and Internet servers can be classified as multi-user systems as they enable multiple-user access to a computer through the sharing of time. Single-user operating systems have only one user but may allow multiple programs to run at the same time.

MULTI-TASKING VS. SINGLE-TASKING

A multi-tasking operating system allows more than one program to be running at the same time, from the point of view of human time scales. A single-tasking system has only one running program. Multi-tasking can be of two types: pre-emptive and co-operative. In pre-emptive multitasking, the operating system slices the CPU time and dedicates one slot to each of the programs. Unix-like operating systems such as Solaris and Linux support pre-emptive multitasking, as does AmigaOS. Cooperative multitasking is achieved by relying on each process to give time to the other processes in a defined manner. 16-bit versions of Microsoft Windows used cooperative multi-tasking. 32-bit versions of both Windows NT and Win9x, used pre-emptive multi-tasking. Mac OS prior to OS X used to support cooperative multitasking.

DISTRIBUTED

A distributed operating system manages a group of independent computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they make a distributed system.

EMBEDDED

Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design. Windows CE and Minix 3 are some examples of embedded operating systems.

BATCH OPERATING SYSTEM

Batch processing is a rudimentary operating system. Batch processing generally requires the program, data, and appropriate system commands to be submitted together in the form of a job. Batch operating systems usually allow little or no interaction between users and executing programs. Batch processing has a greater potential for resource utilization than simple serial processing in computer systems serving multiple users. Due to turnaround delays and offline debugging, batch is not very convenient for program development. Programs that do not require interaction and programs with long execution times may be served well by

a batch operating system. Examples of such programs include payroll, forecasting, statistical analysis, and large scientific number-crunching programs. Serial processing combined with batch like command files is also found in many personal computers. Scheduling in batch is very simple. Jobs are typically processed in order of their submission, that is, first-come first-served basis.

Memory management in batch systems is also very simple. Memory is usually divided into two areas. The resident portion of the OS permanently occupies one of them, and the other is used to load transient programs for execution. When a transient program terminates, a new program is loaded into the same area of memory. Since at most one program is in execution at any time, batch systems do not require any time-critical device management. For this reason, many serial and I/O and ordinary batch operating systems use simple, program controlled method of I/O. The lack of contention for I/O devices makes their allocation and de-allocation trivial.

Batch systems often provide simple forms of file management. Since access to files is also serial, little protection is needed. No concurrency control of file access is required.

MULTIPROGRAMMING OPERATING SYSTEM

A multiprogramming system permits multiple programs to be loaded into memory and execute the programs concurrently. Concurrent execution of programs has a viable potential for improving system throughput and resource utilization relative to batch and serial processing. This

potential is realized by a class of operating systems that multiplex resources of a computer system among a multitude of active programs. Such operating systems usually have the prefix multi in their names, such as multitasking or multiprogramming.

TIME-SHARING SYSTEM

Time-sharing is a popular representative of multi-programmed, multi-user systems. In addition to general program-development environments, many large computer-aided design and text-processing systems belong to this category. One of the primary objectives of the multi-user systems in general, andthe time-sharing in particular, is good terminal response time, giving the illusion to each user of having a machine to oneself, as the time-sharing systems often attempt to provide equitable sharing of common resources. For example, when the system is loaded, users with more demanding processing requirements are made to wait longer.

This philosophy is reflected in the choice of scheduling algorithm. Most time-sharing systems use time-slicing scheduling. In this approach, programs are executed with rotating priority that increases during waiting and drops after the service is granted. In order to prevent programs from monopolizing the processor, a program executing longer than the system-defined time slice is interrupted by the OS,andit is placed at the end of the queue of waiting programs. This mode of operation generally provides quick response time to interactive programs. Memory management in time-sharing systems provides for isolation and protection

of co-resident programs. Some forms of controlled sharing are sometimes provided to conserve memory and possibly to exchange data between programs. Being executed on behalf of different users, programs in time-sharing systems generally do not have much need to communicate with each other.

As in most multi-user environments, allocation and de-allocation of devices must be done in a manner that preserves system integrity and provides good performance.

REAL-TIME SYSTEMS

Real time systems are used in time critical environments where data must be processed extremely quickly because the output influences immediate decisions. Real time systems are used for space flights, airport traffic control, industrial processes, sophisticated medical equipments, telephone switching, etc. A real time system must be 100 percent responsive in time. Response time is measured in fractions of seconds. In real time systems correctness of computations not only depends upon the logical correctness of computation but also upon the time at which the results are produced. If the timing constraints of the system are not met, system failure is bound to occur. Real-time operating systems are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines.

A primary objective of real-time systems is to provide quick event-response time, and to meet the scheduling deadlines. User convenience and resource utilization are of

secondary concern to real-time system designers. It is not uncommon for a real-time system to be expected to process bursts of thousands of interrupts per second without missing a single event. Such requirements usually cannot be met by multi-programming alone, and real-time operating systems usually rely on some specific policies and techniques for doing their job.

The Multitasking operation is accomplished by scheduling processes for execution, independent of each other. Each process is assigned a certain level of priority that corresponds to the relative importance of the event that if services. The processor is normally allocated to the highest-priority process among those that are ready to execute. Higher-priority processes usually preempt execution of the lower-priority processes. This form of scheduling, called priority-based preemptive scheduling, is used by a majority of real-time system. Unlike, say, time-sharing, the process population in real-time systems is fairly static, and there is comparatively little moving of programs between primary andsecondary storage.

On the other hand, processes in real-time systems tend to cooperate closely, thus necessitating support for both separation and sharing of memory. Moreover, as already suggested, time-critical device management is one of the main characteristics of real-time systems. In addition to providing sophisticated forms of interrupt management and I/O buffering, real-time operating systems often provide system calls to allow user processes to connect themselves to interrupt victors and to service events directly. File management is usually found only in larger installations of

real-time system. In fact, some embedded real-time system, such as an onboard automotive controller, may not even have any secondary storage. The primary objective of file management in real-time systems is usually the speed of access, rather than efficient utilization of secondary storage.

COMBINATION OF OPERATING SYSTEMS

Different types of OS are optimized of geared up to serve the needs of specific environments. In practice, however, a given environment may not exactly fit in any of the described models. For instance, both interactive program development and lengthy simulation are often encountered in university computing centers. For this reason, some commercial operating systems provide a combination of the described services. For example, a time-sharing system may support interactive users and also incorporate a full- fledged batch monitor. This allows computationally intensive non-interactive programs to be run concurrently with interactive programs. The common practice is to assign low priority to batch jobs and thus execute batched programs only when the processor would otherwise be idle. In other words, batch may be used as a filler to improve processor utilization while accomplishing a useful service of its own. Similarly, some time-critical events, such as receipt and transmission of network data packets, may be handled in real-time fashion on systems that otherwise provide time-sharing services to their terminal users.

DISTRIBUTED OPERATING SYSTEMS

A distributed computer system is a collection of autonomous computer system capable of communication

and cooperation via their hardware and software interconnections. Historically, distributed computer system has evolved from computer networks in which a number of largely independent hosts are connected by communication links and protocols. A distributed OS governs the operation of a distributed computer system and provides a virtual machine abstraction to its users. The key objective of a distributed OS is transparency.

Ideally, component and resource distribution should be hidden from users and application programs unless they explicitly demand otherwise. Distributed operating system usually provides the means for system-wide sharing of resources, such as computational capacity, files and I/O devices. In addition to typical operating- system services provided at each node for the benefit of local clients, a distributed OS may facilitate access to remote resources, communication with remote processes, and distribution of computations. The added services necessary for pooling of shared system resources include global naming, distributed file system, and facilities for distribution.

UNIX

The UNIX system is a multi-user, multi tasking operating system which means that it allows a single or multiprocessor computer to simultaneously execute several programs by one or several users. It has one or several command interpreters (shell) as well as a great number of commands and many utilities (assembler, compilers for many languages, text processing, email, etc.). Furthermore, it is highly portable, which means that it is possible to implement a

UNIX system on almost all hardware platforms. The UNIX systems have a strong foothold in professional and university environments thanks to their stability, their increased level of security and observance of standards, notably in terms of networks. The first "Unix" system was developed by Ken Thompson in the Bell ATandT laboratories at Murray Hill in New Jersey in the United States from 1965. Ken Thompson's aim was to develop a simple interactive operating system; called "Multics" (Multiplexed Information and Computing System) in order to run a game which he had created (space travel, a simulation of the solar system).

Few commands used in UNIX are:

- ls: lists your files.
- more: shows the first part of a file, just as much as will fit on one screen.
- Chmod: lets you change the read, write, and execute permissions on your files.
- mv: moves a file (i.e. gives it a different name, or moves it into a different directory
- cp: copies a file
- wc: tells you how many lines, words, and characters there are in a file.
- pwd: tells where you are currently.

WINDOWS

The first version of Microsoft Windows (Microsoft Windows 1.0) came out in November 1985. It had a graphical user interface, inspired by the user interface of the Apple computers of the time. Windows 1.0 was not successful

with the public, and Microsoft Windows 2.0, launchedon December 9, 1987, did not do much better. It was on May 22, 1990 that Microsoft Windows became a success, with Windows 3.0, then Windows 3.1 in 1992, and finally Microsoft Windows for Workgroups, later renamed Windows 3.11, which included network capabilities. Windows 3.1 cannot be considered an entirely separate operating system because it was only a graphical user interface running on top of MS-DOS. Concurrent with these releases, Microsoft had been selling (since 1992) an entirely 32-bit operating system (which therefore was not based on MS-DOS) for professional use, at a time when business primarily used mainframes. It was Windows NT (for Windows "New Technology"). Windows NT was not a new version of Windows 95 or an improvement on it, but an entirely different operating system.

WINDOWS NT

Windows NT (for "New Technology") is a 32-bit operating system developed by Microsoft.

Windows NT's outward appearance makes it look a lot like Windows 95/98/Millennium, but Windows NT has a separately developed kernel.

Because of this, Windows NT has the following characteristics:

- Windows NT is a pre-emptive multitasking system.
- Windows NT is a multi-user system, which means that depending on the user who is connected to the system, the interface might be different, as might system privileges.

- Windows NT natively supports numerous network features.
- Windows NT has more security, in particular for the file system (NTFS) as well as for the robustness of the OS.

Windows NT manages users, the network administrator is able to control privileges for each user connected to the system.

What's more, with the NTFS file system, which includes the capability to assign ownership for a file, each user or user group can be assigned specific access privileges for different system files.

When a user opens a session, the sequence CTRL+ALT+DEL opens another dialog box called Windows NT Security. These are the options it offers:

- Lock workstation: Ensures the computer's security without closing the session. All applications will continue running. A system can only be unlocked by the current user or by an administrator.
- Change password: Lets a user change his or her password. The user must know the current password to be able to change it.
- Close session: Closes the current session, but NT's services will stay active. For security reasons, you should always close a session when you no longer need to use the computer.
- Task manager: Shows which applications are running. This option also lets you switch between applications and end an application that has stopped responding.

- Shut Down: Closes all files, saves system data, and prepares the server to be safely switched off.
- Cancel: Closes the dialogue box.

Windows 8, the successor to Windows 7, was released to the market on 26 October 2012.

CAUTION

For security reasons, we must always close our session when we do not have longer use of computer.

WINDOWS98

Windows 98 is an operating system that allows use of different types of applications or software. For example, it allows you to use a word processing application to write a letter, and a spreadsheet application to track your financial information. Windows 98 is a graphical user interface (GUI). It has pictures (graphical) that you use (user) to communicate (interface) with the computer. This type of system is popular because it's logical, entertaining and easy to use.

This operating system has multitasking capabilities, meaning that it can run several applications at the same time. Multitasking allows you to view this lesson on the Internet at the same time that you practise using other applications with Windows 98.

The opening screen of Windows 98 is called the desktop. This workspace on your computer screen contains:

- Start button.
- Taskbar.
- Icons or graphical pictures.

