DESIGNING AND IMPLEMENTING AN AUTOMATIC DIAGNOSTIC SERVER FOR TROUBLESHOOTING PROBLEMS IN END-SYSTEMS ON JUIT LAN

By

ASHMIT CHHABRA	071228
PRAVEEN KUMAR GUPTA	071264
PRINCE RAJVANSHI	071292
VIPUL GOEL	071338

Under the Supervision of
Brig.(Retd.) S. P. Ghrera – HOD (CSE)

Submitted in partial fulfillment
of the degree of
BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
WAKNAGHAT
SOLAN, HIMACHAL PRADESH
INDIA

2011

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT

SOLAN, HIMACHAL PRADESH

Date: 11 May 2011

CERTIFICATE

This is to certify that the work entitled "**Designing and Implementing an Automatic Diagnostic Server for Troubleshooting Problems on JUIT LAN**" submitted by Ashmit Chhabra, Praveen Kumar Gupta, Prince Rajvanshi and Vipul Goel in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering of Jaypee University of Information Technology has been carried out in my supervision. This work has not been submitted partially or wholly to any other university or institution for award of this or any other degree or diploma.

Project Supervisor

Brig.(Retd.) S.P. Ghrera

H.O.D (CSE & IT), JUIT

ACKNOWLEDGEMENT

This project has been an outcome of sustained and continual efforts on part of every group member. We take this opportunity to express our gratitude to the people who have been instrumental in the successful completion of this project.

We are deeply indebted to our project guide **Brig.(Retd) S.P. Ghrera** for help, stimulating suggestions and encouragement helped throughout the project. For his coherent guidance throughout the tenure of the project, we feel fortunate to be taught by him, who gave us his unwavering support.

Sincere thanks to senior lab technicians **Mr. Gupta** and **Mr. Rohit Sharma** for extensive cooperation.

Ashmit Chhabra	Praveen Kumar Gupta	Prince Rajvanshi	Vipul Goel
071228	071264	071292	071338

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1.1 Overview

- Our project deals with construction of a server implementing network problem diagnostic techniques and methodologies to troubleshoot the problems automatically.

- The Diagnostic techniques are executed using proper programming in order to keep an eye on the network activities and tracking the anomalies over the network.

- Our Software is not only a Network Analyzer; it also has the functionalities of both Network Analyzer and Network Debugger making it completely automated and efficient.

- Our Main Purpose is to make the end-user completely oblivious of the diagnosis mechanism and facilitate minimal user effort.

- The diagnosis is performed on an automated basis which requires timely execution of a well-efficient and optimized algorithmic approach.

- Proper tests are run in a timed manner to gauge the network performance and the subsequent results are automatically executed or with the end-user consent.

- Tests run on a computer machine functioning as a server running on Windows or Linux platform.

- The Problem faced on the network are corrected using a Graphical User Interface (GUI) to be used by the end-user or as a stand-alone application which will run whenever any problem arises over the network and automatically corrects it using the analysis and interpretation of various network diagnostic tools like ping, netstat, ipconfig and tracert tools.

- GUI is loaded with features like listing of all tests, automatic/manual test result execution, result analysis of tests etc.

- For the Implementation of this project, the following programming languages and softwares were used:

    - C/C++
    - JAVA
    - Network Programming
    - Windows Server 2000
    - Windows Server 2003
    - Linux OpenSuse Enterprise Server 14
    - Command Tools (ping, netstat, ipconfig and tracert)
    - Network Sniffer Tools
    - Eclipse

1.1.2 Research Challenges

Several studies have shown that the majority of network performance problems occur in or near the users' desktop/laptop computer. These problems include, but are not limited to, duplex mismatch conditions on Ethernet/Fast Ethernet links, incorrectly set TCP buffers in the user's computer, or problems with the local network infrastructure. Our Diagnostic tool is designed to quickly and easily identify a specific set of conditions that are known to impact network performance.

1.1.3 Thesis Structure

This Thesis Comprises of different sections covering the theoretical part of the technologies and tools used, along with the pictorial representation of the working structure wherever needed. The Source code of the software has also been put up in this scripture.

CHAPTER 2

METHODOLOGY

2.1 Overview of the System Architecture

The **client–server model** of computing is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.

Description

The *client–server* characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.

Functions such as email exchange, web access and database access, are built on the client–server model. Users accessing banking services from their computer use a web browser client to send a request to a web server at a bank. That program may in turn forward the request to its own database client program that sends a request to a database server at another bank computer to retrieve the account information. The balance is returned to the bank database client, which in turn serves it back to the web browser client displaying the results to the user. The client–server model has become one of the central ideas of network computing. Many business applications being written today use the client–server model. So do the Internet's main application protocols, such as HTTP, SMTP, Telnet, and DNS.

The interaction between client and server is often described using sequence diagrams. Sequence diagrams are standardized in the Unified Modeling Language.

Specific types of clients include web browsers, email clients, and online Chat clients.

Specific types of servers include web servers, ftp servers, application servers, database servers, name servers, mail servers, file servers, print servers, and terminal servers. Most web services are also types of servers.

Comparison to peer-to-peer architecture

Peer-to-peer networks involve two or more computers pooling individual resources such as disk drives, CD-ROMs and printers. These shared resources are available to every computer in the network. Each computer acts as both the client and the server which means all the computers on the network are equals, that is where the term peer-to-peer comes from. While a client-server network involves multiple clients connecting to a single, central server. The file server on a client-server network is a high capacity, high speed computer with a large hard disk capacity.

In the peer to peer network, software applications can be installed on the single computer and shared by every computer in the network. They are also cheaper to set up because most desktop operating systems have the software required for the network installed by default. On the other hand client-server model works with any size or physical layout of LAN and doesn't tend to slow down with a heavy use.

Peer-to-peer networks are typically less secure than client-server networks because security is handled by the individual computers, not on the network as a whole. The resources of the computers in the network can become overburdened as they have to support not only the workstation user, but also the requests from network users. It is also difficult to provide system wide services because the desktop operating system typically used in this type of network is incapable of hosting the service. Where the client-server networks have a higher initial setup cost. It is possible to set up a server on a desktop computer, but it is recommended that businesses invest in enterprise-class hardware and software. They also

require a greater level of expertise to configure and manage the server hardware and software.

Advantages

- In most cases, client–server architecture enables the roles and responsibilities of a computing system to be distributed among several independent computers that are known to each other only through a network. This creates an additional advantage to this architecture: greater ease of maintenance. For example, it is possible to replace, repair, upgrade, or even relocate a server while its clients remain both unaware and unaffected by that change.
- All data is stored on the servers, which generally have far greater security controls than most clients. Servers can better control access and resources, to guarantee that only those clients with the appropriate permissions may access and change data.
- Since data storage is centralized, updates to that data are far easier to administer in comparison to a P2P paradigm. In the latter, data updates may need to be distributed and applied to each peer in the network, which is time-consuming as there can be thousands or even millions of peers.
- Many mature client–server technologies are already available which were designed to ensure security, friendliness of the user interface, and ease of use.
- It functions with multiple different clients of different capabilities.

Disadvantages

- As the number of simultaneous client requests to a given server increases, the server can become overloaded. Contrast that to a P2P network, where its aggregated bandwidth actually increases as nodes are added, since the P2P network's overall bandwidth can be roughly computed as the sum of the bandwidths of every node in that network.
- The client–server paradigm lacks the robustness of a good P2P network. Under client–server, should a critical server fail, clients' requests cannot be fulfilled. In P2P

networks, resources are usually distributed among many nodes. Even if one or more nodes depart and abandon a downloading file, for example, the remaining nodes should still have the data needed to complete the download.

SERVER

In computing, the term **server** is used to refer to one of the following:

- A computer program running as a service, to serve the needs or requests of other programs (referred to in this context as "clients") which may or may not be running on the same computer.
- A physical computer dedicated to running one or more such services, to serve the needs of programs running on other computers on the same network.
- A software/hardware system (i.e. a software service running on a dedicated computer) such as a database server, file server, mail server, or print server.

In computer networking, a **server** is a program that operates as a socket listener. The term **server** is also often generalized to describe host that is deployed to execute one or more such programs.

A **server computer** is a computer, or series of computers, that link other computers or electronic devices together. They often provide essential services across a network, either to private users inside a large organization or to public users via the internet. For example, when you enter a query in a search engine, the query is sent from your computer over the internet to the servers that store all the relevant web pages. The results are sent back by the server to your computer.

Many servers have dedicated functionality such as web servers, print servers, and database servers. **Enterprise servers** are servers that are used in a business context.

Usage

Servers provide essential services across a network, either to private users inside a large organization or to public users via the Internet. For example, when you enter a query in a search engine, the query is sent from your computer over the internet to the servers that store all the relevant web pages. The results are sent back by the server to your computer.

The term *server* is used quite broadly in information technology. Despite the many server-branded products available (such as server versions of hardware, software or operating systems), in theory any computerized process that shares a resource to one or more client processes is a server. To illustrate this, take the common example of file sharing. While the existence of files on a machine does not classify it as a server, the mechanism which shares these files to clients by the operating system is the server.

Similarly, consider a web server application (such as the multiplatform "Apache HTTP Server"). This web server software can be *run* on any capable computer. For example, while laptop or personal computer is not typically known as a server, they can in these situations fulfill the role of one, and hence be labeled as one. It is in this case that the machine's purpose as a web server classifies it in general as a server.

In the hardware sense, the word *server* typically designates computer models intended for hosting software applications under the heavy demand of a network environment. In this client–server configuration one or more machines, either a computer or a computer appliance, share information with each other with one acting as a host for the other.

While nearly any personal computer is capable of acting as a network server, a dedicated server will contain features making it more suitable for production environments. These features may include a faster CPU, increased high-performance RAM, and typically more than one large hard drive. More obvious distinctions include marked redundancy in power supplies, network connections, and even the servers themselves.

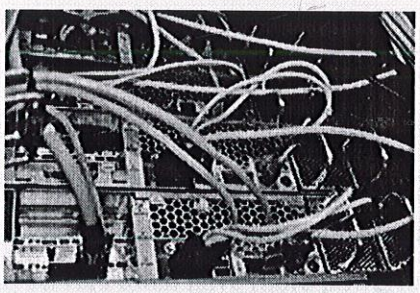
Between the 1990s and 2000s an increase in the use of *dedicated hardware* saw the advent of self-contained **server appliances**. One well-known product is the Google Search Appliance, a unit that combines hardware and software in an out-of-the-box packaging. Simpler examples of such appliances include switches, routers, gateways, and print server, all of which are available in a near plug-and-play configuration.

Modern operating systems such as Microsoft Windows or Linux distributions rightfully seem to be designed with client–server architecture in mind. These operating systems attempt to abstract hardware, allowing a wide variety of software to work with components of the computer. In a sense, the operating system can be seen as *serving* hardware to the software, which in all but low-level programming languages must interact using an API.

These operating systems may be able to run programs in the background called either services or daemons. Such programs may wait in a sleep state for their necessity to become apparent, such as the aforementioned *Apache HTTP Server* software. Since any software that provides services can be *called* a server, modern personal computers can be seen as a forest of servers and clients operating in parallel.

The Internet itself is also a forest of servers and clients. Merely requesting a web page from a few kilometers away involves satisfying a stack of protocols that involve many examples of hardware and software servers. The least of these are the routers, modems, domain name servers, and various other servers necessary to provide us the World Wide Web.