WINDOW XP

Windows XP was introduced in November 2001 with a great sales campaign. Compared with the previous Windows Me, there has also been a very extensive updating of Windows.

The main features in the new program are:

- Technologically, Windows XP is based on the Windows NT and Windows 2000 programs. With this, Windows XP is a genuine 32 bit program.
- Windows XP replaces Windows 2000, Windows NT and Windows 98/Me.
- Windows XP has a new user interface with new buttons, icons and windows.
- Windows XP is optimized to work with digital pictures, sound and video-recordings (with use of the modern pc-plugs USB and FireWire).

WINDOWS OPERATING SYSTEM

An "operating system" is a program that manages the resources of the computer. An operating system sets up a consistent way for programs to request resources, such as time on the processor, or space in memory, from the computer itself. Operating systems look after all the devices attached to the computer, such as printers, modems, disks, and terminals. Another part of an operating system's job is to maintain a file system; that is, to set up a consistent way for information to be stored and retrieved.

BACKGROUND OF OPERATING SYSTEM

The first version of Windows was released in November of 1985. This program wasn't very popular as it lacked

functionality compared to the Apple operating system. Version 1.0 was not a complete system. Instead, it simply extended on MS-DOS. Version 2.0 was released two years later and achieved slightly more popularity than its predecessor. Version 2.03 was released in January of 2008. This version offered a totally different look that resulted in Apple filing a lawsuit against Microsoft with accusations of infringement.

Windows version 3.0 was released in 1990, the first edition to reach commercial success by selling two million copies within its first six months. This version included numerous improvements to the user interface along with new multitasking capabilities. Version 3.1 offered a facelift and was made available in March, 1992.

The Windows NT operating system was released in July of 1993. This version was based on a new kernel and it was considered to be the first designed for a professional platform. NT was later upgraded to function as a home user operating system with the release of Windows XP.

In August of 1995, Windows 95 was released. This operating system offered a consumer solution with significant changes to the user interface that also utilized preemptive multitasking. Windows 95 was introduced to replace version 3.1 and Windows for Workgroups as well as MS-DOS. The first Microsoft operating system to use the plug and play system, Windows 95 revolutionized the desktop platform and achieved mass popularity.

Next up was Windows 98, released in June of 1998. This operating system was criticized for being slower and less

reliable than version 95. Many of those issues were addressed a year later with the unveiling of Windows 98 Second Edition. Microsoft continued their line of professional operating systems with Windows 2000 in February of 2000. The consumer version was released as Windows ME in September of that year. ME integrated several new technologies, most notably the Universal Plug and Play.

Windows XP was released in October 2001. This version was based on the NT kernel and managed to retain the extreme functionality of its home-based predecessors. XP was widely embraced by the public and came in two different editions: Home and Professional. The Home Edition provided exceptionable multimedia support while the Professional edition offered excellent security and networking capabilities. XP has since been succeeded by Vista but support will continue through April of 2009.

WORKING OF OPERATING SYSTEM

When you turn on your computer, it's nice to think that you're in control. There's the trusty computer mouse, which you can move anywhere on the screen, summoning up your music library or Internet browser at the slightest whim. Although it's easy to feel like a director in front of your desktop or laptop, there's a lot going on inside, and the real man behind the curtain handling the necessary tasks is the operating system.

Most desktop or laptop PCs come pre-loaded with Microsoft Windows. Macintosh computers come pre-loaded with Mac OS X. Many corporate servers use the Linux or UNIX operating systems. The operating system (OS) is the

first thing loaded onto the computer — without the operating system, a computer is useless.

More recently, operating systems have started to pop up in smaller computers as well. If you like to tinker with electronic devices, you're probably pleased that operating systems can now be found on many of the devices we use every day, from cell phones to wireless access points. The computers used in these little devices have gotten so powerful that they can now actually run an operating system and applications. The computer in a typical modern cell phone is now more powerful than a desktop computer from 20 years ago, so this progression makes sense and is a natural development.

The purpose of an operating system is to organize and control hardware and software so that the device it lives in behaves in a flexible but predictable way. In this chapter, we'll tell you what a piece of software must do to be called an operating system, show you how the operating system in your desktop computer works and give you some examples of how to take control of the other operating systems around you.

Not all computers have operating systems. The computer that controls the microwave oven in your kitchen, for example, doesn't need an operating system. It has one set of tasks to perform, very straightforward input to expect (a numbered keypad and a few pre-set buttons) and simple, never-changing hardware to control. For a computer like this, an operating system would be unnecessary baggage, driving up the development and manufacturing costs significantly and adding complexity where none is required.

Instead, the computer in a microwave oven simply runs a single hard-wired program all the time.

For other devices, an operating system creates the ability to:

- Serve a variety of purposes
- Interact with users in more complicated ways
- Keep up with needs that change over time

All desktop computers have operating systems. The most common are the Windows family of operating systems developed by Microsoft, the Macintosh operating systems developed by Apple and the UNIX family of operating systems (which have been developed by a whole history of individuals, corporations and collaborators). There are hundreds of other operating systems available for special-purpose applications, including specializations for mainframes, robotics, manufacturing, real-time control systems and so on.

In any device that has an operating system, there's usually a way to make changes to how the device works. This is far from a happy accident; one of the reasons operating systems are made out of portable code rather than permanent physical circuits is so that they can be changed or modified without having to scrap the whole device.

For a desktop computer user, this means you can add a new security update, system patch, new application or even an entirely new operating system rather than junk your computer and start again with a new one when you need to make a change. As long as you understand how an operating system works and how to get at it, in many

cases you can change some of the ways it behaves. The same thing goes for your phone, too.

MICROSOFT WINDOWS AND OTHER USERS

A Windows user can still use the above MS-DOS steps if they wish to create a batch file.

If, however, you're more comfortable using Microsoft Windows or your operating system, you can use any text editor, such as Notepad or WordPad, to create your batch files, as long as the file extension ends with .bat. In the below example we use the Windows notepad to create a batch file.

- Click Start
- Click Run
- *Type*: notepad and press enter.
- Once notepad is open, type the below lines in the file or copy and paste the below lines into notepad.

  @echo off echo Hello this is a test batch file

  pause

  dir c:\windows
- Click File and click Save; browse to where you want to save the file. For the file name, type "test.bat", and if your version of Windows has a "Save as type" option, choose "All files", otherwise it will save as a text file. Once all of this has been done click the Save button and exit notepad.
- Now, to run the batch file, simply double-click or run the file like any other program. Once the batch file has completed running it will close the window automatically.

WINDOWS SYSTEM PROTECTION

From the beginning of electronic computing until 15 years ago, the 'game' of attack and defense was played on a system by system basis, with defenders relying on physical security and ad-hoc operating system protection methods and attackers guessing passwords or exploiting errors and omissions to bypass normal system controls. Over the last 15 years, the computing environment has changed dramatically, with widespread physical distribution of computing power, almost complete loss of physical control over computing hardware, and a dramatic increase in the networking of computers, but defenses have not changed substantially in terms of their reliance on physical security and ad-hoc defenses. As a result, we see new classes of attacks such as computer viruses, which exploit the lack of physical control and fundamental weakness of existing logical controls to spread transitively throughout the computing world. Even the best protection systems available today can quickly and easily be defeated by anyone with physical access and ample knowledge, or by the application of that expertise in a widespread computer virus attack.

One of the major factors in the successful application of information protection techniques is the exploitation of computational advantage. Computational advantage shows up historically in cryptography, where Shannon's theory clearly demonstrates the effect of 'workload' on the complexity of cryptanalysis and introduces the concepts of diffusion and confusion as they relate to statistical attacks on cryptosystems. Most modern cryptosystems exploit this as their primary defense The same basic principle applies in

computer virus analysis in which evolutionary viruses drive the complexity of detection and eradication up dramatically and in password protection in which we try to drive the number of guesses required for a successful attack up by limiting the use of obvious passwords . As we will see, one of the major reasons attacks succeed is because of the static nature of defense, and the dynamic nature of attack.

THE ULTIMATE ATTACK

The ultimate attack against any system begins with physical access, and proceeds to disassembly and reverse engineering of whatever programmed defenses are in place. Even with a cryptographic key provided by the user, an attacker can modify the mechanism to examine and exploit the key, given ample physical access. Eventually, the attacker can remove the defenses by finding decision points and altering them to yield altered decisions.

Without physical protection, nobody has ever found a defense against this attack, and it is unlikely that anyone ever will. The reason is that any protection scheme other than a physical one depends on the operation of a finite state machine, and ultimately, any finite state machine can be examined and modified at will, given enough time and effort.

The best we can ever do is delay attack by increasing the complexity of making desired alterations.

With any static defense in widespread use, attackers can and may eventually find a technique to bypass protection and incorporate the technique in a virus. In the modern environment, this means that a virus writer can eventually

exploit such a weakness to evade protection on a large number of systems. This has already taken place to some degree, with attackers producing computer viruses designed to exploit low-level operating system assumptions and bypass specific defensive products and techniques.

To the extent that defenses have generic weaknesses, this problem will continue unabated, but even in the cases of defenses with no known generic weaknesses, attackers may succeed by using the ultimate attack and programming the results into a virus.

Since the program that runs first 'wins' , and a virus in the boot block of a floppy disk runs first on most modern computer systems, an attacker might exploit knowledge gained from an ultimate attack on one system by placing the successful technique in a boot block initiated virus, and carry the attack to numerous machines. In the case of low-level viruses operating on IBM compatible PCs, the same attacks that work against the DOS operating system (e.g. The 'Stoned' virus and its variants) succeed even when the Unix operating system is used.

THE ULTIMATE DEFENSE

The ultimate defense is to drive the complexity of the ultimate attack up so high that the cost of attack is too high to be worth performing. This is, in effect, security through obscurity, and it is our general conclusion that all technical information protection in computer systems relies at some level either on physical protection, security through obscurity, or combinations thereof.

The goal of security through obscurity is to make the difficulty of attack so great that in practice, it is not worth performing, even though it could eventually be successful. Successful attacks against obscurity defenses depend on the ability to guess some key piece of information. The most obvious example is attacking and defending passwords, and since this problem demonstrates precisely the issues at hand, we will use it as an example. In password protection, there are generally three aspects to making attack difficult. One aspect is making the size of the password space large, so that the potential number of guesses required for an attack is enormous. The second aspect is spreading the probability density out so that there is relatively little advantage to searching the space selectively. This is basically the same as Shannon's concept of diffusion. The third aspect is obscuring the stored password information so that the attacker cannot simply read it in stored form. This is basically the same as Shannon's concept of confusion.

In successful password attacks, the size of the password space is usually substantial, but the way people select passwords leads to very small high probability subspaces. For example, the user's identification to the system spelled backwards is a very common authentication string. In numerous studies performed over many years, the vast majority of user's passwords could be guessed in only a few minutes . The third aspect of password protection usually involves using access controls in combination with trapdoor encryption to drive up the time required to try substantial numbers of guesses.

In the case of current operating systems, the size of the space is enormous, consisting of all programs that fit in the computer's memory, but the operating system may only have a very small number of versions, all of which are almost identical, yielding a highly coherent subspace consisting of only a few very closely tied points. Thus, only a few guesses are required to determine precisely which version of the operating system is in use, and even that may not be required for many attacks. Operating systems also provide no confusion in that the part of the program that performs any given operation is immediately apparent to the knowledgeable attacker. For this reason, low-level operating system attack is quite simple. The problem for defenders is to find a way to increase the difficulty of operating system attack by reducing coherence. The ultimate goal is to obscure defenses so as to make attackers require repeated use of the ultimate attack in order to impact substantial numbers of systems. One solution is to provide each computer with a sound and unique defense. This would tend to make it infeasible to design an automated attack that could systematically bypass all of the defenses, but then we would have to design a new defense for each system, and the costs of defense would probably become intolerable. We could trade off costs for probability of attack by implementing some fixed number of different defenses, thus requiring a fixed number of ultimate attacks for complete success. This is essentially the situation in the world today, where a wide variety of ad-hoc defenses are on the market. Unfortunately, many of these defenses fall to the same classes of attack, and the number is sufficiently

small that defeating most of them requires relatively little effort or expense. A more practical solution to this problem might be the use of evolutionary defenses. In order to make such a defensive strategy cost effective for numerous variations (e.g. one per computer worldwide), we probably have to provide some sort of automation. If the automation is to be effective, it must produce a large search space and provide a substantial degree of confusion, and diffusion. This then is the goal of evolutionary defenses.

Evolution can be provided in many ways and at many different places, ranging from a small finite number of defenses provided by different vendors, and extending toward a defensive system that evolves itself during each system call. With more evolution, we get less performance, but higher cost of attack. Thus, as in all protection functions, there is a price to pay for increased protection. Assuming we can find reasonably efficient mechanisms for effective evolution, we may be able to create a great deal of diversity at practically no cost to the end-user, while making the cost of large scale attack very high. As a very pleasant side effect, the ultimate attack may become necessary for each system under attack. In other words, except for endemic flaws, attackers may again be reduced to a case-by-case expert attack and defense scenario involving physical access.

TECHNIQUES FOR PROGRAM IN WINDOWS

We will consider two programs equivalent if, given identical input sequences, they produce identical output sequences. The equivalence of two programs is undecidable as is the determination of whether one program can evolve from

another . This result would seem to indicate that evolution has the potential for increasing complexity of analysis, and thus difficulty of attack. In a practical operating system design, we may also have very stringent requirements on the space and time used by the protection mechanism, and certain instruction sequences may be highly undesirable because they impact some other aspect of system operation or are incompatible across some similar machines. For this reason, we may not be able to reach the levels of complexity required to eliminate concerted human attack, but we may succeed in increasing the complexity of automated attacks to a level where the time required for attack is sufficient to have noticeable performance impacts, even to a level where no attacker is able to design a strong enough attack to defeat more than a small number of evolutions.

We know that evolution is as general as Turing machine computation , and that an exhaustive set of equivalent programs is easily described mathematically (i.e. the definition of equivalence), but this is not particularly helpful in terms of designing practical evolutionary schemes. We now describe a number of practical techniques we have explored for program evolution and some results regarding the space, time, and complexity issues introduced by these techniques.

OPERATING SYSTEM PRINCIPLE

An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is an essential component of the system software

in a computer system. Application programs usually require an operating system to function.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and will frequently make a system call to an OS function or be interrupted by it. Operating systems can be found on almost any device that contains a computer—from cellular phones and video game consoles to supercomputers and web servers.

Examples of popular modern operating systems include Android, BSD, iOS, Linux, OS X, QNX, Microsoft Windows, Windows Phone, and IBM z/OS. All these, except Windows, Windows Phone and z/OS, share roots in UNIX.

Generally an operating system can be defined as follows:

(i) A technical layer of software for driving the hardware of the computer, like disk drives, the keyboard, and the screen.

(ii) A file system which provides a way of organizing files logically.

(iii) A simple command language which enables users to run their own programs and to manipulate their files in a simple way. Some operating systems also provide text editors, compilers, debuggers and a variety of

other tools. Since the operating system (OS) is in charge of a computer, all requests to use its resources and devices need to go through the OS. An OS therefore provides legal entry points into its code for performing basic operations such as writing to devices.

Operating systems may be classified by both the number of tasks they can perform 'simultaneously', and on the basis of users ofthe system simultaneously. There are:

1. Single-user or multi-user.
2. Single-task or multi-tasking.

A multi-user system must clearly be multi-tasking. The first of these (MS/PC DOS/Windows 3x) are single user, single-task systems, which are built on a ROM - based library of basic functions known as the BIOS. These are system calls which write to the screen or to disk, etc. Although all the operating systems can serve as interrupts, and therefore simulate the appearance of multitasking in some situations, the older PC environments cannot be thought of as a multi-tasking system in any sense. Only a single user application could be opened at any time. Windows 95 replaced the old co-routine approach of quasi-multitasking with a true context switching approach, but only as a single user system, without proper memory protection. Windows NT added a proper kernel with memory protection, based on the VMS system, originally written for the DEC/Vax. Later versions of Windows NT and Windows 2000 (a security and kernel enhanced version of NT) allowed multiple logins also through a terminal server. Windows 2000 thus has a comparable functionality to UNIX in this respect Amiga DOS is an operating system for the

Commodore Amiga computer. It is based on the UNIX model and is a fully multi-tasking, single-user system. Several programs may run actively at any time. The operating system includes a window environment which means that each independent program has a screen of its own and does not therefore have to compete for the screen with other programs. This has been a major limitation on multi-tasking operating systems in the past.

Operating system (OS) is a program or set of programs, which acts as an interface between a user of the computer and the computer hardware. The main purpose of an OS is to provide an environment in which we can execute programs.

The main goals of the OS are: (i) to make the computer system convenient to use, and (ii) to make use of computer hardware in an efficient manner. Operating System is system software, which may be viewed as a collection of software consisting of procedures for operating the computer and for providing an environment for execution of programs. It's an interface between user and computer. So, OS does everything in a computer to work together smoothly and efficiently.

Basically, an OS has three main responsibilities: (a) to perform basic tasks, such as recognizing input from the keyboard, sending output to the display screen, keeping track of files and directories on the disk, and controlling peripheral devices namely disk drives and printers (b) to ensure that different programs and users running at the same time do not interfere with each other; and (c) to provide a software platform on top of which other programs can run. The OS is also responsible for security, ensuring

that unauthorized users do not access the system. The first two responsibilities address the need for managing the computer hardware and the application programs that use the hardware. The third responsibility focuses on providing an interface between application software and hardware so that application software can be efficiently developed. Since the OS is already responsible for managing the hardware, it should provide a programming interface for application developers. As a user, one normally interacts with the OS through a set of commands. The commands are accepted and executed by a part of the OS, called as the command processor or command line interpreter.

In order to understand operating systems, we must understand at the outset the computer hardware and the development of OS. Hardware means the physical machine and its electronic components, including memory chips, input/output devices, storage devices and the central processing unit. Software is a program written for the computer system. Main memory is where the data and instructions are stored for processing. Input/output devices are the peripherals attached to a system, such as keyboard, printers, disk drives, CD drives, magnetic tape drives, modem, monitor, etc. The central processing unit is the brain of a computer system; it has circuitry to control interpretation and execution of instructions. It controls the operation of entire computer system. All the storage references, data manipulations and I/O operations are performed by the CPU. The entire computer system can be divided into four parts or components: (1) the hardware, (2) the OS, (3) the application and system programs, and (4)

the users. Hardware provides the basic computing power. The system programs the way in which these resources are used to solve the computing problems of the users. There may be many different users trying to solve variety of problems. The OS controls and coordinates the use of the hardware among various users vis-a-vis the application programs.

We can view an OS as a resource allocator. A computer system has many resources, which are required to solve a computing problem. These resources are the CPU time, memory space, files storage space, input/output devices, etc. The OS acts as a manager of all the resources and allocates them to the specific programs and users as required to perform diffrent tasks. Since there can be many conflicting requests for the resources, the OS must decide which requests are to be allocated resources to operate the computer system fairly and efficiently.

An OS can also be viewed as a control program, for the various I/O devices and the users programs. A control program controls the execution of the user programs to prevent errors and improper use of the computer resources. It is especially concerned with the operation and control of I/O devices. As stated above, the fundamental goal of a computer system is to execute user programs and solve user problems. For this goal,computer hardware is devised. But a bare hardware is not easy to use and for this purpose application or system programs are developed. Various programs require some common operations, such as controlling/use of some input/output devices and the use of CPU time for execution. The common functions of

controlling and allocation of resources between different users and application programs is brought together into one piece of software called operating system. It is easy to define operating systems by what they do rather than what they are. The primary goal of the operating systems is convenience for the user to use the computer. Operating system makes it easier for user. A secondary goal is an efficient operation of the computer system. The large computer systems are very expensive, and it is desirable to make them as efficient as possible. Operating system thus makes an optimal use of computer resources. In order to understand what operating systems are and what they do, we have to study how they are developed. Operating systems and the computer architecture have a great influence on each other. To facilitate the use of the hardware, operating systems have been developed.

First, professional computer operators were used to operate the computer. The programmers no longer operated the machine. As soon as one job was finished, an operator could start the next one and if some errors crept in the program, the operator could take a dump of memory and registers, and from this the programmer had to debug the programs.

The second major solution to reduce the setup time was to batch together jobs of similar needs and run through the computer as a group. But there were still problems. For example, when a job stopped, the operator would have to notice it by observing the console, determining why the program stopped; would take a dump, if necessary and start with the next job. To overcome this idle time, automatic job

sequencing was introduced. But even with batching technique, the faster computers allowed expensive time lags between the CPU and the I/O devices. Eventually several factors helped improve the performance of CPU. First, the speed of I/O devices became faster. Second, to use more of the available storage area in these devices, records were blocked before they were retrieved. Third, to reduce the gap in speed between the I/O devices and the CPU, an interface called the control unit was placed between them to perform the function of buffering. A buffer is an interim storage area that works as the slow input device which reads a record; and the control unit places each character of the record into the buffer. When the buffer is full, the entire record is transmitted to the CPU. The process is just opposite to the output devices. Fourth, in addition to buffering, an early form of spooling was developed by moving off-line the operations of card reading, printing, etc. SPOOL is an acronym that stands for, Simultaneous Peripherals Operations On-Line. For example, incoming jobs would be transferred from the card decks to tape/disks off-line, and then they would be read into the CPU from the tape/disks at a speed much faster than the card reader.

Moreover, the range and extent of services provided by an OS depends on a number of factors. Among other things, the needs and characteristics of the target environmental that the OS is intended to support largely determine user-visible functions of an operating system. For example, an OS intended for program development in an interactive environment may have a quite different set of system calls and commands than the OS designed for run-time support of a car engine.

UNIX AND UNIX-LIKE OPERATING SYSTEMS

Unix was originally written in assembly language. Ken Thompson wrote B, mainly based on BCPL, based on his experience in the MULTICS project.

B was replaced by C, and Unix, rewritten in C, developed into a large, complex family of inter-related operating systems which have been influential in every modern operating system.

The *UNIX-like* family is a diverse group of operating systems, with several major sub-categories including System V, BSD, and Linux. The name "UNIX" is a trademark of The Open Group which licenses it for use with any operating system that has been shown to conform to their definitions. "UNIX-like" is commonly used to refer to the large set of operating systems which resemble the original UNIX.

Unix-like systems run on a wide variety of computer architectures. They are used heavily for servers in business, as well as workstations in academic and engineering environments.

Free UNIX variants, such as Linux and BSD, are popular in these areas. Four operating systems are certified by the The Open Group (holder of the Unix trademark) as Unix.

HP's HP-UX and IBM's AIX are both descendants of the original System V Unix and are designed to run only on their respective vendor's hardware. In contrast, Sun Microsystems's Solaris Operating System can run on multiple types of hardware, including x86 and Sparc servers, and PCs. Apple's OS X, a replacement for Apple's earlier (non-Unix) Mac OS, is a hybrid kernel-based BSD variant derived

from NeXTSTEP, Mach, and FreeBSD. Unix interoperability was sought by establishing the POSIX standard. The POSIX standard can be applied to any operating system, although it was originally created for various Unix variants.

OS X

OS X (formerly "Mac OS X") is a line of open core graphical operating systems developed, marketed, and sold by Apple Inc., the latest of which is pre-loaded on all currently shipping Macintosh computers. OS X is the successor to the original Mac OS, which had been Apple's primary operating system since 1984. Unlike its predecessor, OS X is a UNIX operating system built on technology that had been developed at NeXT through the second half of the 1980s and up until Apple purchased the company in early 1997. The operating system was first released in 1999 as Mac OS X Server 1.0, with a desktop-oriented version (Mac OS X v10.0 "Cheetah") following in March 2001. Since then, six more distinct "client" and "server" editions of OS X have been released, the most recent being OS X 10.8 "Mountain Lion", which was first made available on February 16, 2012 for developers, and was then released to the public on July 25, 2012. Releases of OS X are named after big cats.