Server hardware



A Server Rack

Hardware requirements for servers vary, depending on the server application. Absolute CPU speed is not usually as critical to a server as it is to a desktop machine. Servers' duties to provide service to many users over a network lead to different requirements like fast network connections and high I/O throughput. Since servers are usually accessed over a network, they may run in headless mode without a monitor or input device. Processes that are not needed for the server's function are not used. Many servers do not have a graphical user

interface (GUI) as it is unnecessary and consumes resources that could be allocated elsewhere. Similarly, audio and USB interfaces may be omitted.

Servers often run for long periods without interruption and availability must often be very high, making hardware reliability and durability extremely important. Although servers can be built from commodity computer parts, mission-critical enterprise servers are ideally very fault tolerant and use specialized hardware with low failure rates in order to maximize uptime, for even a short-term failure can cost more than purchasing and installing the system. For example, it may take only a few minutes of down time at a national stock exchange to justify the expense of entirely replacing the system with something more reliable. Servers may incorporate faster, higher-capacity hard drives, larger computer fans or water cooling to help remove heat, and uninterruptible power supplies that ensure the servers continue to function in the event of a power failure. These components offer higher performance and reliability at a correspondingly higher price. Hardware redundancy—installing more than one instance of modules such as power supplies and hard disks arranged so that if one fails another is automatically available—is widely used. ECC memory devices that detect and correct errors are used; non-ECC memory is more likely to cause data corruption.

To increase reliability, most of the servers use memory with error detection and correction, redundant disks, redundant power supplies and so on. Such components are also frequently hot swappable, allowing replacing them on the running server without shutting it down. To prevent overheating, servers often have more powerful fans. As servers are usually administered by qualified engineers, their operating systems are also more tuned for stability and performance than for user friendliness and ease of use, Linux taking noticeably larger percentage than for desktop computers.

As servers need stable power supply, good Internet access, enhanced security and less noise, it is usual to store them in dedicated server centers or special rooms. This requires reducing power consumption as extra energy used generates more heat and the temperature in the room could exceed the acceptable limits. Normally server rooms are equipped with air conditioning devices. Server casings are usually flat and wide, adapted to store many devices

next to each other in server rack. Unlike ordinary computers, servers usually can be configured, powered up and down or rebooted remotely, using out-of-band management.

Server operating systems

Server-oriented operating systems tend to have certain features in common that make them more suitable for the server environment, such as

- GUI not available or optional
- ability to reconfigure and update both hardware and software to some extent without restart,
- advanced backup facilities to permit regular and frequent online backups of critical data,
- transparent data transfer between different volumes or devices,
- flexible and advanced networking capabilities,
- automation capabilities such as daemons in UNIX and services in Windows, and
- Tight system security, with advanced user, resource, data, and memory protection.

Server-oriented operating systems can, in many cases, interact with hardware sensors to detect conditions such as overheating, processor and disk failure, and consequently alert an operator or take remedial measures itself.

Because servers must supply a restricted range of services to perhaps many users while a desktop computer must carry out a wide range of functions required by its user, the requirements of an operating system for a server are different from those of a desktop machine. While it is possible for an operating system to make a machine both provide services and respond quickly to the requirements of a user, it is usual to use different operating systems on servers and desktop machines. Some operating systems are supplied in both server and desktop versions with similar user interface.

The desktop versions of the Windows and Mac OS X operating systems are deployed on a minority of servers, as are some proprietary mainframe operating systems, such as z/OS. The dominant operating systems among servers are UNIX-based or open source kernel distributions, such as Linux (the kernel).

The rise of the microprocessor-based server was facilitated by the development of UNIX to run on the x86 microprocessor architecture. The Microsoft Windows family of operating systems also runs on x86 hardware, and since Windows NT have been available in versions suitable for server use.

While the role of server and desktop operating systems remains distinct, improvements in the reliability of both hardware and operating systems have blurred the distinction between the two classes. Today, many desktop and server operating systems share similar code bases, differing mostly in configuration. The shift towards web applications and middleware platforms has also lessened the demand for specialist application servers.

Servers on the Internet

Almost the entire structure of the Internet is based upon a client–server model. High-level root name servers, DNS servers, and routers direct the traffic on the internet. There are millions of servers connected to the Internet, running continuously throughout the world.

- World Wide Web
- Domain Name System
- E-mail
- FTP file transfer
- Chat and instant messaging
- Voice communication
- Streaming audio and video
- Online gaming
- Database servers

Virtually every action taken by an ordinary Internet user requires one or more interactions with one or more servers.

There are also technologies that operate on an inter-server level. Other services do not use dedicated servers; for example peer-to-peer file sharing, some implementations of telephony (e.g. Skype), and supplying television programs to several users (e.g. Kontiki, SlingBox).

CLIENT

A **client** is an application or system that accesses a remote service on another computer system, known as a server, by way of a network. The term was first applied to devices that were not capable of running their own stand-alone programs, but could interact with remote computers via a network. These dumb terminals were clients of the time-sharing mainframe computer.

The client–server model is still used today. Client and server can run on the same machine and connect via UNIX domain sockets. Using Internet sockets a user may connect to a service operating on a possibly remote system through the Internet protocol suite. Servers set up *listening* sockets, and clients initiate connections that a server may *accept*. Web browsers are clients that connect to web servers and retrieve web pages for display. Most people use email clients to retrieve their email from their internet service provider's mail storage servers. Online chat uses a variety of clients, which vary depending on the chat protocol being used. Multiplayer online games may run as Game Clients on each local computer.

Increasingly, existing large client applications are being switched to websites, making the browser a sort of universal client. This avoids the hassle of downloading a large piece of software onto any computer you want to use the application on. An example of this is the rise of webmail.

In personal computers and computer workstations, the difference between client and server operating system is often just a matter of marketing - the server version may contain more operating system components, allow more simultaneous logins, and may be more expensive, while the client version may contain more end-user software.

Types

Clients are generally classified as either "fat clients", "thin clients", or "hybrid clients".

	Local storage	Local processing
Fat Client	Yes	Yes
Hybrid Client	No	Yes
Thin Client	No	No

Fat

A **fat-with low-fat client**, also known as a **rich-poor client** or **thick-thin client**, the personal computers or laptops can operate independently.

Programming languages and/or development tools for rich clients typically include Delphi, .NET Framework, Java and Visual Studio.

Thin

A **thin client** is a minimal sort of client. Thin **clients** use the resources of the host computer. A thin client's job is generally just to graphically display pictures provided by an application server, which performs the bulk of any required data processing. Programming environments for thin clients include JavaScript/AJAX (client side automation), ASP, JSP, Ruby on Rails, Python's Django, PHP and other (depends on server-side backend and uses HTML pages or rich media like Flash, Flex or Silverlight on client).

Hybrid

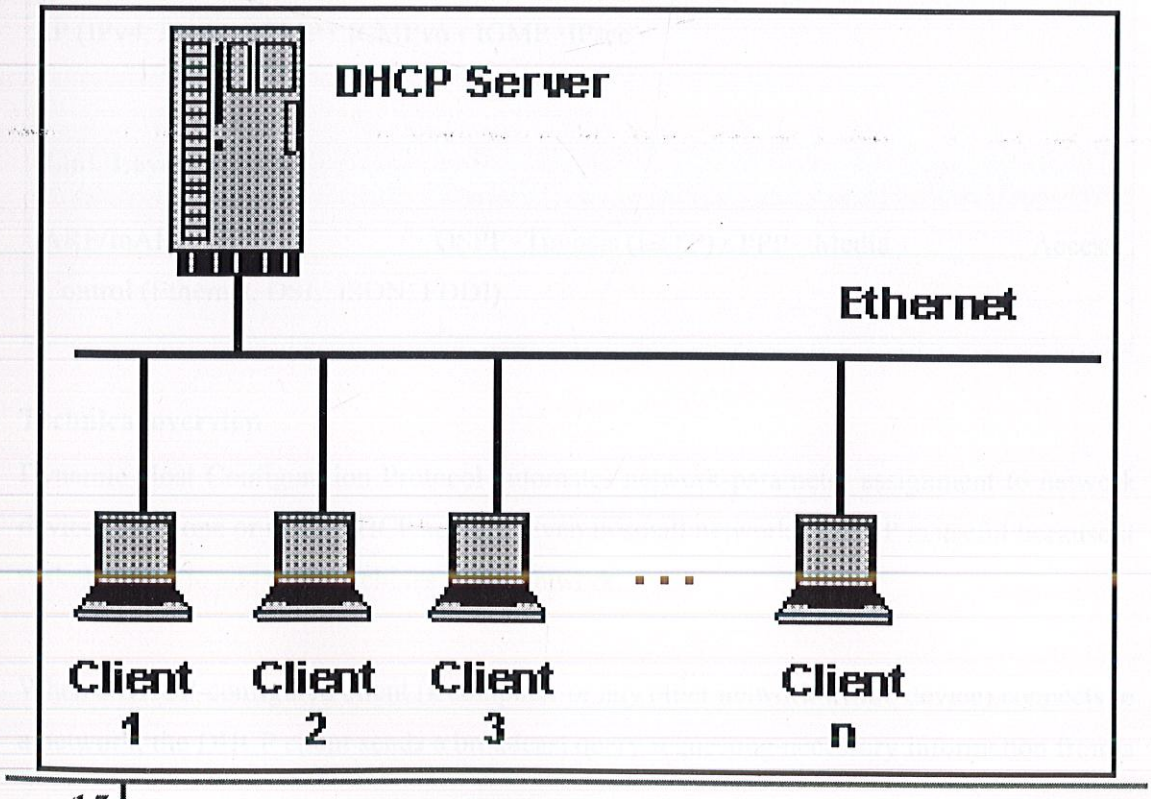
A **hybrid client** is a mixture of the above two client models. Similar to a fat client, it processes locally, but relies on the server for storage data. This approach offers features from both the fat client (multimedia support, high performance) and the thin client (high manageability, flexibility). The arrival of technologies such as Java allows hybrid clients the high performance required even for multimedia applications, with the data stored in the Cloud.

DHCP Server

The **Dynamic Host Configuration Protocol (DHCP)** is an automatic configuration protocol used on IP networks. Computers that are connected to IP networks must be configured before they can communicate with other computers on the network. DHCP allows a computer to be configured automatically, eliminating the need for intervention by a network administrator. It also provides a central database for keeping track of computers that have been connected to the network. This prevents two computers from accidentally being configured with the same IP address.

In the absence of DHCP, hosts may be manually configured with an IP address. Alternatively IPv6 hosts may use stateless address auto configuration to generate an IP address. IPv4 hosts may use link-local addressing to achieve limited local connectivity.

In addition to IP addresses, DHCP also provides other configuration information, particularly the IP addresses of local caching DNS resolvers. Hosts that do not use DHCP for address configuration may still use it to obtain other configuration information.



There are two versions of DHCP, one for IPv4 and one for IPv6. While both versions bear the same name and perform much the same purpose, the details of the protocol for IPv4 and IPv6 are sufficiently different that they can be considered separate protocols.

Internet Protocol Suite
Application Layer
BGP · DHCP · DNS · FTP · HTTP ·IMAP · IRC · LDAP · MGCP · NNTP ·NTP · POP · RIP · RPC · RTP · SIP ·SMTP · SNMP · SOCKS · SSH · Telnet · TLS/SSL · XMPP ·
Transport Layer
TCP · UDP · DCCP · SCTP · RSVP ·ECN ·
Internet Layer
IP (IPv4, IPv6) · ICMP · ICMPv6 · IGMP ·IPsec ·
Link Layer
ARP/InARP · NDP · OSPF ·Tunnels (L2TP) · PPP · Media Access Control (Ethernet, DSL, ISDN, FDDI)

Technical overview

Dynamic Host Configuration Protocol automates network-parameter assignment to network devices from one or more DHCP servers. Even in small networks, DHCP is useful because it makes it easy to add new machines to the network.