GOOGLE CHROMIUM OS

Chromium is an operating system based on the Linux kernel and designed by Google. Since Chromium OS targets computer users who spend most of their time on the Internet, it is mainly a web browser with limited ability to run local applications, though it has a built-in file manager and

media player. Instead, it relies on Internet applications (or Web apps) used in the web browser to accomplish tasks such as word processing.

MICROSOFT WINDOWS

Microsoft Windows is a family of proprietary operating systems designed by Microsoft Corporation and primarily targeted to Intel architecture based computers, with an estimated 88.9 percent total usage share on Web connected computers. The newest version is Windows 8 for workstations and Windows Server 2012 for servers. Windows 7 recently overtook Windows XP as most used OS.

Microsoft Windows originated in 1985 as an operating environment running on top of MS-DOS, which was the standard operating system shipped on most Intel architecture personal computers at the time.

In 1995, Windows 95 was released which only used MS-DOS as a bootstrap. For backwards compatibility, Win9x could run real-mode MS-DOS and 16 bits Windows 3.x drivers.

Windows ME, released in 2000, was the last version in the Win9x family. Later versions have all been based on the Windows NT kernel. Current versions of Windows run on IA-32 and x86-64 microprocessors, although Windows 8 will support ARM architecture. In the past, Windows NT supported non-Intel architectures.

Server editions of Windows are widely used. In recent years, Microsoft has expended significant capital in an effort to promote the use of Windows as a server operating system.

However, Windows' usage on servers is not as widespread as on personal computers, as Windows competes against Linux and BSD for server market share.

OTHER

There have been many operating systems that were significant in their day but are no longer so, such as AmigaOS; OS/2 from IBM and Microsoft; Mac OS, the non-Unix precursor to Apple's Mac OS X; BeOS; XTS-300; RISC OS; MorphOS and FreeMint. Some are still used in niche markets and continue to be developed as minority platforms for enthusiast communities and specialist applications. OpenVMS formerly from DEC, is still under active development by Hewlett-Packard. Yet other operating systems are used almost exclusively in academia, for operating systems education or to do research on operating system concepts. A typical example of a system that fulfills both roles is MINIX, while for example Singularity is used purely for research.

Other operating systems have failed to win significant market share, but have introduced innovations that have influenced mainstream operating systems, not least Bell Labs' Plan 9.

EXTENDED MACHINE VIEW OF AN OPERATING SYSTEM

There arises a need to identify the system resources that must be managed by the OS, using the process viewpoint. Here we know role of the corresponding resource manager. We now ask the questions: How are these resource managers

activated and where do they reside? Does memory manager ever invoke the process scheduler? Does scheduler ever call upon the services of a memory manager? Is the concept of process only for the user or is it used by the OS also?

The OS provides many instructions in addition to the Bare Machine Instructions (A Bare machine is a machine without its software clothing, and it does not provide the environment which most programmers desired for). The instructions that form a part of aBare machine plus those provided by the OS constitute the instructions set of the extended machine. The OS kernel runs on the bare machine; and the user programs run on the extended machine. This means that the kernel of OS is written by using the instructions of Bare machine only; whereas the users can write their programs by making use of instructions provided by the extended machine. The OS kernel runs on the Bare machine; user programs run on the extended machine. This means that the kernel of OS is written by using the instructions of the bare machine only; whereas the users can write their programs by making use of instructions provided by the extended machine.

SCHEDULING ALGORITHMS

THE SCHEDULING BASICS

Process scheduling is one of the most important functions of an operating system that supports multiprogramming. Such a function is heavily dependent on queues. There are three types of queues that are used in process scheduling:

- Job Queue –This contains all processes that have been introduced into the system
- Ready Queue – It contains all the processes that are waiting for CPU time, and can be selected to run at any time
- Device Queue – It contains processes waiting on a certain device. Each device has its own queue for processes that need input/output from it.

SCHEDULING STATISTICS

To gauge the performance of a scheduling algorithm, several statistics are used.

- Turnaround Time = Time Job Finished - Time Job First Ready
- Average Turnaround Time = Average of Turnaround Time of all jobs
- Throughput = Number of Jobs Finished / Total CPU Time
- CPU Utilization = (Total CPU Time - Idle Time)/Total CPU Time

The general goals of scheduling are:

- To reduce average turnaround time.
- To increase the throughput
- To increase CPU Utilization

FIRST COME FIRST SERVE

This algorithm is self-explanatory. This is the simplest (but least efficient) algorithm used to schedule processes for execution.

The first process to arrive in the ready queue will be executed first. Since this is not a *preemptive* algorithm, an incoming process is not allowed to take over the CPU if the running process is not complete. Thus, the running process will continue until it is completed.

SHORTEST JOB FIRST

In this algorithm, if more than one process is ready for execution, the scheduler selects the process with the shortest "burst time" (*time to finish*) and assigns it to the CPU.

This is also a non-preemptive algorithm, so as soon as a process is assigned to the CPU, it will not be replaced until it is completed.

SHORTEST REMAINING TIME FIRST

This is similar to Shortest Job First, but this algorithm is preemptive.

If a new process becomes ready, the scheduler will check if its burst time is shorter than the remaining time of the currently running process, then the new process will "preempt" the current process. The current process will be returned to the ready queue.

MEMORY TRANSLATION

Each process treats the computer's memory as its own private playground. How can give each process the illusion that it can have reference addresses in memory as it wants, but not have them step on each other's toes?

The trick is by distinguishing between the virtual addresses – the addresses used in the process code and the

physical addresses – the actual addresses in memory. Each process is actually given a fraction of physical memory. The memory management unit translates the virtual address in the code to a physical address within the user's space, and the translation remains invisible to the process.

5

System Software Architecture

SYSTEM SOFTWARE

System software is computer software designed to operate the computer hardware and to provide a platform for running application software. The most basic types of system software are:

- The computer BIOS and device firmware, which provide basic functionality to operate and control the hardware connected to or built into the computer.
- The operating system (prominent examples being Microsoft Windows, Mac OS X and Linux), which allows the parts of a computer to work together by performing tasks like transferring data between memory and disks or rendering output onto a display device. It also provides a platform to run high-level system software and application software.

- Utility software, which helps to analyze, configure, optimize and maintain the computer.

In some publications, the term *system software* is also used to designate software development tools (like a compiler, linker or debugger). Computer purchasers seldom buy a computer primarily because of its system software (But purchasers of devices like mobile phones because of there system software, as is the case with the iPhone, as the system software of such devices is difficult for the end-user to modify). Rather, system software serves as a useful (even necessary) level of infrastructure code, generally built-in or pre-installed. In contrast to system software, software that allows users to do things like create text documents, play games, listen to music, or surf the web is called application software.

TYPES OF SYSTEM SOFTWARE PROGRAMMES

System software helps use the operating system and computer system. It includes diagnostic tools, compilers, servers, windowing systems, utilities, language translator, data communication programmes, database systems and more. The purpose of system software is to insulate the applications programmer as much as possible from the complexity and specific details of the particular computer being used, especially memory and other hardware features, and such accessory devices as communications, printers, readers, displays, keyboards, etc. Specific kinds of system software include:

- Loaders
- Linkers

- Utility software
- Desktop environment / Graphical user interface
- Shells
- BIOS
- Hypervisors
- Boot loaders
- Database Management Systems(SQL, NoSQL)

If system software is stored on non-volatile memory such as integrated circuits, it is usually termed firmware.

SYSTEMS ARCHITECTURE

A system architecture or systems architecture is the conceptual model that defines the structure, behaviour, and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structure of the system which comprises system components, the externally visible properties of those components, the relationships (e.g. the behaviour) between them, and provides a plan from which products can be procured, and systems developed, that will work together to implement the overall system. The language for architecture description is called the architecture description language (ADL).

OVERVIEW

There is no universally agreed definition of which aspects constitute a system architecture, and various organizations define it in different ways, including:

- The fundamental organization of a system, embodied in its components, their relationships to each other

and the environment, and the principles governing its design and evolution.

- The composite of the design architectures for products and their life cycle processes.
- A representation of a system in which there is a mapping of functionality onto hardware and software components, a mapping of the software architecture onto the hardware architecture, and human interaction with these components.
- An allocated arrangement of physical elements which provides the design solution for a consumer product or life-cycle process intended to satisfy the requirements of the functional architecture and the requirements baseline.
- An architecture is the most important, pervasive, top-level, strategic inventions, decisions, and their associated rationales about the overall structure (i.e., essential elements and their relationships) and associated characteristics and behaviour.
- A description of the design and contents of a computer system. If documented, it may include information such as a detailed inventory of current hardware, software and networking capabilities; a description of long-range plans and priorities for future purchases, and a plan for upgrading and/or replacing dated equipment and software.
- A formal description of a system, or a detailed plan of the system at component level to guide its implementation.

- The structure of components, their interrelationships, and the principles and guidelines governing their design and evolution over time.

A system architecture can best be thought of as a set of representations of an existing (or To Be Created) system. It is used to convey the informational content of the elements comprising a system, the relationships among those elements, and the rules governing those relationships. The architectural components and set of relationships between these components that an architecture describes may consist of hardware, software, documentation, facilities, manual procedures, or roles played by organizations or people. A system architecture is primarily concerned with the internal interfaces among the system's components or subsystems, and the interface between the system and its external environment, especially the user. (In the specific case of computer systems, this latter, special interface, is known as the computer human interface, *AKA* human computer interface, or CHI; formerly called the man-machine interface.) A system architecture can be contrasted with system architecture engineering, which is the method and discipline for effectively implementing the architecture of a system:

- It is a *method* because a sequence of steps is prescribed to produce or change the architecture of a system within a set of constraints.
- It is a *discipline* because a body of knowledge is used to inform practitioners as to the most effective way to architect the system within a set of constraints.

HISTORY

It is important to keep in mind that the modern systems architecture did not appear out of nowhere. Systems architecture depends heavily on practices and techniques which were developed over thousands of years in many other fields most importantly being, perhaps, civil architecture. Prior to the advent of digital computers, the electronics and other engineering disciplines used the term system as it is still commonly used today. However, with the arrival of digital computers and the development of software engineering as a separate discipline, it was often necessary to distinguish among engineered hardware artifacts, software artifacts, and the combined artifacts. A programmable hardware artifact, or computing machine, that lacks its software programme is impotent; even as a software artifact, or programme, is equally impotent unless it can be used to alter the sequential states of a suitable (hardware) machine. However, a hardware machine and its software programme can be designed to perform an almost illimitable number of abstract and physical tasks. Within the computer and software engineering disciplines (and, often, other engineering disciplines, such as communications), then, the term system came to be defined as containing all of the elements necessary (which generally includes both hardware and software) to perform a useful function.

Consequently, within these engineering disciplines, a system generally refers to a programmable hardware machine and its included programme. And a systems engineer is defined as one concerned with the complete device, both

hardware and software and, more particularly, all of the interfaces of the device, including that between hardware and software, and especially between the complete device and its user (the CHI). The hardware engineer deals (more or less) exclusively with the hardware device; the software engineer deals (more or less) exclusively with the software programme; and the systems engineer is responsible for seeing that the software programme is capable of properly running within the hardware device, and that the system composed of the two entities is capable of properly interacting with its external environment, especially the user, and performing its intended function. By analogy, then, a systems architecture makes use of elements of both software and hardware and is used to enable design of such a composite system. A good architecture may be viewed as a 'partitioning scheme,' or algorithm, which partitions all of the system's present and foreseeable requirements into a workable set of cleanly bounded subsystems with nothing left over. That is, it is a partitioning scheme which is exclusive, inclusive, and exhaustive.

A major purpose of the partitioning is to arrange the elements in the sub systems so that there is a minimum of communications needed among them. In both software and hardware, a good sub system tends to be seen to be a meaningful "object". Moreover, a good architecture provides for an easy mapping to the user's requirements and the validation tests of the user's requirements. Ideally, a mapping also exists from every least element to every requirement and test. A *robust architecture* is said to be one that exhibits an optimal degree of fault-tolerance, backward compatibility,

forward compatibility, extensibility, reliability, maintainability, availability, serviceability, usability, and such other quality attributes as necessary and/or desirable.

TYPES OF SYSTEMS ARCHITECTURES

Several types of systems architectures (underlain by the same fundamental principles) have been identified as follows:

- Collaborative Systems (such as the Internet, intelligent transportation systems, and joint air defense systems)
- Manufacturing Systems
- Social Systems
- Software and Information Technology Systems
- Strategic Systems Architecture

SYSTEMS ENGINEERING

Systems engineering is an interdisciplinary field of engineering that focuses on how complex engineering projects should be designed and managed over the life cycle of the project. Issues such as logistics, the coordination of different teams, and automatic control of machinery become more difficult when dealing with large, complex projects. Systems engineering deals with work-processes and tools to handle such projects, and it overlaps with both technical and human-centered disciplines such as control engineering, industrial engineering, organizational studies, and project management.

HISTORY

The term *systems engineering* can be traced back to Bell Telephone Laboratories in the 1940s. The need to identify

and manipulate the properties of a system as a whole, which in complex engineering projects may greatly differ from the sum of the parts' properties, motivated the Department of Defense, NASA, and other industries to apply the discipline. When it was no longer possible to rely on design evolution to improve upon a system and the existing tools were not sufficient to meet growing demands, new methods began to be developed that addressed the complexity directly. The evolution of systems engineering, which continues to this day, comprises the development and identification of new methods and modeling techniques. These methods aid in better comprehension of engineering systems as they grow more complex. Popular tools that are often used in the systems engineering context were developed during these times, including USL, UML, QFD, and IDEF0. In 1990, a professional society for systems engineering, the *National Council on Systems Engineering* (NCOSE), was founded by representatives from a number of U.S. corporations and organizations. NCOSE was created to address the need for improvements in systems engineering practices and education.

As a result of growing involvement from systems engineers outside of the U.S., the name of the organization was changed to the International Council on Systems Engineering (INCOSE) in 1995. Schools in several countries offer graduate programmes in systems engineering, and continuing education options are also available for practicing engineers.

CONCEPT

Systems engineering signifies both an approach and, more recently, a discipline in engineering. The aim of

education in systems engineering is to simply formalize the approach and in doing so, identify new methods and research opportunities similar to the way it occurs in other fields of engineering. As an approach, systems engineering is holistic and interdisciplinary in flavour.

ORIGINS AND TRADITIONAL SCOPE

The traditional scope of engineering embraces the design, development, production and operation of physical systems, and systems engineering, as originally conceived, falls within this scope. "Systems engineering", in this sense of the term, refers to the distinctive set of concepts, methodologies, organizational structures (and so on) that have been developed to meet the challenges of engineering functional physical systems of unprecedented complexity. The Apollo programme is a leading example of a systems engineering project.

The use of the term " system engineer " has evolved over time to embrace a wider, more holistic concept of "systems" and of engineering processes. This evolution of the definition has been a subject of ongoing controversy [9], and the term continues to be applied to both the narrower and broader scope.

HOLISTIC VIEW

Systems engineering focuses on analyzing and eliciting customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem, the system lifecycle.

Oliver *et al.* claim that the systems engineering process can be decomposed into

- a *Systems Engineering Technical Process*, and
- a *Systems Engineering Management Process*.

Within Oliver's model, the goal of the Management Process is to organize the technical effort in the lifecycle, while the Technical Process includes *assessing available information*, *defining effectiveness measures*, to *create a behaviour model*, *create a structure model*, *perform trade-off analysis*, and *create sequential build & test plan*. Depending on their application, although there are several models that are used in the industry, all of them aim to identify the relation between the various stages mentioned above and incorporate feedback. Examples of such models include the Waterfall model and the VEE model.

INTERDISCIPLINARY FIELD

System development often requires contribution from diverse technical disciplines. By providing a systems (holistic) view of the development effort, systems engineering helps mold all the technical contributors into a unified team effort, forming a structured development process that proceeds from concept to production to operation and, in some cases, to termination and disposal. This perspective is often replicated in educational programmes in that systems engineering courses are taught by faculty from other engineering departments which, in effect, helps create an interdisciplinary environment.

MANAGING COMPLEXITY

The need for systems engineering arose with the increase in complexity of systems and projects, in turn exponentially increasing the possibility of component friction, and therefore the reliability of the design. When speaking in this context, complexity incorporates not only engineering systems, but also the logical human organization of data. At the same time, a system can become more complex due to an increase in size as well as with an increase in the amount of data, variables, or the number of fields that are involved in the design. The International Space Station is an example of such a system. The development of smarter control algorithms, microprocessor design, and analysis of environmental systems also come within the purview of systems engineering. Systems engineering encourages the use of tools and methods to better comprehend and manage complexity in systems. Some examples of these tools can be seen here:

- *System model, Modeling, and Simulation*,
- System architecture,
- *Optimization*,
- *System dynamics*,
- *Systems analysis*,
- *Statistical analysis*,
- *Reliability analysis*, and
- *Decision making*

Taking an interdisciplinary approach to engineering systems is inherently complex since the behaviour of and interaction among system components is not always

immediately well defined or understood. Defining and characterizing such systems and subsystems and the interactions among them is one of the goals of systems engineering. In doing so, the gap that exists between informal requirements from users, operators, marketing organizations, and technical specifications is successfully bridged.

SCOPE

One way to understand the motivation behind systems engineering is to see it as a method, or practice, to identify and improve common rules that exist within a wide variety of systems. Keeping this in mind, the principles of systems engineering — holism, emergent behaviour, boundary, et al. — can be applied to any system, complex or otherwise, provided systems thinking is employed at all levels. Besides defense and aerospace, many information and technology based companies, software development firms, and industries in the field of electronics & communications require systems engineers as part of their team. An analysis by the INCOSE Systems Engineering center of excellence (SECOE) indicates that optimal effort spent on systems engineering is about 15-20% of the total project effort. At the same time, studies have shown that systems engineering essentially leads to reduction in costs among other benefits. However, no quantitative survey at a larger scale encompassing a wide variety of industries has been conducted until recently. Such studies are underway to determine the effectiveness and quantify the benefits of systems engineering. Systems engineering encourages the use of modeling and simulation to validate assumptions or theories on systems and the interactions within them.

Use of methods that allow early detection of possible failures, in safety engineering, are integrated into the design process. At the same time, decisions made at the beginning of a project whose consequences are not clearly understood can have enormous implications later in the life of a system, and it is the task of the modern systems engineer to explore these issues and make critical decisions. There is no method which guarantees that decisions made today will still be valid when a system goes into service years or decades after it is first conceived but there are techniques to support the process of systems engineering. Examples include the use of soft systems methodology, Jay Wright Forrester's System dynamics method and the Unified Modeling Language (UML), each of which are currently being explored, evaluated and developed to support the engineering decision making process.

EDUCATION

Education in systems engineering is often seen as an extension to the regular engineering courses, reflecting the industry attitude that engineering students need a foundational background in one of the traditional engineering disciplines (e.g. automotive engineering, mechanical engineering, industrial engineering, computer engineering, electrical engineering) plus practical, real-world experience in order to be effective as systems engineers. Undergraduate university programmes in systems engineering are rare. INCOSE maintains a continuously updated Directory of Systems Engineering Academic Programmes worldwide. As of 2006, there are about 75 institutions in United States

that offer 130 undergraduate and graduate programmes in systems engineering. Education in systems engineering can be taken as *SE-centric* or *Domain-centric*.

- *SE-centric* programmes treat systems engineering as a separate discipline and all the courses are taught focusing on systems engineering practice and techniques.
- *Domain-centric* programmes offer systems engineering as an option that can be exercised with another major field in engineering.

Both these patterns cater to educate the systems engineer who is able to oversee interdisciplinary projects with the depth required of a core-engineer.

SYSTEMS ENGINEERING TOPICS

Systems engineering tools are strategies, procedures, and techniques that aid in performing systems engineering on a project or product. The purpose of these tools vary from database management, graphical browsing, simulation, and reasoning, to document production, neutral import/export and more.

SYSTEM

There are many definitions of what a system is in the field of systems engineering. Below are a few authoritative definitions:

- ANSI/EIA-632-1999: "An aggregation of end products and enabling products to achieve a given purpose."
- IEEE Std 1220-1998: "A set or arrangement of

elements and processes that are related and whose behaviour satisfies customer/operational needs and provides for life cycle sustainment of the products."

- ISO/IEC 15288:2008: "A combination of interacting elements organized to achieve one or more stated purposes."
- NASA Systems Engineering Handbook: "(1) The combination of elements that function together to produce the capability to meet a need. The elements include all hardware, software, equipment, facilities, personnel, processes, and procedures needed for this purpose. (2) The end product (which performs operational functions) and enabling products (which provide life-cycle support services to the operational end products) that make up a system."
- INCOSE Systems Engineering Handbook: "homogeneous entity that exhibits predefined behaviour in the real world and is composed of heterogeneous parts that do not individually exhibit that behaviour and an integrated configuration of components and/or subsystems."
- INCOSE: "A system is a construct or collection of different elements that together produce results not obtainable by the elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results. The results include system level qualities, properties, characteristics, functions, behaviour and performance. The value added by the system as a

whole, beyond that contributed independently by the parts, is primarily created by the relationship among the parts; that is, how they are interconnected."

THE SYSTEMS ENGINEERING PROCESS

Depending on their application, tools are used for various stages of the systems engineering process:

USING MODELS

Models play important and diverse roles in systems engineering. A model can be defined in several ways, including:

- An abstraction of reality designed to answer specific questions about the real world
- An imitation, analogue, or representation of a real world process or structure; or
- A conceptual, mathematical, or physical tool to assist a decision maker.

Together, these definitions are broad enough to encompass physical engineering models used in the verification of a system design, as well as schematic models like a functional flow block diagram and mathematical (i.e., quantitative) models used in the trade study process. This section focuses on the last. The main reason for using mathematical models and diagrams in trade studies is to provide estimates of system effectiveness, performance or technical attributes, and cost from a set of known or estimable quantities. Typically, a collection of separate models is needed to provide all of these outcome variables.

The heart of any mathematical model is a set of meaningful quantitative relationships among its inputs and outputs. These relationships can be as simple as adding up constituent quantities to obtain a total, or as complex as a set of differential equations describing the trajectory of a spacecraft in a gravitational field. Ideally, the relationships express causality, not just correlation.

TOOLS FOR GRAPHIC REPRESENTATIONS

Initially, when the primary purpose of a systems engineer is to comprehend a complex problem, graphic representations of a system are used to communicate a system's functional and data requirements. Common graphical representations include:

- Functional Flow Block Diagram (FFBD)
- VisSim
- Data Flow Diagram (DFD)
- N2 (N-Squared) Chart
- IDEF0 Diagram
- UML Use case diagram
- UML Sequence diagram
- USL Function Maps and Type Maps.
- Enterprize Architecture frameworks, like TOGAF, MODAF, Zachman Frameworks etc.

A graphical representation relates the various subsystems or parts of a system through functions, data, or interfaces. Any or each of the above methods are used in an industry based on its requirements. For instance, the N2 chart may be used where interfaces between systems is important.