When a DHCP-configured client (a computer or any other network-aware device) connects to a network, the DHCP client sends a broadcast query requesting necessary information from a

DHCP server. The DHCP server manages a pool of IP addresses and information about client configuration parameters such as default gateway, domain name, the name servers, other servers such as time servers, and so forth. On receiving a valid request, the server assigns the computer an IP address, a lease (length of time the allocation is valid), and other IP configuration parameters, such as the subnet mask and the default gateway. The query is typically initiated immediately after booting, and must complete before the client can initiate IP-based communication with other hosts.

Depending on implementation, the DHCP server may have three methods of allocating IP-addresses:

- *Dynamic allocation*: A network administrator assigns a range of IP addresses to DHCP, and each client computer on the LAN is configured to request an IP address from the DHCP server during network initialization. The request-and-grant process uses a lease concept with a controllable time period, allowing the DHCP server to reclaim (and then reallocate) IP addresses that are not renewed.
- *Automatic allocation*: The DHCP server permanently assigns a free IP address to a requesting client from the range defined by the administrator. This is like dynamic allocation, but the DHCP server keeps a table of past IP address assignments, so that it can preferentially assign to a client the same IP address that the client previously had.
- *Static allocation*: The DHCP server allocates an IP address based on a table with MAC address/IP address pairs, which are manually filled in (perhaps by a network administrator). Only requesting clients with a MAC address listed in this table will be allocated an IP address. This feature (which is not supported by all DHCP servers) is variously called *Static DHCP Assignment* (by DD-WRT), *fixed-address* (by the dhcpd documentation), *Address Reservation* (by Netgear), *DHCP reservation* or *Static DHCP* (by Cisco/Linksys), and *IP reservation* or *MAC/IP binding* (by various other router manufacturers).

Technical details

DHCP uses the same two ports assigned by IANA for BOOTP: UDP port 67 for sending data to the server, and UDP port 68 for data to the client. DHCP communications are connectionless in nature.

DHCP operations fall into four basic phases: IP discovery, IP lease offer, IP request, and IP lease acknowledgement.

DHCP clients and servers on the same subnet communicate via UDP broadcasts. If the client and server are on different subnets, IP discovery and IP request messages are sent via UDP broadcasts, but IP lease offer and IP lease acknowledgement messages are unicast.

DHCP discovery

The client broadcasts messages on the physical subnet to discover available DHCP servers. Network administrators can configure a local router to forward DHCP packets to a DHCP server from a different subnet. This client-implementation creates a User Datagram Protocol (UDP) packet with the broadcast destination of 255.255.255.255 or the specific subnet broadcast address.

A DHCP client can also request its last-known IP address (in the example below, 192.168.1.100). If the client remains connected to a network for which this IP is valid, the server might grant the request. Otherwise, it depends whether the server is set up as authoritative or not. An authoritative server will deny the request, making the client ask for a new IP address immediately. A non-authoritative server simply ignores the request, leading to an implementation-dependent timeout for the client to give up on the request and ask for a new IP address.

DHCP request

A client can receive DHCP offers from multiple servers, but it will accept only one DHCP offer and broadcast a DHCP request message. Based on the Transaction ID field in the request, servers are informed whose offer the client has accepted. When other DHCP servers receive this message, they withdraw any offers that they might have made to the client and return the offered address to the pool of available addresses. The DHCP request message is broadcast, instead of being unicast to a particular DHCP server, because the DHCP client has

still not received an IP address. Also, this way one message can let all other DHCP servers know that another server will be supplying the IP address without missing any of the servers with a series of unicast messages.

DHCP acknowledgement

When the DHCP server receives the DHCPREQUEST message from the client, the configuration process enters its final phase. The acknowledgement phase involves sending a DHCPACK packet to the client. This packet includes the lease duration and any other configuration information that the client might have requested. At this point, the IP configuration process is completed.

The protocol expects the DHCP client to configure its network interface with the negotiated parameters.

After the client obtains an IP address, the client may use the Address Resolution Protocol (ARP) to prevent IP conflicts caused by overlapping address pools of DHCP servers.

DHCP information

A DHCP client may request more information than the server sent with the original DHCPOFFER. The client may also request repeat data for a particular application. For example, browsers use *DHCP Inform* to obtain web proxy settings via WPAD. Such queries do not cause the DHCP server to refresh the IP expiry time in its database.

DHCP releasing

The client sends a request to the DHCP server to release the DHCP information and the client deactivates its IP address. As client devices usually do not know when users may unplug them from the network, the protocol does not mandate the sending of *DHCP Release.*

Client configuration parameters in DHCP

A DHCP server can provide optional configuration parameters to the client. RFC 2132 describes the available DHCP options defined by Internet Assigned Numbers Authority (IANA) -DHCP and BOOTP PARAMETERS.

A DHCP client can select, manipulate and overwrite parameters provided by a DHCP server.

Options

An option exists to identify the vendor and functionality of a DHCP client. The information is a variable-length string of characters or octets which has a meaning specified by the vendor of the DHCP client. One method that a DHCP client can utilize to communicate to the server that it is using a certain type of hardware or firmware is to set a value in its DHCP requests called the Vendor Class Identifier (VCI) (Option 60). This method allows a DHCP server to differentiate between the two kinds of client machines and process the requests from the two types of modems appropriately. Some types of set-top boxes also set the VCI (Option 60) to inform the DHCP server about the hardware type and functionality of the device. The value that this option is set to gives the DHCP server a hint about any required extra information that this client needs in a DHCP response.

DHCP Relaying

In small networks, where only one IP subnet is being managed, DHCP clients communicate directly with DHCP servers. However, DHCP servers can also provide IP addresses for multiple subnets. In this case, a DHCP client that has not yet acquired an IP address cannot communicate directly with the DHCP server using IP routing, because it doesn't have a routable IP address, nor does it know the IP address of a router. In order to allow DHCP clients on subnets not directly served by DHCP servers to communicate with DHCP servers, DHCP relay agents can be installed on these subnets. The DHCP client broadcasts on the local link; the relay agent receives the broadcast and transmits it to one or more DHCP servers using unicast. The relay agent stores its own IP address in the GIADDR field of the DHCP packet. The DHCP server uses the GIADDR to determine the subnet on which the relay agent received the broadcast, and allocates an IP address on that subnet. When the DHCP server replies to the client, it sends the reply to the GIADDR address, again using unicast. The relay agent then retransmits the response on the local network.

Reliability

The DHCP protocol provides reliability in several ways: periodic renewal, rebinding, and failover. DHCP clients are allocated leases that last for some period of time. Clients begin to attempt to renew their leases once half the lease interval has expired. They do this by sending a unicast DHCPREQUEST message to the DHCP server that granted the original lease. If that server is down or unreachable, it will fail to respond to the DHCPREQUEST. However, the DHCPREQUEST will be repeated by the client from time to time, so when the DHCP server comes back up or becomes reachable again, the DHCP client will succeed in contacting it, and renew its lease.

If the DHCP server is unreachable for an extended period of time, the DHCP client will attempt to rebind, by broadcasting its DHCPREQUEST rather than unicasting it. Because it is broadcast, the DHCPREQUEST message will reach all available DHCP servers. If some other DHCP server is able to renew the lease, it will do so at this time.

In order for rebinding to work, when the client successfully contacts a backup DHCP server, that server must have accurate information about the client's binding. Maintaining accurate binding information between two servers is a complicated problem; if both servers are able to update the same lease database, there must be a mechanism to avoid conflicts between updates on the independent servers. A standard for implementing fault-tolerant DHCP servers was developed at the Internet Engineering Task Force.

If rebinding fails, the lease will eventually expire. When the lease expires, the client must stop using the IP address granted to it in its lease. At that time, it will restart the DHCP process from the beginning by broadcasting a DHCPDISCOVER message. Since its lease has expired, it will accept any IP address offered to it. Once it has a new IP address, presumably from a different DHCP server, it will once again be able to use the network. However, since its IP address has changed, any ongoing connections will be broken.

Security

The base DHCP protocol does not include any mechanism for authentication. Because of this, it is vulnerable to a variety of attacks. These attacks fall into three main categories:

- Unauthorized DHCP servers providing false information to clients.

- Unauthorized clients gaining access to resources.
- Resource exhaustion attacks from malicious DHCP clients.

Because the client has no way to validate the identity of a DHCP server, unauthorized DHCP servers can be operated on networks, providing incorrect information to DHCP clients. This can serve either as a denial-of-service attack, preventing the client from gaining access to network connectivity or as a man-in-the-middle attack. Because the DHCP server provides the DHCP client with server IP addresses, such as the IP address of one or more DNS servers, an attacker can convince a DHCP client to do its DNS lookups through its own DNS server, and can therefore provide its own answers to DNS queries from the client. This in turn allows the attacker to redirect network traffic through itself, allowing it to eavesdrop on connections between the clients and network servers it contacts, or to simply replace those network servers with its own.

Because the DHCP server has no secure mechanism for authenticating the client, clients can gain unauthorized access to IP addresses by presenting credentials, such as client identifiers, that belong to other DHCP clients. This also allows DHCP clients to exhaust the DHCP server's store of IP addresses—by presenting new credentials each time it asks for an address, the client can consume all the available IP addresses on a particular network link, preventing other DHCP clients from getting service.

DHCP does provide some mechanisms for mitigating these problems. The Relay Agent Information Option protocol extension (RFC 3046) allows network operators to attach tags to DHCP messages as these messages arrive on the network operator's trusted network. This tag is then used as an authorization token to control the client's access to network resources.

Because the client has no access to the network upstream of the relay agent, the lack of authentication does not prevent the DHCP server operator from relying on the authorization token.

Another extension, Authentication for DHCP Messages (RFC 3118), provides a mechanism for authenticating DHCP messages. Unfortunately RFC 3118 has not seen widespread adoption because of the problems of managing keys for large numbers of DHCP clients.

2.2 Summary

Deploying DHCP on your enterprise network provides the following benefits:

- **Safe and reliable configuration:** DHCP minimizes configuration errors caused by manual IP address configuration, such as typographical errors, as well as address conflicts caused by a currently assigned IP address accidentally being reissued to another computer.
- **Reduced network administration**
- TCP/IP configuration is centralized and automated.
- Network administrators can centrally define global and subnet-specific TCP/IP configurations.
- Clients can be automatically assigned a full range of additional TCP/IP configuration values by using DHCP options.
- Address changes for client configurations that must be updated frequently, such as remote access clients that move around constantly, can be made efficiently and automatically when the client restarts in its new location.
- Most routers can forward DHCP configuration requests, eliminating the requirement of setting up a DHCP server on every subnet, unless there is another reason to do so.

CHAPTER 3

COMMON NETWORK PROBLEMS

3.1.1 THE FIVE MOST COMMON NETWORK PROBLEMS:

1. Cable Problem: Cables that connect different parts of a network can be cut or shorted. A short can happen when the wire conductor comes in contact with another conductive surface, changing the path of the signal. Cable testers can be used to test for many types of cable problems such as: Cut cable, incorrect cable connections, Cable shorts, Interference level, Connector Problem

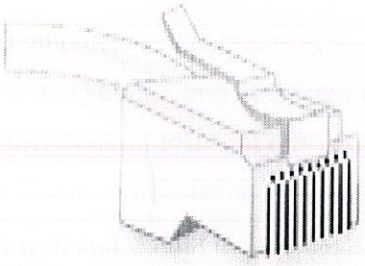
2. Connectivity Problem: A connectivity problem with one or more devices in a network can occur after a change is made in configuration or by a malfunction of a connectivity component, such as hub, a router or a Switch.