Part of the design phase is to create structural and behavioural models of the system. Once the requirements are understood, it is now the responsibility of a systems engineer to refine them, and to determine, along with other engineers, the best technology for a job. At this point starting with a trade study, systems engineering encourages the use of weighted choices to determine the best option. A decision matrix, or Pugh method, is one way (QFD is another) to make this choice while considering all criteria that are important. The trade study in turn informs the design which again affects the graphic representations of the system (without changing the requirements). In an SE process, this stage represents the iterative step that is carried out until a feasible solution is found. A decision matrix is often populated using techniques such as statistical analysis, reliability analysis, system dynamics (feedback control), and optimization methods. At times a systems engineer must assess the existence of feasible solutions, and rarely will customer inputs arrive at only one. Some customer requirements will produce no feasible solution. Constraints must be traded to find one or more feasible solutions.

The customers' wants become the most valuable input to such a trade and cannot be assumed. Those wants/ desires may only be discovered by the customer once the customer finds that he has overconstrained the problem. Most commonly, many feasible solutions can be found, and a sufficient set of constraints must be defined to produce an optimal solution. This situation is at times advantageous because one can present an opportunity to improve the design towards one or many ends, such as cost or schedule.

Various modeling methods can be used to solve the problem including constraints and a cost function. Systems Modeling Language (SysML), a modeling language used for systems engineering applications, supports the specification, analysis, design, verification and validation of a broad range of complex systems. Universal Systems Language (USL) is a systems oriented object modeling language with executable (computer independent) semantics for defining complex systems, including software.

RELATED FIELDS AND SUB-FIELDS

Many related fields may be considered tightly coupled to systems engineering. These areas have contributed to the development of systems engineering as a distinct entity.

COGNITIVE SYSTEMS ENGINEERING

Cognitive systems engineering (CSE) is a specific approach to the description and analysis of human-machine systems or sociotechnical systems. The three main themes of CSE are how humans cope with complexity, how work is accomplished by the use of artefacts, and how human-machine systems and socio-technical systems can be described as joint cognitive systems. CSE has since its beginning become a recognised scientific discipline, sometimes also referred to as Cognitive Engineering. The concept of a Joint Cognitive System (JCS) has in particular become widely used as a way of understanding how complex socio-technical systems can be described with varying degrees of resolution. The more than 20 years of experience with CSE has been described extensively.

CONFIGURATION MANAGEMENT

Like systems engineering, Configuration Management as practiced in the defence and aerospace industry is a broad systems-level practice. The field parallels the taskings of systems engineering; where systems engineering deals with requirements development, allocation to development items and verification, Configuration Management deals with requirements capture, traceability to the development item, and audit of development item to ensure that it has achieved the desired functionality that systems engineering and/or Test and Verification Engineering have proven out through objective testing.

CONTROL ENGINEERING

Control engineering and its design and implementation of control systems, used extensively in nearly every industry, is a large sub-field of systems engineering. The cruise control on an automobile and the guidance system for a ballistic missile are two examples. Control systems theory is an active field of applied mathematics involving the investigation of solution spaces and the development of new methods for the analysis of the control process.

INDUSTRIAL ENGINEERING

Industrial engineering is a branch of engineering that concerns the development, improvement, implementation and evaluation of integrated systems of people, money, knowledge, information, equipment, energy, material and process. Industrial engineering draws upon the principles and methods of engineering analysis and synthesis, as well

as mathematical, physical and social sciences together with the principles and methods of engineering analysis and design to specify, predict and evaluate the results to be obtained from such systems.

INTERFACE DESIGN

Interface design and its specification are concerned with assuring that the pieces of a system connect and inter-operate with other parts of the system and with external systems as necessary. Interface design also includes assuring that system interfaces be able to accept new features, including mechanical, electrical and logical interfaces, including reserved wires, plug-space, command codes and bits in communication protocols. This is known as extensibility. Human-Computer Interaction (HCI) or Human-Machine Interface (HMI) is another aspect of interface design, and is a critical aspect of modern systems engineering. Systems engineering principles are applied in the design of network protocols for local-area networks and wide-area networks.

MECHATRONIC ENGINEERING

Mechatronic engineering, like Systems engineering, is a multidisciplinary field of engineering that uses dynamical systems modeling to express tangible constructs. In that regards it is almost indistinguishable from Systems Engineering, but what sets it apart is the focus on smaller details rather than larger generalizations and relationships. As such, both fields are distinguished by the scope of their projects rather than the methodology of their practice.

OPERATIONS RESEARCH

Operations research supports systems engineering. The tools of operations research are used in systems analysis, decision making, and trade studies. Several schools teach SE courses within the operations research or industrial engineering department, highlighting the role systems engineering plays in complex projects. Operations research, briefly, is concerned with the optimization of a process under multiple constraints.

PERFORMANCE ENGINEERING

Performance engineering is the discipline of ensuring a system will meet the customer's expectations for performance throughout its life. Performance is usually defined as the speed with which a certain operation is executed or the capability of executing a number of such operations in a unit of time. Performance may be degraded when an operations queue to be executed is throttled when the capacity is of the system is limited. For example, the performance of a packet-switched network would be characterised by the end-to-end packet transit delay or the number of packets switched within an hour. The design of high-performance systems makes use of analytical or simulation modeling, whereas the delivery of high-performance implementation involves thorough performance testing. Performance engineering relies heavily on statistics, queueing theory and probability theory for its tools and processes.

PROGRAMME MANAGEMENT AND PROJECT MANAGEMENT

Programme management (or programme management) has many similarities with systems engineering, but has

broader-based origins than the engineering ones of systems engineering. Project management is also closely related to both programme management and systems engineering.

PROPOSAL ENGINEERING

Proposal engineering is the application of scientific and mathematical principles to design, construct, and operate a cost-effective proposal development system. Basically, proposal engineering uses the "systems engineering process" to create a cost effective proposal and increase the odds of a successful proposal.

RELIABILITY ENGINEERING

Reliability engineering is the discipline of ensuring a system will meet the customer's expectations for reliability throughout its life; i.e. it will not fail more frequently than expected. Reliability engineering applies to all aspects of the system. It is closely associated with maintainability, availability and logistics engineering. Reliability engineering is always a critical component of safety engineering, as in failure modes and effects analysis (FMEA) and hazard fault tree analysis, and of security engineering. Reliability engineering relies heavily onstatistics, probability theory and reliability theory for its tools and processes.

SAFETY ENGINEERING

The techniques of safety engineering may be applied by non-specialist engineers in designing complex systems to minimize the probability of safety-critical failures. The "System Safety Engineering" function helps to identify "safety

hazards" in emerging designs, and may assist with techniques to "mitigate" the effects of (potentially) hazardous conditions that cannot be designed out of systems.

SECURITY ENGINEERING

Security engineering can be viewed as an interdisciplinary field that integrates the community of practice for control systems design, reliability, safety and systems engineering. It may involve such sub-specialties as authentication of system users, system targets and others: people, objects and processes.

SOFTWARE ENGINEERING

From its beginnings, software engineering has helped shape modern systems engineering practice. The techniques used in the handling of complexes of large software-intensive systems has had a major effect on the shaping and reshaping of the tools, methods and processes of SE.

SYSTEMS DESIGN

Systems design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. One could see it as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture and systems engineering. If the broader topic of product development "blends the perspective of marketing, design, and manufacturing into a single approach to product development, then design is the act of taking the marketing information and creating the design of the product

to be manufactured. Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user. Until the 1990s systems design had a crucial and respected role in the data processing industry. In the 1990s standardization of hardware and software resulted in the ability to build modular systems. The increasing importance of software running on generic platforms has enhanced the discipline of software engineering. Object-oriented analysis and design methods are becoming the most widely used methods for computer systems design. The UML has become the standard language in object-oriented analysis and design. It is widely used for modeling software systems and is increasingly used for high designing non-software systems and organizations.

LOGICAL DESIGN

The logical design of a system pertains to an abstract representation of the data flows, inputs and outputs of the system. This is often conducted via modelling, using an over-abstract (and sometimes graphical) model of the actual system. In the context of systems design, modelling can undertake the following forms, including

PHYSICAL DESIGN

The physical design relates to the actual input and output processes of the system. This is laid down in terms of how data is input into a system, how it is verified/ authenticated, how it is processed, and how it is displayed as output. Physical design, in this context, does not refer to the tangible physical design of an information system.

To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc. It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc.

ALTERNATIVE DESIGN METHODOLOGIES

RAPID APPLICATION DEVELOPMENT (RAD)

Rapid application development (RAD) is a methodology in which a systems designer produces prototypes for an end-user. The end-user reviews the prototype, and offers feedback on its suitability. This process is repeated until the end-user is satisfied with the final system.

JOINT APPLICATION DESIGN (JAD)

Joint application design (JAD) is a methodology which evolved from RAD, in which a systems designer consults with a group consisting of the following parties:

- Executive sponsor
- Systems Designer
- Managers of the system

JAD involves a number of stages, in which the group collectively develops an agreed pattern for the design and implementation of the system.

TOPICS OF SYSTEMS DESIGN

- Requirements analysis - analyzes the needs of the end users or customers

- Benchmarking — is an effort to evaluate how current systems are used
- Systems architecture - creates a blueprint for the design with the necessary specifications for the hardware, software, people and data resources. In many cases, multiple architectures are evaluated before one is selected.
- Design — designers will produce one or more 'models' of what they see a system eventually looking like, with ideas from the analysis section either used or discarded. A document will be produced with a description of the system, but nothing is specific — they might say 'touchscreen' or 'GUI operating system', but not mention any specific brands;
- Computer programming and debugging in the software world, or detailed design in the consumer, enterprize or commercial world - specifies the final system components.
- System testing - evaluates the system's actual functionality in relation to expected or intended functionality, including all integration aspects.

SYSTEMS ARCHITECT

In systems engineering, the systems architect is the high-level designer of a system to be implemented. The systems architect establishes the basic structure of the system, defining the essential core design features and elements that provide the framework for all that follows, and are the hardest to change later. The systems architect provides the engineering view of the users' vision for what

the system needs to be and do, and the paths along which it must be able to evolve, and strives to maintain the integrity of that vision as it evolves during detailed design and implementation.

OVERVIEW

In systems engineering, the systems architect is responsible for:

- Interfacing with the user(s) and sponsor(s) and all other stakeholders in order to determine their (evolving) needs.
- Generating the highest level of system requirements, based on the user's needs and other constraints such as cost and schedule.
- Ensuring that this set of high level requirements is consistent, complete, correct, and operationally defined.
- Performing cost-benefit analyses to determine whether requirements are best met by manual, software, or hardware functions; making maximum use of commercial off-the-shelf or already developed components.
- Developing partitioning algorithms (and other processes) to allocate all present and foreseeable requirements into discrete partitions such that a minimum of communications is needed among partitions, and between the user and the system.
- Partitioning large systems into (successive layers of) subsystems and components each of which can be handled by a single engineer or team of engineers or subordinate architect.

- Interfacing with the design and implementation engineers, or subordinate architects, so that any problems arising during design or implementation can be resolved in accordance with the fundamental architectural concepts, and user needs and constraints.
- Ensuring that a maximally robust architecture is developed.
- Generating a set of acceptance test requirements, together with the designers, test engineers, and the user, which determine that all of the high level requirements have been met, especially for the computer-human-interface.
- Generating products such as sketches, models, an early user guide, and prototypes to keep the user and the engineers constantly up to date and in agreement on the system to be provided as it is evolving.
- Ensuring that all architectural products and products with architectural input are maintained in the most current state and never allowed to become obsolete.

MAIN TOPICS OF SYSTEMS ARCHITECT

Large systems architecture was developed as a way to handle systems too large for one person to conceive of, let alone design. Systems of this size are rapidly becoming the norm, so architectural approaches and architects are increasingly needed to solve the problems of large systems.