3. Excessive Network Collisions: These often lead to slow connectivity. The problem can occur as a result of bad network setup/plan, a user transferring a lot of information or jabbering network card.

   NB: A jabbering Network card is a network card that is stuck in a transmit mode. This will be evident because the transmit light will remain on constantly, indicating that the Network card is always transmitting.

4. Software Problem: Network problems can often be traced to software configuration such as DNS configuration, WINS configuration, the registry etc.

5. Duplicate IP Addressing: A common problem in many networking environments occurs when two machines try to use the same IP address. This can result in intermittent communications.



3.1.2 SOME OTHER NETWORK FLAWS INCLUDE:

- End-System Flaws (Server/Client)
    - OS may be missing TCP features.
    - Only Authorized Person can make changes in system.
    - Reduced Data rate.
    - Loss Rate excess.
- Path Flaws
    - Test shorter/limited path to narrow down search for problem.
    - Find any invisible infrastructure.
    - Problems caused due to path-mismatch and NIC (Network Interface Card) Problems and defects.
- Insufficient Capacity
    - Buffer Overflow on the network switches/routers.
    - Network needs to be re-balanced.
    - Usage of free/vacant ports can be a solution.
- Loss Rate
    - Cable and NIC Problems can be a primary reason.
- Tester flaws
    - It may include oversights or bugs.
    - Mainly, fault at the user-end.

3.2.1 End System Flaws:

These are flaws in the computer system that is acting as the test target (the web client) at one end of the path under test. They are best corrected by having a system administrator refer to the detailed tuning directions at **PSC's TCP tuning page** or the similar pages at **LBL**. Note that some operating systems may be missing required TCP features. Such systems cannot be expected to perform well and should to be upgraded or replaced.

In most organizations, the networking group is only responsible for the network as far as the connector on the wall. Generally they cannot (or will not) make changes to computer systems which are not theirs. Only the owner or a properly authorized system administrator should make changes to the end system.

3.2.2 Path flaws:

To further localize the flaws, test shorter subsections of this path or partial alternate paths by using additional testers and targets. Since there can be hidden switches and other invisible infrastructure, it is rarely effective to debug a network path without participation by the responsible network engineer. Unless you have access to the physical network and software configurations, you should not try to debug the path, except for a couple of specific checks:

- Excess loss rate and/or reduced data rate can symptoms of true network congestion caused by other traffic. If you have reason to suspect that the problems may be caused by other network traffic, (e.g. the test results vary from run to run) try re-running the tests early in the morning or other times when you expect the network to be lightly loaded. Although problems of insufficient capacity can sometimes be fixed simply by re-balancing the network (e.g. by moving some users to different network ports) they are often impossible to repair without the proper application of money. You may want to keep this in mind when you report the problem.

- If it was the loss rate test that failed and that you have access to the last few feet of cabling you may want to try swapping the cable, the Network Interface Card (NIC) or the entire end-system, to eliminate them as sources of loss.

- If the "duplex miss-match" test failed, the system administrator should check and possibly adjust the network duplex settings on the NIC. This may require the help of somebody who can check settings in the network switch.

If you have the access to the physical network and configurations, the easiest way to debug the path using a diagnostic server is to connect a portable diagnostic client to various places in the network, either by physically carrying a laptop to various wiring hubs or connecting it logically by reconfiguring vLANs.

3.2.3 Tester flaws:

Often tester flaws are not persistent, and will not be repeated on later runs of the same tests. If they do, flaws that seem to be related to this particular server (e.g. server bottlenecks) should be reported to the site contact for the server. Flaws that may indicate oversights or bugs in the tester itself (e.g. messages about unexpected events) should be reported.

We periodically retrieve results from public diagnostic servers and inspect the reports for accuracy. We pay particular attention to all reports indicating tester problems.

3.2.4 No path or target flaws:

If the target and path both pass all tests, you should be done, and if you are lucky, the application will work. If not, test the path with a traditional end-to-end diagnostic tool (e.g. **iperf**, **ttcp**, etc). If the traditional diagnostic test fails:

- Think carefully about any ignored warnings. Review the detailed help for each message.
- Think carefully about caveats in the detailed help for the pass messages.
- Report it. We are not a aware of any undocumented "false pass" results and would like to eliminate or document any that are found.

If the traditional end-to-end diagnostic test passes:

- The application itself may not be delay tolerant. It is very difficult to write an application with any sort of complicated internal controls that is completely delay tolerant. To do so requires that the application be structured such that all three elements (the sender, receiver and network) are fully overlapped, that is all three elements have to be busy at the same time. Furthermore, since the network delay is largely determined by the speed of light and the distance between the endpoints, the network has to carry a variable amount of data (depending on the distance) to match the performances of the sender and receiver. Systematically exploring this problem will be the subject of a future page on application diagnosis.

- In the short term the only method is to be aware of application reputations. Is this application known to work well over any long path? If the answer is no, and you fully pass both pathdiag and a classical end-to-end diagnostic, then the chances are that the application itself is flawed and would not function over a perfect network.

- If you received a warning about "Insufficient buffering (queue space)", it is possible that the traditional end-to-end diagnostic tool causes bursts that fit within the router or switch queue, but the end-to-end application causes larger bursts and overflows the queue. Although we suspect that such cases are common, they are very hard to observe in the field. Please try to collect a packet trace.

3.3 NETWORK CONGESTION

In data networking and queuing theory, **network congestion** occurs when a link or node is carrying so much data that its quality of service deteriorates. Typical effects include queuing delay, packet loss or the blocking of new connections. A consequence of these latter two is that incremental increases in offered load lead either only to small increase in network throughput, or to an actual reduction in network throughput.

Network protocols which use aggressive retransmissions to compensate for packet loss tend to keep systems in a state of network congestion even after the initial load has been reduced

to a level which would not normally have induced network congestion. Thus, networks using these protocols can exhibit two stable states under the same level of load. The stable state with low throughput is known as congestive collapse.

Modern networks use congestion control and network congestion avoidance techniques to try to avoid congestion collapse. These include: exponential back off in protocols such as802.11's CSMA/CA and the original Ethernet, window reduction in TCP, and fair queuing in devices such as routers. Another method to avoid the negative effects of network congestion is implementing priority schemes, so that some packets are transmitted with higher priority than others. Priority schemes do not solve network congestion by themselves, but they help to alleviate the effects of congestion for some services. An example of this is 802.1p. A third method to avoid network congestion is the explicit allocation of network resources to specific flows. One example of this is the use of Contention-Free Transmission Opportunities (CFTXOPs) in the ITU-T G.hn standard, which provides high-speed (up to 1 Gbit/s) Local area networking over existing home wires (power lines, phone lines and coaxial cables).

RFC 2914 addresses the subject of congestion control in detail.

Network capacity

The fundamental problem is that all network resources are limited, including router processing time and link throughput.

However:

- Today's (2010) Wireless LAN effective bandwidth throughput (15-100Mbit/s) is easily filled by a single personal computer.
- Even on fast computer networks (e.g. 1 Gbit), the backbone can easily be congested by a few servers and client PCs.
- Because P2P scales very well, file transmissions by P2P have no problem filling and will fill an uplink or some other network bottleneck, particularly when nearby peers are preferred over distant peers.

- Denial of service attacks by botnets are capable of filling even the largest Internet backbone network links (40 Gbit/s as of 2007), generating large-scale network congestion

Congestive collapse

Congestive collapse is a condition which a packet switched computer network can reach, when little or no useful communication is happening due to congestion. Congestion collapse generally occurs at choke points in the network, where the total incoming bandwidth to a node exceeds the outgoing bandwidth. Connection points between a local area network and a wide area network are the most likely choke points. A DSL modem is the most common small network example, with between 10 and 1000 Mbit/s of incoming bandwidth and at most 8 Mbit/s of outgoing bandwidth.

When a network is in such a condition, it has settled (under overload) into a stable state where traffic demand is high but little useful throughput is available, and there are high levels of packet delay and loss (caused by routers discarding packets because their output queues are too full) and general quality of service is extremely poor.

Cause

When more packets were sent than could be handled by intermediate routers, the intermediate routers discarded many packets, expecting the end points of the network to retransmit the information. However, early TCP implementations had very bad retransmission behavior. When this packet loss occurred, the end points sent *extra* packets that repeated the information lost; doubling the data rate sent, exactly the opposite of what should be done during congestion. This pushed the entire network into a 'congestion collapse' where most packets were lost and the resultant throughput was negligible.

Congestion control

Congestion control concerns controlling traffic entry into a telecommunications network, so as to avoid congestive collapse by attempting to avoid oversubscription of any of the

processing or link capabilities of the intermediate nodes and networks and taking resource reducing steps, such as reducing the rate of sending packets. It should not be confused with flow control, which prevents the sender from overwhelming the receiver.

Theory of congestion control

The modern theory of congestion control was pioneered by Frank Kelly, who applied microeconomic theory and convex optimization theory to describe how individuals controlling their own rates can interact to achieve an "optimal" network-wide rate allocation.

Examples of "optimal" rate allocation are max-min fair allocation and Kelly's suggestion of proportional fair allocation, although many others are possible.

The mathematical expression for optimal rate allocation is as follows. Let x_i be the rate of flow i, C_l be the capacity of link l, and r_{li} be 1 if flow i uses link l and 0 otherwise. Let x, c and R be the corresponding vectors and matrix. Let $U(x)$ be an increasing, strictly convex function, called the utility, which measures how much benefit a user obtains by transmitting at rate x. The optimal rate allocation then satisfies

$$\max_x \sum_i U(x_i)$$

such that $Rx \leq c$

The Lagrange dual of this problem decouples, so that each flow sets its own rate, based only on a "price" signaled by the network. Each link capacity imposes a constraint, which gives rise to a Lagrange multiplier, p_l. The sum of these Lagrange multipliers,

$$y_i = \sum_l p_l r_{li},$$

is the price to which the flow responds.

Congestion control then becomes a distributed optimization algorithm for solving the above problem. Many current congestion control algorithms can be modeled in this framework, with p_l being either the loss probability or the queuing delay at link l.

A major weakness of this model is that it assumes all flows observe the same price, while sliding window flow control causes "burstiness" which causes different flows to observe different loss or delay at a given link.

Classification of congestion control algorithms

There are many ways to classify congestion control algorithms:

- By the type and amount of feedback received from the network: Loss; delay; single-bit or multi-bit explicit signals
- By incremental deployability on the current Internet: Only sender needs modification; sender and receiver need modification; only router needs modification; sender, receiver and routers need modification.
- By the aspect of performance it aims to improve: high bandwidth-delay product networks; lossy links; fairness; advantage to short flows; variable-rate links
- By the fairness criterion it uses: max-min, proportional, "minimum potential delay".

Avoidance

The prevention of network congestion and collapse requires two major components:

1. A mechanism in routers to reorder or drop packets under overload,
2. End-to-end flow control mechanisms designed into the end points which respond to congestion and behave appropriately.

The correct end point behavior is usually still to repeat dropped information, but progressively slow the rate that information is repeated. Provided all end points do this, the congestion lifts and good use of the network occurs, and the end points all get a fair share of the available bandwidth. Other strategies such as slow-start ensure that new connections don't overwhelm the router before the congestion detection can kick in.