USERS AND SPONSORS

Engineers as a group do not have a reputation for understanding and responding to human needs comfortably or for developing humanly functional and aesthetically pleasing products. Architects *are* expected to understand human needs and develop humanly functional and aesthetically pleasing products. A good architect is a translator between the user/sponsor and the engineers— and even among just engineers of different specialities. A good architect is also the principal keeper of the user's vision of the end product— and of the process of deriving requirements from and implementing that vision. Determining what the users/sponsors actually need, rather than what they say they want, is not engineering. An architect does not follow an exact procedure. S/he communicates with users/sponsors in a highly interactive way— together they extract the true *requirements* necessary for the engineered system. The architect must remain constantly in communication with the end users. Therefore, the architect must be intimately familiar with the user's environment and problem. (The engineer need only be very knowledgeable of the potential engineering solution space.)

HIGH LEVEL REQUIREMENTS

The user/sponsor should view the architect as the user's representative and provide *all input through the architect.* Direct interaction with project engineers is generally discouraged as the chance of mutual misunderstanding is very high. The user requirements' specification should be a joint product of the user and architect: the user brings

his needs and wish list, the architect brings knowledge of what is likely to prove doable within cost and time constraints. When the user needs are translated into a set of high level requirements is also the best time to write the first version of the acceptance test, which should, thereafter, be religiously kept up to date with the requirements. That way, the user will be absolutely clear about what s/he is getting. It is also a safeguard against untestable requirements, misunderstandings, and requirements creep. The development of the first level of engineering requirements is not a purely analytical exercise and should also involve both the architect and engineer. If any compromises are to be made— to meet constraints like cost, schedule, power, or space, the architect must ensure that the final product and overall look and feel do not stray very far from the user's intent. The engineer should focus on developing a design that optimizes the constraints but ensures a workable and reliable product.

The architect is primarily concerned with the comfort and usability of the product; the engineer is primarily concerned with the producibility and utility of the product. The provision of needed services to the user is the true function of an engineered system. However, as systems become ever larger and more complex, and as their emphases move away from simple hardware and software components, the narrow application of traditional systems development principles is found to be insufficient— the application of the more general principles of systems, hardware, and software architecture to the design of (sub)systems is seen to be needed. An architecture is also a simplified model of the

finished end product— its primary function is to define the parts and their relationships to each other so that the whole can be seen to be a consistent, complete, and correct representation of what the user had in mind— especially for the computer-human-interface. It is also used to ensure that the parts fit together and relate in the desired way.

It is necessary to distinguish between the architecture of the user's world and the engineered systems architecture. The former represents and addresses problems and solutions in the *user's* world. It is principally captured in the computer-human-interfaces (CHI) of the engineered system. The engineered system represents the *engineering* solutions— how the *engineer* proposes to develop and/or select and combine the components of the technical infrastructure to support the CHI. In the absence of an experienced architect, there is an unfortunate tendency to confuse the two architectures. But— the engineer thinks in terms of hardware and software and the technical solution space, whereas the user may be thinking in terms of solving a problem of getting people from point A to point B in a reasonable amount of time and with a reasonable expenditure of energy, or of getting needed information to customers and staff. A systems architect is expected to combine knowledge of both the architecture of the user's world and of (all potentially useful) engineering systems architectures. The former is a joint activity with the user; the latter is a joint activity with the engineers. The product is a set of high level requirements reflecting the user's requirements which can be used by the engineers to develop systems design requirements. Because requirements evolve over the course of a project, especially

a long one, an architect is needed until the system is accepted by the user: the architect is the best insurance that all changes and interpretations made during the course of development do not compromise the user's viewpoint.

COST/BENEFIT ANALYSES

Most engineers are specialists. They know the applications of one field of engineering science intimately, apply their knowledge to practical situations— that is, solve real world problems, evaluate the cost/benefits of various solutions within their specialty, and ensure the correct operation of whatever they design. Architects are generalists. They are not expected to be experts in any one technology but are expected to be knowledgeable of many technologies and able to judge their applicability to specific situations. They also apply their knowledge to practical situations, but evaluate the cost/benefits of various solutions using different technologies, for example, hardware versus software versus manual, and assure that the system as a whole performs according to the user's expectations. Many commercial-off-the-shelf or already developed hardware and software components may be selected independently according to constraints such as cost, response, throughput, etc. In some cases, the architect can already assemble the end system unaided. Or, s/he may still need the help of a hardware or software engineer to select components and to design and build any special purpose function. The architects (or engineers) may also enlist the aid of specialists— in safety, security, communications, special purpose hardware, graphics, human factors, test and evaluation, quality control,

RMA, interface management, etc. An effective systems architectural team must have immediate access to specialists in critical specialties.,

PARTITIONING AND LAYERING

An architect planning a building works on the overall design, making sure it will be pleasing and useful to its inhabitants. While a single architect by himself may be enough to build a single-family house, many engineers may be needed, in addition, to solve the detailed problems that arise when a novel high-rise building is designed. If the job is large and complex enough, parts of the architecture may be designed as independent components. That is, if we are building a housing complex, we may have one architect for the complex, and one for each type of building, as part of an *architectural team.* Large automation systems also require an architect and much engineering talent. If the engineered system is large and complex enough, the systems architect may defer to a hardware architect and a software architect for parts of the job, although they all may be members of a joint architectural team. The architect should sub-allocate the system requirements to major components or subsystems that are within the scope of a single hardware or software engineer, or engineering manager and team. *But the architect must never be viewed as an engineering supervisor.* (If the item is sufficiently large and/or complex, the chief architect will sub-allocate portions to more specialized architects.) Ideally, each such component/subsystem is a sufficiently stand-alone object that it can be tested as a complete component, separate from the whole, using only a simple

testbed to supply simulated inputs and record outputs. That is, it is not necessary to know how an air traffic control system works in order to design and build a data management subsystem for it.

It is only necessary to know the constraints under which the subsystem will be expected to operate. A good architect ensures that the system, however complex, is built upon relatively simple and "clean" concepts for each (sub)system or layer and is easily understandable by everyone, especially the user, without special training. The architect will use a minimum of heuristics to ensure that each partition is well defined and clean of kludges, work-arounds, short-cuts, or confusing detail and exceptions. As user needs evolve, (once the system is fielded and in use), it is a lot easier subsequently to evolve a simple concept than one laden with exceptions, special cases, and lots of "fine print." *Layering* the architecture is important for keeping the architecture sufficiently simple at each *layer* so that it remains comprehensible to a single mind. As layers are ascended, whole systems at *lower layers* become simple *components* at the *higher layers,* and may disappear altogether at the *highest layers.*

ACCEPTANCE TEST

The acceptance test is a principal responsibility of the systems architect. It is the chief means by which the architect will prove to the user that the system is as originally planned and that all subordinate architects and engineers have met their objectives.

COMMUNICATIONS WITH USERS AND ENGINEERS

A building architect uses sketches, models, and drawings. An automation systems (or software or hardware) architect should use sketches, models, and prototypes to discuss different solutions and results with users, engineers, and other architects. An early, draft version of the user's manual is invaluable, especially in conjunction with a prototype. A set of (engineering) *requirements* as a sole, or even principal, means of communicating with the users is explicitly to be avoided. Nevertheless, it is important that a workable, well written *set of requirements,* or specification, be created which is understandable to the customer (so that they can properly sign off on it). But it must use precise and unambiguous language so that designers and other implementers are left in no doubt as to meanings or intentions. In particular, all requirements must be testable, and the initial draft of the test plan should be developed contemporaneously with the requirements. All stakeholders should sign off on the acceptance test descriptions, or equivalent, as the sole determinant of the satisfaction of the requirements, at the outset of the programme.

ENTERPRIZE SOFTWARE

Enterprize software, also known as enterprize application software (EAS), is software used in organizations, such as in a business or government, as opposed to software chosen by individuals (for example, retail software). Enterprize software is an integral part of a (Computer Based) Information System. Services provided by enterprize software are typically

business-oriented tools such as online shopping and online payment processing, interactive product catalogue, automated billing systems, security, content management, IT service management, customer relationship management, resource planning, business intelligence, HR management, manufacturing, application integration, and forms automation.

DEFINITIONS

While there is no single, widely accepted list of enterprize software characteristics, this section is intended to summarize definitions from multiple sources. Enterprize software describes a collection of computer programmes with common business applications, tools for modeling how the entire organization works, and development tools for building applications unique to the organization. The software is intended to solve an enterprize-wide problem (rather than a departmental problem) and often written using an Enterprize Software Architecture. Enterprize level software aims to improve the enterprize's productivity and efficiency by providing business logic support functionality. Capterra broadly defines enterprize software in the following manner:

- Targets any type of organization — corporations, partnerships, sole proprietorships, nonprofits, government agencies — but does not directly target consumers.
- Targets any industry.
- Targets both large and small organizations — from Fortune 500 to "mom and pop" businesses.

- Includes function-specific (Accounting, HR, Supply Chain, etc.) and industry-specific (Manufacturing, Retail, Healthcare, etc.) solutions.

Due to the cost of building or buying what is often non-free proprietary software, only large enterprizes attempt to implement such enterprize software that models the entire business enterprize and is the core IT system of governing the enterprize and the core of communication within the enterprize. As business enterprizes have similar departments and systems in common, enterprize software is often available as a suite of programmes that have attached enterprize development tools to customize the programmes to the specific enterprize. Generally, these tools are complex enterprize programming tools that require specialist capabilities. Thus, one often sees job listings for a programmer who is required to have specific knowledge of a particular set of enterprize tools, such as "must be an SAP developer". Characteristics of enterprize software are performance, scalability, and robustness. Enterprize software typically has interfaces to other enterprize software (for example LDAP to directory services) and is centrally managed (a single admin page for example).

ENTERPRIZE APPLICATION SOFTWARE

Enterprize application software is application software that performs business functions such as order processing, procurement, production scheduling, customer information management, and accounting. It is typically hosted on servers and provides simultaneous services to a large number of users, typically over a computer network. This is in contrast

to a single-user application that is executed on a user's personal computer and serves only one user at a time.

TYPES

- Enterprize software can be designed and implemented by an information technology (IT) group within a company.
- It may also be purchased from an independent enterprize software developer, that often installs and maintains the software for their customers. Installation, customization, and maintenance can also be outsourced to an IT consulting company.
- Another model is based on a concept called on-demand software, or Software as a Service (SaaS). The on-demand model of enterprize software is made possible through the widespread distribution of broadband access to the Internet. Software as a Service vendors maintain enterprize software on servers within their own company data center and then provide access to the software to their enterprize customers via the Internet.

Enterprize software is often categorized by the business function that it automates - such as accounting software or sales force automation software. Similarly for industries - for example, there are enterprize systems devised for the health care industry, or for manufacturing enterprizes.

DEVELOPERS

Major organizations in the enterprize software field include SAP, IBM, BMC Software, HP Software Division, Redwood

Software, UC4 Software, JBoss (Red Hat), Microsoft, Adobe Systems, Oracle Corporation, Inquest Technologies, Computer Associates, and ASG Software Solutions but there are thousands of competing vendors.

CRITICISM

The word *enterprize* can have various connotations. Sometimes the term is used merely as a synonym for *organization*, whether it be very large (e.g., a corporation with thousands of employees), very small (a sole proprietorship), or an intermediate size. Often the term is used only to refer to very large organizations, although it has become a corporate-speak buzzword and may be heard in other uses. Some enterprize software vendors using the latter definition develop highly complex products that are often overkill for smaller organizations, and the application of these can be a very frustrating task. Thus, sometimes "enterprize" might be used sarcastically to mean overly complex software. The adjective "enterprizey" is sometimes used to make this sarcasm explicit. In this usage, the term "enterprizey" is intended to go beyond the concern of "overkill for smaller organizations" to imply the software is overly complex even for large organizations and simpler solutions are available.

APPLICATION SOFTWARE

Application software, also known as an application or an "app", is computer software designed to help the user to perform singular or multiple related specific tasks. Examples include enterprize software, accounting software, office

suites, graphics software and media players. Many application programmes deal principally with documents. Application software is contrasted with system software and middleware, which manage and integrate a computer's capabilities, but typically do not directly apply them in the performance of tasks that benefit the user. A simple, if imperfect, analogy in the world of hardware would be the relationship of an electric light bulb (an application) to an electric power generation plant (a system). The power station merely generates electricity, not itself of any real use until harnessed to an application like the electric light that performs a service that benefits the user. Application software applies the power of a particular computing platform or system software to a particular purpose. Some apps such as Microsoft Office are available in versions for several different platforms; others have narrower requirements.

TERMINOLOGY

In information technology, an application is a computer programme designed to help people perform an activity. An application thus differs from an operating system (which runs a computer), a utility (which performs maintenance or general-purpose chores), and a programming language (with which computer programmes are created). Depending on the activity for which it was designed, an application can manipulate text, numbers, graphics, or a combination of these elements. Some application packages offer considerable computing power by focusing on a single task, such as word processing; others, called integrated software, offer somewhat less power but include several applications. User-written

software tailors systems to meet the user's specific needs. User-written software include spreadsheet templates, word processor macros, scientific simulations, graphics and animation scripts. Even email filters are a kind of user software. Users create this software themselves and often overlook how important it is.

The delineation between system software such as operating systems and application software is not exact, however, and is occasionally the object of controversy. For example, one of the key questions in the United States v. Microsoft antitrust trial was whether Microsoft's Internet Explorer web browser was part of its Windows operating system or a separable piece of application software. As another example, the GNU/Linux naming controversy is, in part, due to disagreement about the relationship between the Linux kernel and the operating systems built over this kernel. In some types of embedded systems, the application software and the operating system software may be indistinguishable to the user, as in the case of software used to control a VCR, DVD player or microwave oven. The above definitions may exclude some applications that may exist on some computers in large organizations. For an alternative definition of an app: *see Application Portfolio Management.*

APPLICATION SOFTWARE CLASSIFICATION

Application software falls into two general categories; horizontal applications and vertical applications. Horizontal Application are the most popular and its widely spread in departments or companies. Vertical Applications are designed

for a particular type of business or for specific division in a company. There are many types of application software:

- An *application suite* consists of multiple applications bundled together. They usually have related functions, features and user interfaces, and may be able to interact with each other, e.g. open each other's files. Business applications often come in suites, e.g. Microsoft Office, OpenOffice.org and iWork, which bundle together a word processor, a spreadsheet, etc.; but suites exist for other purposes, e.g. graphics or music.
- *Enterprize software* addresses the needs of organization processes and data flow, often in a large distributed environment. (Examples include financial systems, customer relationship management (CRM) systems and supply-chain management software). Note that Departmental Software is a sub-type of Enterprize Software with a focus on smaller organizations or groups within a large organization. (Examples include Travel Expense Management and IT Helpdesk)
- *Enterprize infrastructure software* provides common capabilities needed to support enterprize software systems. (Examples include databases, email servers, and systems for managing networks and security.)
- *Information worker software* addresses the needs of individuals to create and manage information, often for individual projects within a department, in contrast to enterprize management. Examples include time management, resource management, documentation

tools, analytical, and collaborative. Word processors, spreadsheets, email and blog clients, personal information system, and individual media editors may aid in multiple information worker tasks.

- *Content access software* is software used primarily to access content without editing, but may include software that allows for content editing. Such software addresses the needs of individuals and groups to consume digital entertainment and published digital content. (Examples include Media Players, Web Browsers, Help browsers and Games)
- *Educational software* is related to content access software, but has the content and/or features adapted for use in by educators or students. For example, it may deliver evaluations (tests), track progress through material, or include collaborative capabilities.
- *Simulation software* are computer software for simulation of physical or abstract systems for either research, training or entertainment purposes.
- *Media development software* addresses the needs of individuals who generate print and electronic media for others to consume, most often in a commercial or educational setting. This includes Graphic Art software, Desktop Publishing software, Multimedia Development software, HTML editors, Digital Animation editors, Digital Audio and Video composition, and many others.
- *Mobile applications* run on hand-held devices such as mobile phones, personal digital assistants and enterprize digital assistants : see mobile application development.

- *Product engineering software* is used in developing hardware and software products. This includes computer aided design (CAD), computer aided engineering (CAE), computer language editing and compiling tools, Integrated Development Environments, and Application Programmer Interfaces.
- A command-driven interface is one in which you type in commands to make the computer do something. You have to know the commands and what they do and they have to be typed correctly. DOS and Unix are examples of command-driven interfaces.
- A graphical user interface (GUI) is one in which you select command choices from various menus, buttons and icons using a mouse. It is a user-friendly interface. The Windows and Mac OS are both graphical user interfaces.

Applications can also be classified by computing platform.

INFORMATION WORKER SOFTWARE

-
  - o Enterprize Resource Planning
  - o Accounting software
  - o Task and Scheduling
  - o Field service management
- Data Management
  - o Contact Management
  - o Spreadsheet
  - o Personal Database

- Documentation
  - o Document Automation/Assembly
  - o Word Processing
  - o Desktop publishing software
  - o Diagramming Software
  - o Presentation software
  - o Analytical software
  - o Computer algebra systems
  - o Numerical computing
  - o List of numerical software
  - o Physics software
  - o Science software
  - o List of statistical software
  - o Neural network software
- Collaborative software
  - o E-mail
  - o Blog
- Reservation systems
- Financial Software
  - o Day trading software
  - o Banking systems
  - o Clearing systems
  - o arithmetic software

CONTENT ACCESS SOFTWARE

- Electronic media software
  - o Web browser
  - o Media Players
  - o Hybrid editor players

ENTERTAINMENT SOFTWARE

- Digital pets
- Screen savers
- Video Games
  - o Arcade games
  - o Emulators for console games
  - o Personal computer games
  - o Console games
- o Mobile games

EDUCATIONAL SOFTWARE

CLASSROOM MANAGEMENT

- Learning/Training Management Software
- Reference software
- Sales Readiness Software
- Survey Management

ENTERPRIZE INFRASTRUCTURE SOFTWARE

- Business workflow software
- Database management system (DBMS) software
- Digital asset management (DAM) software
- Document Management software
- Geographic Information System (GIS) software

SIMULATION SOFTWARE

- Computer simulators
  - o Scientific simulators
  - o Social simulators

- o Battlefield simulators
- o Emergency simulators
- o Vehicle simulators
  - Flight simulators
  - Driving simulators
- o Simulation games
  - Vehicle simulation games

MEDIA DEVELOPMENT SOFTWARE

- Image organizer
- Media content creating/editing
  - o 3D computer graphics software
  - o Animation software
  - o Graphic art software
  - o Image editing software
    - Raster graphics editor
    - Vector graphics editor
  - o Video editing software
  - o Sound editing software
    - Digital audio editor
  - o Music sequencer
    - Scorewriter
  - o Hypermedia editing software
    - Web Development Software

PRODUCT ENGINEERING SOFTWARE

- Hardware Engineering
  - o Computer-aided engineering

- o Computer-aided design (CAD)
- o Finite Element Analysis

- Software Engineering
  - o Computer Language Editor
  - o Compiler Software
  - o Integrated Development Environments
  - o Game creation software
  - o Debuggers
  - o Programme testing tools
  - o License manager

ACCOUNTING SOFTWARE

Accounting software is application software that records and processes accounting transactions within functional modules such as accounts payable, accounts receivable, payroll, and trial balance. It functions as an accounting information system. It may be developed in-house by the company or organization using it, may be purchased from a third party, or may be a combination of a third-party application software package with local modifications. It varies greatly in its complexity and cost. The market has been undergoing considerable consolidation since the mid 1990s, with many suppliers ceasing to trade or being bought by larger groups.

MODULES

Accounting software is typically composed of various modules, different sections dealing with particular areas of accounting. Among the most common are:

CORE MODULES

- Accounts receivable—where the company enters money received
- Accounts payable—where the company enters its bills and pays money it owes
- General ledger—the company's "books"
- Billing—where the company produces invoices to clients/customers
- Stock/Inventory—where the company keeps control of its inventory
- Purchase Order—where the company orders inventory
- Sales Order—where the company records customer orders for the supply of inventory
- Cash Book—where the company records collection and payment

NON CORE MODULES

- Debt Collection—where the company tracks attempts to collect overdue bills (sometimes part of accounts receivable)
- Electronic payment processing
- Expense—where employee business-related expenses are entered
- Inquiries—where the company looks up information on screen without any edits or additions
- Payroll—where the company tracks salary, wages, and related taxes
- Reports—where the company prints out data
- Timesheet—where professionals (such as attorneys

and consultants) record time worked so that it can be billed to clients

- Purchase Requisition—where requests for purchase orders are made, approved and tracked

(Different vendors will use different names for these modules)

IMPLEMENTATIONS

In many cases, implementation (i.e. the installation and configuration of the system at the client) can be a bigger consideration than the actual software chosen when it comes down to the total cost of ownership for the business. Most midmarket and larger applications are sold exclusively through resellers, developers and consultants. Those organizations generally pass on a license fee to the software vendor and then charge the client for installation, customization and support services. Clients can normally count on paying roughly 50-200% of the price of the software in implementation and consulting fees. Other organizations sell to, consult with and support clients directly, eliminating the reseller.

CATEGORIES

PERSONAL ACCOUNTING

Mainly for home users that use accounts payable type accounting transactions, managing budgets and simple account reconciliation at the inexpensive end of the market.

LOW END

At the low end of the business markets, inexpensive applications software allows most general business accounting functions to be performed. Suppliers frequently serve a single national market, while larger suppliers offer separate solutions in each national market. Many of the low end products are characterized by being "single-entry" products, as opposed to double-entry systems seen in many businesses. Some products have considerable functionality but are not considered GAAP or IFRS/FASB compliant. Some low-end systems do not have adequate security nor audit trails.

MID MARKET

The mid-market covers a wide range of business software that may be capable of serving the needs of multiple national accountancy standards and allow accounting in multiple currencies. In addition to general accounting functions, the software may include integrated or add-on management information systems, and may be oriented towards one or more markets, for example with integrated or add-on project accounting modules. Software applications in this market typically include the following features:

- Industry-standard robust databases
- Industry-standard reporting tools
- Tools for configuring or extending the application (eg an SDK, access to programme code.

HIGH END

The most complex and expensive business accounting software is frequently part of an extensive suite of software

often known as Enterprize resource planning or *ERP* software. These applications typically have a very long implementation period, often greater than six months. In many cases, these applications are simply a set of functions which require significant integration, configuration and customization to even begin to resemble an accounting system. The advantage of a high-end solution is that these systems are designed to support individual company specific processes, as they are highly customizable and can be tailored to exact business requirements. This usually comes at a significant cost in terms of money and implementation time.

VERTICAL MARKET

Some business accounting software is designed for specific business types. It will include features that are specific to that industry. The choice of whether to purchase an industry-specific application or a general-purpose application is often very difficult. Concerns over a custom-built application or one designed for a specific industry include:

- Smaller development team
- Increased risk of vendor business failing
- Reduced availability of support

This can be weighed up against:

- Less requirement for customization
- Reduced implementation costs
- Reduced end-user training time and costs

Some important types of vertical accounting software are:

- Banking
- Construction
- Medical
- Nonprofit
- Point of Sale (Retail)
- Daycare accounting (a.k.a. Child care management software)

HYBRID SOLUTIONS

As technology improves, software vendors have been able to offer increasingly advanced software at lower prices. This software is suitable for companies at multiple stages of growth. Many of the features of Mid Market and High End software (including advanced customization and extremely scalable databases) are required even by small businesses as they open multiple locations or grow in size. Additionally, with more and more companies expanding overseas or allowing workers to home office, many smaller clients have a need to connect multiple locations. Their options are to employ software-as-a-service or another application that offers them similar accessibility from multiple locations over the internet. Bob Frankston has noted that his VisiCalc wasn't an early accounting programme and that software that "overly tuned for such function (Javelin, Lotus Improv, etc.) completely failed."