The most common router mechanisms used to prevent congestive collapses are fair queuing and other scheduling algorithms, and random early detection, or RED, where packets are randomly dropped proactively triggering the end points to slow transmission before congestion collapse actually occurs. Fair queuing is most useful in routers at choke points with a small number of connections passing through them. Larger routers must rely on RED.

Some end-to-end protocols are better behaved under congested conditions than others. TCP is perhaps the best behaved. The first TCP implementations to handle congestion well were developed in 1984, but it was not until Van Jacobson's inclusion of an open source solution in the Berkeley Standard Distribution UNIX ("BSD") in 1988 that good TCP implementations became widespread.

UDP does not, in itself, have any congestion control mechanism. Protocols built atop UDP must handle congestion in their own way. Protocols atop UDP which transmit at a fixed rate, independent of congestion, can be troublesome. Real-time streaming protocols, including much Voice over IP protocols, have this property. Thus, special measures, such as quality-of-service routing, must be taken to keep packets from being dropped from streams.

In general, congestion in pure datagram networks must be kept out at the periphery of the network, where the mechanisms described above can handle it. Congestion in the Internet backbone is very difficult to deal with. Fortunately, cheap fiber-optic lines have reduced costs in the Internet backbone. The backbone can thus be provisioned with enough bandwidth to keep congestion at the periphery.

Practical network congestion avoidance

Implementations of connection-oriented protocols, such as the widely-used TCP protocol, generally watch for packet errors, losses, or delays (see Quality of Service) in order to adjust the transmit speed. There are many different network congestion avoidance processes, since there are a number of different trade-offs available.

TCP/IP congestion avoidance

The TCP congestion avoidance algorithm is the primary basis for congestion control in the Internet.

Problems occur when many concurrent TCP flows are experiencing port queue buffer tail-drops. Then TCP's automatic congestion avoidance is not enough. All flows that experience port queue buffer tail-drop will begin a TCP retrain at the same moment - this is called TCP global synchronization.

Active Queue Management (AQM)

Purpose

"Recommendations on Queue Management and Congestion Avoidance in the Internet" (RFC 2309) states that:

- Fewer packets will be dropped with Active Queue Management (AQM).
- The link utilization will increase because less TCP global synchronization will occur.
- By keeping the average queue size small, queue management will reduce the delays and jitter seen by flows.
- The connection bandwidth will be more equally shared among connection oriented flows, even without flow-based RED or WRED.

Random early detection

One solution is to use random early detection (RED) on network equipments port queue buffer. On network equipment ports with more than one queue buffer, weighted random early detection (WRED) could be used if available.

RED indirectly signals to sender and receiver by deleting some packets, e.g. when the average queue buffer lengths are more than e.g. 50% (lower threshold) filled and deletes linearly more or (better according to paper) cubical more packets, up to e.g. 100% (higher threshold). The average queue buffer lengths are computed over 1 second at a time.

Flowbased-RED/WRED

Some network equipment are equipped with ports that can follow and measure each flow (**flowbased-RED/WRED**) and are hereby able to signal to a too big bandwidth flow according to some QoS policy. A policy could divide the bandwidth among all flows by some criteria.

IP ECN

Another approach is to use IP ECN. ECN is only used when the two hosts signal that they want to use it. With this method, an ECN bit is used to signal that there is explicit congestion. This is better than the indirect packet delete congestion notification performed by the RED/WRED algorithms, but it requires explicit support by both hosts to be effective. Some outdated or buggy network equipment drops packets with the ECN bit set, rather than ignoring the bit. More information on the status of ECN including the version required for Cisco IOS, by Sally Floyd, one of the authors of ECN.

When a router receives a packet marked as ECN capable and anticipates (using RED) congestion, it will set an ECN-flag notifying the sender of congestion. The sender then ought to decrease its transmission bandwidth; e.g. by decreasing the tcp window size (sending rate) or by other means.

Cisco AQM: Dynamic buffer limiting (DBL)

Cisco has taken a step further in their Catalyst 4000 series with engine IV and V. Engine IV and V has the possibility to classify all flows in "aggressive" (bad) and "adaptive" (good). It ensures that no flows fill the port queues for a long time. **DBL** can utilize **IP ECN** instead of packet-delete-signaling.

TCP Window Shaping

Congestion avoidance can also efficiently be achieved by reducing the amount of traffic flowing into a network. When an application requests a large file, graphic or web page, it usually advertises a "window" of between 32K and 64K. This results in the server sending a full window of data (assuming the file is larger than the window). When there are many applications simultaneously requesting downloads, this data creates a congestion point at an

upstream provider by flooding the queue much faster than it can be emptied. By using a device to reduce the window advertisement, the remote servers will send less data, thus reducing the congestion and allowing traffic to flow more freely. This technique can reduce congestion in a network by a factor of 40.

Side effects of congestive collapse avoidance

Radio links

The protocols that avoid congestive collapse are often based on the idea that data loss on the Internet is caused by congestion. This is true in nearly all cases; errors during transmission are rare on today's fiber based Internet. However, this causes WiFi, 3G or other networks with a radio layer to have poor throughput in some cases since wireless networks are susceptible to data loss due to interference. The TCP connections running over a radio based physical layer see the data loss and tend to believe that congestion is occurring when it isn't and erroneously reduce the data rate sent.

Short-lived connections

The slow-start protocol performs badly for short-lived connections. Older web browsers would create many consecutive short-lived connections to the web server, and would open and close the connection for each file requested. This kept most connections in the slow start mode, which resulted in poor response time.

To avoid this problem, modern browsers either open multiple connections simultaneously or reuse one connection for all files requested from a particular web server. However, the initial performance can be poor, and many connections never get out of the slow-start regime, significantly increasing latency.

CHAPTER 4

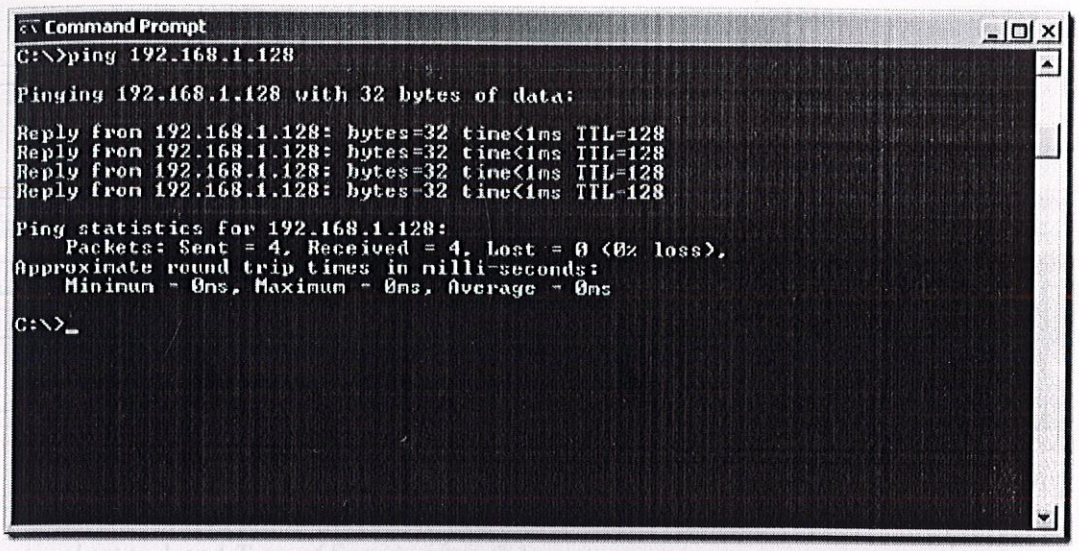
NETWORK TOOLS USED

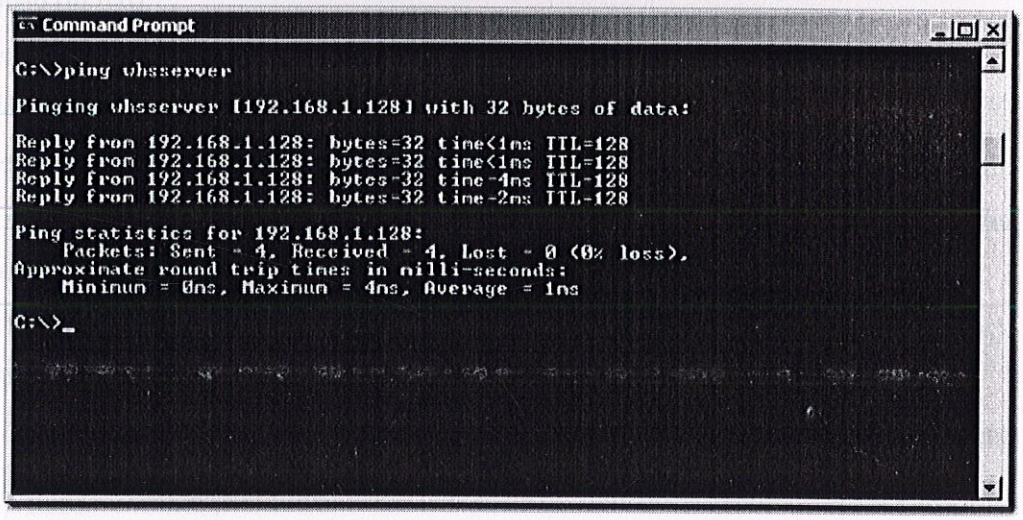
4.1.1 PING

Ping is a computer network administration utility used to test the reachability of a host on an Internet Protocol (IP) network and to measure the round-trip time for messages sent from the originating host to a destination computer. The name comes from active sonar terminology.

Ping operates by sending Internet Control Message Protocol (ICMP) *echo request* packets to the target host and waiting for an ICMP response. In the process it measures the time from transmission to reception (*round-trip time*) and records any packet loss. The results of the test are printed in form of a statistical summary of the response packets received, including the minimum, maximum, and the mean round-trip times, and sometimes the standard deviation of the mean.

Ping may be run using various options (command line switches) depending on the implementation that enable special operational modes, such as to specify the packet size used as the probe, automatic repeated operation for sending a specified count of probes, time stamping options, or to perform a ping flood. Flood pinging may be abused as a simple form of denial-of-service attack, in which the attacker overwhelms the victim with ICMP echo request packets.





ICMP packet

ICMP packet

	Bit 0 - 7	Bit 8 - 15	Bit 16 - 23	Bit 24 - 31
IP Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags* and *offset*	

	Time To Live (TTL)	Protocol	Checksum
	Source IP address		
	Destination IP address		
ICMP Payload (8+ bytes)	Type of message	Code	Checksum
	Quench		
	Data (*optional*)		

Generic composition of an ICMP packet Header (in blue):

Protocol set to 1 and *Type of Service* set to 0.

Payload (in red):

Type of ICMP message (8 bits)

Code (8 bits)

Checksum (16 bits), calculated with the ICMP part of the packet (the header is not used).

It is the 16-bit one's complement of the one's complement sum of the ICMP message starting with the Type field.

The ICMP 'Quench' (32 bits) field, which in this case (ICMP echo request and replies), will be composed of identifier (16 bits) and sequence number (16 bits).

Data load for the different kind of answers (Can be an arbitrary length, left to implementation detail. However must be less than the maximum MTU of the network.

Data Transportation

Message format

Echo request

The *echo request* is an ICMP message whose data is expected to be received back in an *echo reply* ("ping"). The host must respond to all echo requests with an echo reply containing the exact data received in the request message.

0 0	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8	0 9	1 0	1 1	1 2	1 3	1 4	1 5	1 6	1 7	1 8	1 9	2 0	2 1	2 2	2 3	2 4	2 5	2 6	2 7	2 8	2 9	3 0	3 1
Type = 8								Code = 0								Header Checksum															
Identifier																Sequence Number															
Data :::																															

The Identifier and Sequence Number can be used by the client to match the reply with the request that caused the reply. In practice, most Linux systems use a unique identifier for every ping process, and sequence number is an increasing number within that process. Windows uses a fixed identifier, which varies between Windows versions, and a sequence number that is only reset at boot time.

The data received by the Echo Request must be entirely included in the Echo Reply.

Echo reply

The *echo reply* is an ICMP message generated in response to an echo request, and is mandatory for all hosts and routers.

0 0	0 1	0 2	0 3	0 4	0 5	0 6	0 7	0 8	0 9	1 0	1 1	1 2	1 3	1 4	1 5	1 6	1 7	1 8	1 9	2 0	2 1	2 2	2 3	2 4	2 5	2 6	2 7	2 8	2 9	3 0	3 1
Type = 0								Code = 0								Header Checksum															
Identifier																Sequence Number															
Data :::																															

Type and *code* must be set to 0.

The *identifier* and *sequence number* can be used by the client to determine which echo requests are associated with the echo replies.

The data received in the echo request must be entirely included in the echo reply.

Other replies

In case of error, destination host or intermediate router will send back an ICMP error message, i.e. host unreachable or TTL exceeded in transit.

In addition these messages include the first 8 bytes of original message (in this case header of ICMP echo request, including quench value), so ping utility can match it to originating query.

Payload

The payload of the packet is generally filled with ASCII characters.

The payload includes a timestamp of when the message was sent, as well a sequence number. This allows ping to compute the round trip time in a stateless manner without needing to record when packets were sent. In cases of no answer and no error message, most implementations of ping display nothing, or periodically print notifications about timing out.

The term *ping* is commonly used to describe the transmission of an effectively content-less message that is used for a variety of purposes. For example, a ping may be sent using the User Datagram Protocol (UDP) to a device located behind a network address translator (NAT) to keep the port binding on the NAT translator from timing out and removing the forward mapping. Other examples are short or empty instant messages, emails, voice mails, or missed-call notification to indicate availability.

In gaming

In various network multi-player games, the server notes the time it requires for a game packet to reach a client and a response to be received. This round-trip time is usually reported as the player's 'ping'. It is used as an effective measurement of the player's latency, with lower ping times being desirable. Note that this style of ping typically does not use ICMP packets.

4.1.2 TRACERT

Traceroute is a computer network diagnostic tool for displaying the route (path) and measuring transit delays of packets across an Internet Protocol (IP) network. Traceroute

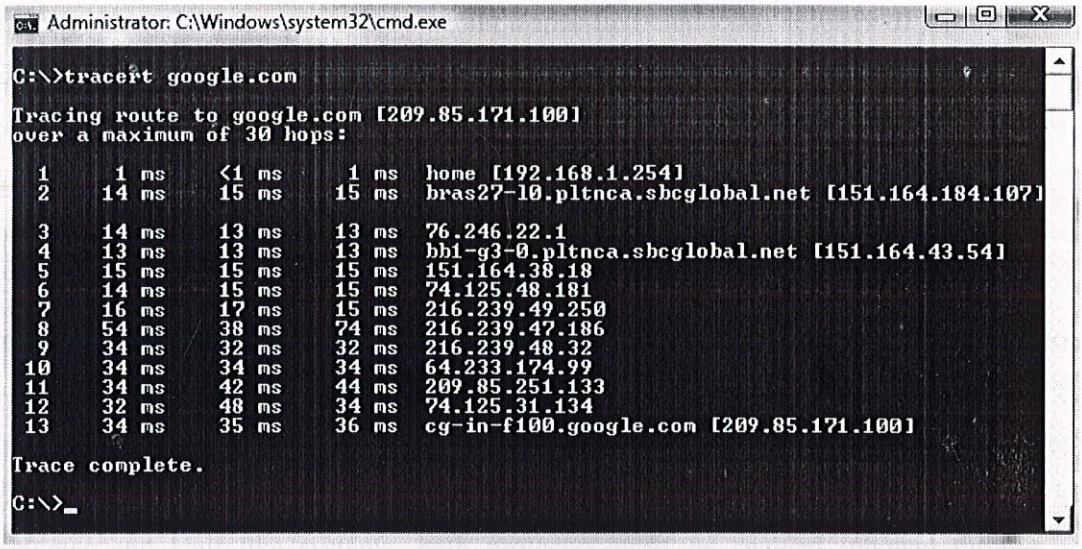
outputs the list of traversed routers in simple text format, together with timing information. Traceroute is available on most operating systems.

On Microsoft Windows operating systems it is named **tracert**. Windows NT-based operating systems also provide PathPing, with similar functionality. Variants with similar functionality are also available, such as **tracepath** on Linux installations. For Internet Protocol Version 6 (IPv6) the tool sometimes has the name **traceroute6**.

Implementation

Traceroute sends a sequence of Internet Control Message Protocol (ICMP) packets addressed to a destination host. Determining the intermediate routers traversed involves adjusting the time-to-live (TTL) aka **hop limit** Internet Protocol parameter. Frequently starting with a value like 30, routers decrement this and discard a packet when the TTL value has reached zero, returning the ICMP error message ICMP Time Exceeded.

Traceroute works by increasing the TTL value of each successive set of packets sent. The first set of packets sent have a **hop limit** value of 1, expecting that they are not forwarded by the first router. The next set have a **hop limit** value of 2, so that the second router will send the error reply. This continues until the destination host receives the packets and returns an **ICMP Echo Reply message.**



Traceroute uses the returned ICMP messages to produce a list of routers that the packets have traversed. The timestamp values returned for each router along the path are the delay (aka latency) values, typically measured in milliseconds for each packet.

Hop 192.168.1.2 Depth 1

Probe status: unsuccessful

Parent: ()

Return code: Label-switched at stack-depth 1

Sender timestamp: 2008-04-17 09:35:27 EDT 400.88 msec

Receiver timestamp: 2008-04-17 09:35:27 EDT 427.87 msec

Response time: 26.99 msec

MTU: Unknown

Multipath type: IP

Address Range 1: 127.0.0.64 ~ 127.0.0.127

Label Stack:

Label 1 Value 299792 Protocol RSVP-TE

The originating host expects a reply within a specified number of seconds. If a packet is not acknowledged within the expected timeout, an asterisk is displayed. The hosts listed may not be hosts used by other packets. The Internet Protocol does not require that packets between two hosts take the same route. Also note that if the host at hop number N does not reply, the hop will be skipped in the output.

On Unix-like operating systems, the traceroute utility by default uses User Datagram Protocol (UDP) datagrams with destination port numbers from 33434 to 33534. The traceroute utility usually has an option to specify use of ICMP echo request (type 8) instead, as used by the Windows **tracert** utility. If a network has a firewall and operates both MS Windows and Unix-like systems, both protocols must be enabled inbound through the firewall.

There are also traceroute implementations that use TCP packets, such as tcptraceroute or layer four traceroute. PathPing is a utility introduced with Windows NT that combines ping and traceroute functionality. mtr (my traceroute) is an enhanced version of ICMP traceroute which is available for Unix-like and Windows systems. All implementations of traceroute rely on ICMP (type 11) packets being sent to the originator.

The implementations of traceroute shipped with Linux , FreeBSD, NetBSD, OpenBSD, DragonFly BSD, and Mac OS X include an option to use ICMP Echo packets (-I) or any arbitrary protocol (-P) such as UDP, TCP, ICMP, or GRE.

Usage

Most implementations include at least options to specify the number of queries to send per hop, time to wait for a response, the **hop limit** and port to use. `traceroute` will display the options if invoked without any, `man traceroute` will display details including error flags displayed. Simple example on linux:

```
traceroute -w 3 -q 1 -m 16 -p  example.com
```

Only wait 3 seconds (instead of 5), only send out 1 query to each hop (instead of 3), limit the maximum number of hops to 16 before giving up (instead of 30) with the final hostexample.com.

This can help identify incorrect routing table definitions or firewalls that may be blocking ICMP traffic, or high port UDP in UNIX ping, to a site. Note that a firewall may permit IMCP packets but not permit packets of other protocols.

Traceroute is also used by penetration testers to gather information about network infrastructure and IP ranges around a given host.

It can also be used when downloading data, and if there are multiple mirrors available for the same piece of data, one can trace each mirror to get a good idea of which mirror would be the fastest to use.

Security concerns

Supplying such detailed information about the pathways taken was considered acceptable and convenient in the early days of the Internet, but later was considered questionable for privacy and security reasons. Traceroute information has been frequently used by hackers as a way to acquire sensitive information about a company's network architecture. By using the traceroute command, a hacker can quickly map out intermediate routers for known destinations on a company's network architecture.

For these reasons, while traceroute was widely unrestricted during the early days of the Internet, today many networks block, or de-prioritize the ICMP time exceeded message that is required to determine round trip time. However, filtering traffic except at network end-points is a controversial practice.

4.1.3 NETSTAT

Netstat (**net**work **stat**istics) is a command-line tool that displays network connections (both incoming and outgoing), routing tables, and a number of network interface statistics. It is available on UNIX, Unix-like, and Windows NT-based operating systems.

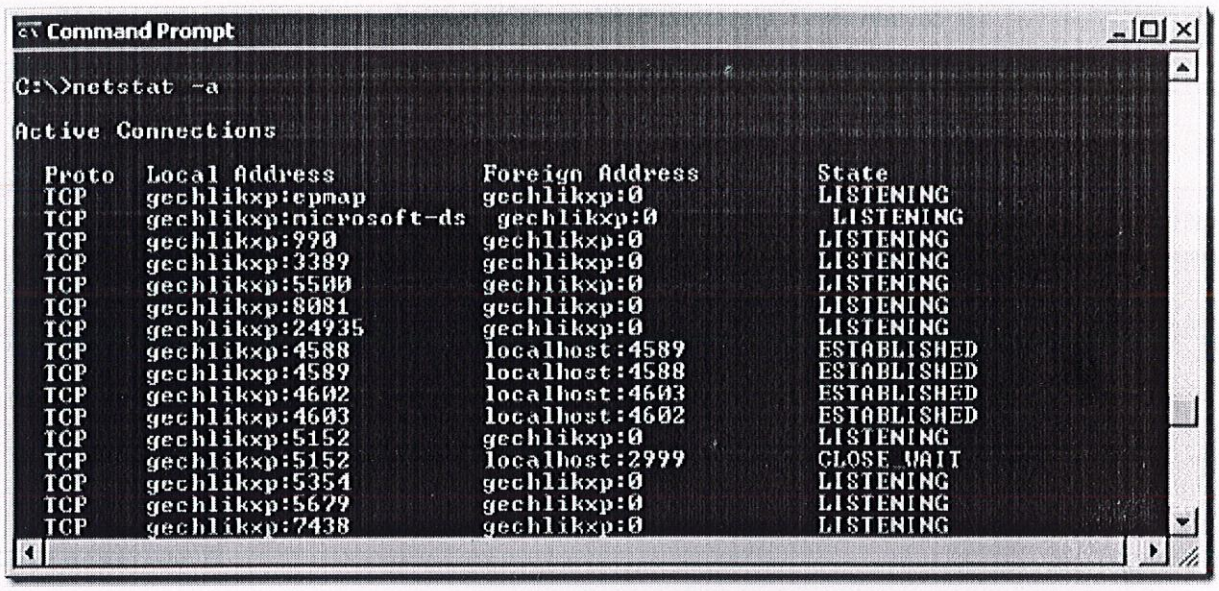
It is used for finding problems in the network and to determine the amount of traffic on the network as a performance measurement.

Parameters

Parameters used with this command must be prefixed with a hyphen (-) rather than a slash (/).

- **-a** : Displays **a**ll active connections and the TCP and UDP ports on which the computer is listening.
- **-b** : Displays the **b**inary (executable) program's name involved in creating each connection or listening port. (Windows XP, 2003 Server and newer Windows operating systems (not Microsoft Windows 2000 or other non-Windows operating systems)) On Mac OS X when combined with **-i**, the total number of **b**ytes of traffic will be reported.
- **-e** : Displays **E**thernet statistics, such as the number of bytes and packets sent and received. This parameter can be combined with **-s**.
- **-f** : Displays **f**ully qualified domain names <FQDN> for foreign addresses.(only available on Windows Vista and newer operating systems)
- **-g** : Displays multicast **g**roup membership information for both IPv4 and IPv6 (may only be available on newer operating systems)
- **-i** : Displays network **i**nterfaces and their statistics (not available under Windows)
- **-n** : Displays active TCP connections, however, addresses and port numbers are expressed numerically and no attempt is made to determine names.
- **-m** : Displays the STREAMS statistics.
- **-o** : Displays active TCP connections and includes the process ID (PID) for each connection. You can find the application based on the PID on the **Processes** tab in Windows Task Manager. This parameter can be combined with **-a, -n,** and **-p**. This parameter is available on Microsoft Windows XP, 2003 Server (and Windows 2000 if a hotfix is applied.

- **-p** Windows and BSD: **P**rotocol: Shows connections for the protocol specified by Protocol. In this case, the Protocol can be **tcp, udp, tcpv6, or udpv6**. If this parameter is used with **-s**to display statistics by protocol, Protocol can be **tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6,** or **ipv6**.
- **-p** Linux: **P**rocess : Show which processes are using which sockets (similar to -b under Windows) (you must be root to do this)
- **-P** Solaris: **P**rotocol: Shows connections for the protocol specified by Protocol. In this case, the Protocol can be **ip, ipv6, icmp, icmpv6, igmp, udp, tcp,** or **rawip**.
- **-r** : Displays the contents of the IP **r**outing table. (This is equivalent to the **route print** command under Windows.)
- **-s** : Displays **s**tatistics by protocol. By default, statistics are shown for the TCP, UDP, ICMP, and IP protocols. If the IPv6 protocol for Windows XP is installed, statistics are shown for the TCP over IPv6 UDP over IPv6, ICMPv6, and IPv6 protocols. The **-p** parameter can be used to specify a set of protocols.
- **-v** : When used in conjunction with **-b** it will display the sequence of components involved in creating the connection or listening port for all executables.
- **Interval** : Redisplays the selected information every Interval seconds. Press CTRL+C to stop the redisplay. If this parameter is omitted, netstat prints the selected information only once.
- **-h** (UNIX) / (Windows): Displays help at the command prompt.



Statistics provided

Netstat provides statistics for the following:

- Proto - The name of the protocol (TCP or UDP).
- Local Address - The IP address of the local computer and the port number being used. The name of the local computer that corresponds to the IP address and the name of the port is shown unless the **-n** parameter is specified. If the port is not yet established, the port number is shown as an asterisk (*).
- Foreign Address - The IP address and port number of the remote computer to which the socket is connected. The names that corresponds to the IP address and the port are shown unless the **-n** parameter is specified. If the port is not yet established, the port number is shown as an asterisk (*).
- State - Indicates the state of a TCP connection. The possible states are as follows: CLOSE_WAIT, CLOSED, ESTABLISHED, FIN_WAIT_1, FIN_WAIT_2, LAST_ACK, LISTEN, SYN_RECEIVED, SYN_SEND, and TIME_WAIT. For more information about the states of a TCP connection.

Examples

To display the statistics for only the TCP or UDP protocols, type one of the following commands:

```
netstat -sp tcp
netstat -sp udp
```

To display active TCP connections and the process IDs every **5** seconds, type the following command (On Microsoft Windows, works on XP and 2003 only, or Windows 2000 with hotfix):

```
netstat -o 5
```

Mac OS X version

```
netstat -w 5
```

To display active TCP connections and the process IDs using **n**umerical form, type the following command (On Microsoft Windows, works on XP and 2003 only, or Windows 2000 with hotfix):

```
netstat -no
```

To display all ports open by a process with id *pid*

```
netstat -ao | grep "pid"
```

Caveats

Some versions of `netstat` lack explicit field delimiters in their printf-generated output, leading to numeric fields running together and thus corrupting the output data.

Platform specific remarks

Under Linux, raw data can often be obtained from the **/proc/net/dev** to work around the printf output corruption arising in netstat's network interface statistics summary, `netstat -i`, until such time as the problem is corrected.

On the Windows platform, netstat information can be retrieved by calling the **GetTcpTable** and **GetUdpTable** functions in the IP Helper API, or IPHLPAPI.DLL. Information returned includes local and remote IP addresses, local and remote ports, and (for GetTcpTable) TCP status codes.

In addition to the command-line netstat.exe tool that ships with Windows, GUI-based netstat programs are available.

On the Windows platform, this command is available only if the **Internet Protocol (TCP/IP)** protocol is installed as a component in the properties of a network adapter in Network Connections.

On Mac OS X **10.5**, the above option "-o" is not available. With Mac OS X **10.5**, the /Applications/Utilities folder contains a network utility called: Network Utility, see tab **Netstat** for these stats presented in a GUI application, along with Ping, Lookup, Traceroute, Whois, Finger and Port Scan.

4.2 HARDWARE USED

- A Main computer to be used as Server
- 2-3 more computers to act as client onto the test network
- Ethernet Cable
- 8-port Switch
- Router.

4.2.1 Server

In computing, the term **server** is used to refer to one of the following:

- A computer program running as a service, to serve the needs or requests of other programs (referred to in this context as "clients") which may or may not be running on the same computer.

- A physical computer dedicated to running one or more such services, to serve the needs of programs running on other computers on the same network.
- A software/hardware system (i.e. a software service running on a dedicated computer) such as a database server, file server, mail server, or print server.

In computer networking, a **server** is a program that operates as a socket listener. The term **server** is also often generalized to describe host that is deployed to execute one or more such programs.

A **server computer** is a computer, or series of computers, that link other computers or electronic devices together. They often provide essential services across a network, either to private users inside a large organization or to public users via the internet. For example, when you enter a query in a search engine, the query is sent from your computer over the internet to the servers that store all the relevant web pages. The results are sent back by the server to your computer.

Many servers have dedicated functionality such as web servers, print servers, and database servers. **Enterprise servers** are servers that are used in a business context.

Usage

Servers provide essential services across a network, either to private users inside a large organization or to public users via the Internet. For example, when you enter a query in a search engine, the query is sent from your computer over the internet to the servers that store all the relevant web pages. The results are sent back by the server to your computer.

The term *server* is used quite broadly in information technology. Despite the many server-branded products available (such as server versions of hardware, software or operating systems), in theory any computerized process that shares a resource to one or more client processes is a server. To illustrate this, take the common example of file sharing. While the existence of files on a machine does not classify it as a server, the mechanism which shares these files to clients by the operating system is the server.

Similarly, consider a web server application (such as the multiplatform "Apache HTTP Server"). This web server software can be *run* on any capable computer. For example, while laptop or personal computer is not typically known as a server, they can in these situations fulfill the role of one, and hence be labeled as one. It is in this case that the machine's purpose as a web server classifies it in general as a server.

In the hardware sense, the word *server* typically designates computer models intended for hosting software applications under the heavy demand of a network environment. In this client–server configuration one or more machines, either a computer or a computer appliance, share information with each other with one acting as a host for the other.

While nearly any personal computer is capable of acting as a network server, a dedicated server will contain features making it more suitable for production environments. These features may include a faster CPU, increased high-performance RAM, and typically more than one large hard drive. More obvious distinctions include marked redundancy in power supplies, network connections, and even the servers themselves.

Between the 1990s and 2000s an increase in the use of *dedicated hardware* saw the advent of self-contained **server appliances**. One well-known product is the Google Search Appliance, a unit that combines hardware and software in an out-of-the-box packaging. Simpler examples of such appliances include switches, routers, gateways, and print server, all of which are available in a near plug-and-play configuration.

Modern operating systems such as Microsoft Windows or Linux distributions rightfully seem to be designed with client–server architecture in mind. These operating systems attempt to abstract hardware, allowing a wide variety of software to work with components of the computer. In a sense, the operating system can be seen as *serving* hardware to the software, which in all but low-level programming languages must interact using an API.

These operating systems may be able to run programs in the background called either services or daemons. Such programs may wait in a sleep state for their necessity to become apparent, such as the aforementioned *Apache HTTP Server* software. Since any software that provides services can be *called* a server, modern personal computers can be seen as a forest of servers and clients operating in parallel.

The Internet itself is also a forest of servers and clients. Merely requesting a web page from a few kilometers away involves satisfying a stack of protocols that involve many examples of hardware and software servers. The least of these are the routers, modems, domain name servers, and various other servers necessary to provide us the World Wide Web.

Server hardware

Hardware requirements for servers vary, depending on the server application. Absolute CPU speed is not usually as critical to a server as it is to a desktop machine· Servers' duties to provide service to many users over a network lead to different requirements like fast network connections and high I/O throughput. Since servers are usually accessed over a network, they may run in headless mode without a monitor or input device. Processes that are not needed for the server's function are not used.

Many servers do not have a graphical user interface (GUI) as it is unnecessary and consumes resources that could be allocated elsewhere. Similarly, audio and USB interfaces may be omitted.

Servers often run for long periods without interruption and availability must often be very high, making hardware reliability and durability extremely important. Although servers can be built from commodity computer parts, mission-critical enterprise servers are ideally very fault tolerant and use specialized hardware with low failure rates in order to maximize uptime, for even a short-term failure can cost more than purchasing and installing the system. For example, it may take only a few minutes of down time at a national stock exchange to justify the expense of entirely replacing the system with something more reliable. Servers may incorporate faster, higher-capacity hard drives, larger computer fans or water cooling to help remove heat, and uninterruptible power supplies that ensure the servers continue to function in the event of a power failure. These components offer higher performance and reliability at a correspondingly higher price. Hardware redundancy—installing more than one instance of modules such as power supplies and hard disks arranged so that if one fails another is automatically available—is widely used. ECC memory devices that detect and correct errors are used; non-ECC memory is more likely to cause data corruption.

To increase reliability, most of the servers use memory with error detection and correction, redundant disks, redundant power supplies and so on. Such components are also frequently hot swappable, allowing replacing them on the running server without shutting it down. To prevent overheating, servers often have more powerful fans. As servers are usually administered by qualified engineers, their operating systems are also more tuned for stability and performance than for user friendliness and ease of use, Linux taking noticeably larger percentage than for desktop computers.

As servers need stable power supply, good Internet access, increased security and less noise, it is usual to store them in dedicated server centers or special rooms. This requires reducing power consumption as extra energy used generates more heat and the temperature in the room could exceed the acceptable limits. Normally server rooms are equipped with air conditioning devices. Server casings are usually flat and wide, adapted to store many devices next to each other in server rack. Unlike ordinary computers, servers usually can be configured, powered up and down or rebooted remotely, using out-of-band management.

Many servers take a long time for the hardware to start up and load the operating system. Servers often do extensive pre-boot memory testing and verification and startup of remote management services. The hard drive controllers then start up banks of drives sequentially, rather than all at once, so as not to overload the power supply with startup surges, and afterwards they initiate RAID system pre-checks for correct operation of redundancy. It is common for a machine to take several minutes to start up, but it may not need restarting for months or years.

Server operating systems

Server-oriented operating systems tend to have certain features in common that make them more suitable for the server environment, such as

- GUI not available or optional
- ability to reconfigure and update both hardware and software to some extent without restart,

- advanced backup facilities to permit regular and frequent online backups of critical data,
- transparent data transfer between different volumes or devices,
- flexible and advanced networking capabilities,
- automation capabilities such as daemons in UNIX and services in Windows, and
- Tight system security, with advanced user, resource, data, and memory protection.

Server-oriented operating systems can, in many cases, interact with hardware sensors to detect conditions such as overheating, processor and disk failure, and consequently alert an operator or take remedial measures itself.

Because servers must supply a restricted range of services to perhaps many users while a desktop computer must carry out a wide range of functions required by its user, the requirements of an operating system for a server are different from those of a desktop machine. While it is possible for an operating system to make a machine both provide services and respond quickly to the requirements of a user, it is usual to use different operating systems on servers and desktop machines. Some operating systems are supplied in both server and desktop versions with similar user interface.

The desktop versions of the Windows and Mac OS X operating systems are deployed on a minority of servers, as are some proprietary mainframe operating systems, such as z/OS. The dominant operating systems among servers are UNIX-based or open source kernel distributions, such as Linux (the kernel). The rise of the microprocessor-based server was facilitated by the development of UNIX to run on the x86 microprocessor architecture. The Microsoft Windows family of operating systems also runs on x86 hardware, and since Windows NT have been available in versions suitable for server use.

While the role of server and desktop operating systems remains distinct, improvements in the reliability of both hardware and operating systems have blurred the distinction between the two classes. Today, many desktop and server operating systems share similar code bases, differing mostly in configuration. The shift towards web

applications and middleware platforms has also lessened the demand for specialist application servers.

4.2.2 CLIENT

A **client** is an application or system that accesses a remote service on another computer system, known as a server, by way of a network. The term was first applied to devices that were not capable of running their own stand-alone programs, but could interact with remote computers via a network. These dumb terminals were clients of the time-sharing mainframe computer.

The client–server model is still used today. Client and server can run on the same machine and connect via UNIX domain sockets. Using Internet sockets a user may connect to a service operating on a possibly remote system through the Internet protocol suite. Servers set up *listening* sockets, and clients initiate connections that a server may *accept*.

Web browsers are clients that connect to web servers and retrieve web pages for display. Most people use email clients to retrieve their email from their internet service provider's mail storage servers. Online chat uses a variety of clients, which vary depending on the chat protocol being used. Multiplayer online games may run as Game Clients on each local computer.

Increasingly, existing large client applications are being switched to websites, making the browser a sort of universal client. This avoids the hassle of downloading a large piece of software onto any computer you want to use the application on. An example of this is the rise of webmail.

In personal computers and computer workstations, the difference between client and server operating system is often just a matter of marketing - the server version may contain more operating system components, allow more simultaneous logins, and may be more expensive, while the client version may contain more end-user software.

4.2.3 Ethernet Cable

Ethernet is a family of frame-based computer networking technologies for local area networks (LAN). It defines a number of wiring and signaling standards for the Physical Layer of the standard networking model as well as a common addressing format and a variety of Medium Access Control procedures at the lower part of the Data Link Layer.

Ethernet has been commercially available since around 1980, largely replacing competing wired LAN standards. Most common are Ethernet over twisted pair to connect end systems, and fiber optic versions for site backbones. It is standardized as IEEE 802.3.

Standardization

Notwithstanding its technical merits, timely standardization was instrumental to the success of Ethernet. It required well-coordinated and partly competitive activities in several standardization bodies such as the IEEE, ECMA, IEC, and finally ISO.

In February 1980, the Institute of Electrical and Electronics Engineers (IEEE) started project 802 to standardization local area networks (LAN). The "DIX-group" with Gary Robinson (DEC), Phil Arst (Intel), and Bob Printis (Xerox) submitted the so-called "Blue Book" CSMA/CD specification as a candidate for the LAN specification. Since IEEE membership is open to all professionals, including students, the group received countless comments on this technology.

In addition to CSMA/CD, Token Ring (supported by IBM) and Token Bus (selected and henceforward supported by General Motors) were also considered as candidates for a LAN standard. Due to the goal of IEEE 802 to forward only one standard and due to the strong company support for all three designs, the necessary agreement on a LAN standard was significantly delayed.

In the Ethernet camp, it put at risk the market introduction of the Xerox Star workstation and 3Com's Ethernet LAN products. With such business implications in mind, David Liddle(General Manager, Xerox Office Systems) and Metcalfe (3Com) strongly supported a proposal of Fritz Röscheisen (Siemens Private Networks) for an alliance in the emerging office communication market, including Siemens' support for the international standardization of Ethernet (April 10, 1981). Ingrid Fromm, Siemens' representative to IEEE 802, quickly achieved broader support for Ethernet beyond IEEE by the establishment of a

competing Task Group "Local Networks" within the European standards body ECMA TC24. As early as March 1982 ECMA TC24 with its corporate members reached agreement on a standard for CSMA/CD based on the IEEE 802 draft. The speedy action taken by ECMA decisively contributed to the conciliation of opinions within IEEE and approval of IEEE 802.3 CSMA/CD by the end of 1982. IEEE published the 802.3 standard as a draft in 1983[10] and as a standard in 1985.

Approval of Ethernet on the international level was achieved by a similar, cross-partisan action with Fromm as liaison officer working to integrate International Electrotechnical Commission, TC83 and International Organization for Standardization (ISO) TC97SC6, and the ISO/IEEE 802/3 standard was approved in 1984.

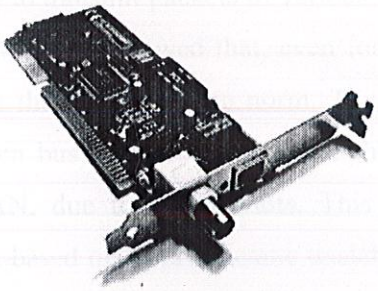
Evolution

Ethernet evolved to include higher bandwidth, improved media access control methods, and different physical media. The coaxial cable was replaced with point-to-point links connected by Ethernet repeaters or switches to reduce installation costs, increase reliability, and improve management and troubleshooting. Many variants of Ethernet remain in common use.

Ethernet stations communicate by sending each other data packets: blocks of data individually sent and delivered. As with other IEEE 802 LANs, each Ethernet station is given a 48-bit MAC address. The MAC addresses are used to specify both the destination and the source of each data packet. Network interfaces normally do not accept packets addressed to other Ethernet stations. Adapters come programmed with a globally unique address. Despite the significant changes in Ethernet, all generations of Ethernet (excluding early experimental versions) use the same frame formats (and hence the same interface for higher layers), and can be readily interconnected through bridging.

Due to the ubiquity of Ethernet, the ever-decreasing cost of the hardware needed to support it, and the reduced panel space needed by twisted pair Ethernet, most manufacturers now build Ethernet interfaces directly into PC motherboards, eliminating the need for installation of a separate network card.

Shared media



Ethernet was originally based on the idea of computers communicating over a shared coaxial cable acting as a broadcast transmission medium. The methods used were similar to those used in radio systems, with the common cable providing the communication channel likened to the *Luminiferous aether* in 19th century physics, and it was from this reference that the name "Ethernet" was derived.

Original Ethernet's shared coaxial cable (the shared medium) traversed a building or campus to every attached machine. A scheme known as carrier sense multiple accesses with collision detection (CSMA/CD) governed the way the computers shared the channel. This scheme was simpler than the competing token ring or token bus technologies. Computers were connected to an Attachment Unit Interface (AUI) transceiver, which was in turn connected to the cable (later with thin Ethernet the transceiver was integrated into the network adapter). While a simple passive wire was highly reliable for small networks, it was not reliable for large extended networks, where damage to the wire in a single place, or a single bad connector, could make the whole Ethernet segment unusable.

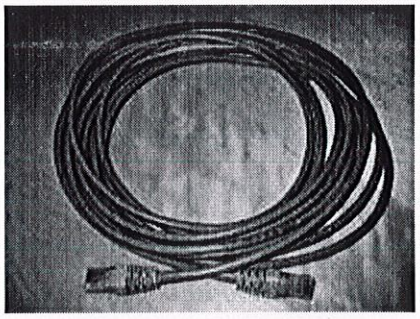
Since all communications happen on the same wire, any information sent by one computer is received by all, even if that information is intended for just one destination. The network interface card interrupts the CPU only when applicable packets are received: The card ignores information not addressed to it. Use of a single cable also means that the bandwidth is shared, so that network traffic can be very slow when many stations are simultaneously active.

Collisions reduce throughput by their very nature. In the worst case, when there are lots of hosts with long cables that attempt to transmit many short frames, excessive collisions can

reduce throughput dramatically. However, a Xerox report in 1980 summarized the results of having 20 fast nodes attempting to transmit packets of various sizes as quickly as possible on the same Ethernet segment. The results showed that, even for the smallest Ethernet frames (64 bytes), 90% throughput on the LAN was the norm. This is in comparison with token passing LANs (token ring, token bus), all of which suffer throughput degradation as each new node comes into the LAN, due to token waits. This report was controversial, as modeling showed that collision-based networks became unstable under loads as low as 40% of nominal capacity. Many early researchers failed to understand the subtleties of the CSMA/CD protocol and how important it was to get the details right, and were really modeling somewhat different networks (usually not as good as real Ethernet).

Repeaters and hubs

For signal degradation and timing reasons, coaxial Ethernet segments had a restricted size. Somewhat larger networks could be built by using an Ethernet repeater. Early repeaters had only 2 ports, but they gave way to 4, 6, 8, and more ports as the advantages of cabling in a star network were recognized.



A Twisted Pair Cable

Ethernet on unshielded twisted-pair cables (UTP) began with StarLAN at 1 Mbit/s. SynOptics introduced the first twisted-pair Ethernet at 10 Mbit/s in a star-wired cabling topology with a central hub, later called LattisNet, which later became 10BASE-T, which was designed for point-to-point links only, and all termination was built into the device. This changed repeaters from a specialist device used at the center of large networks

to a device that every twisted pair-based network with more than two machines had to use. The tree structure that resulted from this made Ethernet networks more reliable by preventing most faults with one peer or its associated cable from affecting other devices on the network.

Despite the physical star topology, repeater based Ethernet networks still use half-duplex and CSMA/CD, with only minimal activity by the repeater, primarily the Collision Enforcement signal, in dealing with packet collisions. Every packet is sent to every port on the repeater, so bandwidth and security problems are not addressed. The total throughput of the repeater is limited to that of a single link, and all links must operate at the same speed.

Bridging and switching

While repeaters could isolate some aspects of Ethernet segments, such as cable breakages, they still forwarded all traffic to all Ethernet devices. These created practical limits on how many machines could communicate on an Ethernet network. The entire network was one collision domain, and all hosts had to be able to detect collisions anywhere on the network. This limited the number of repeaters between the farthest nodes. Segments joined by repeaters had to all operate at the same speed, making phased-in upgrades impossible.

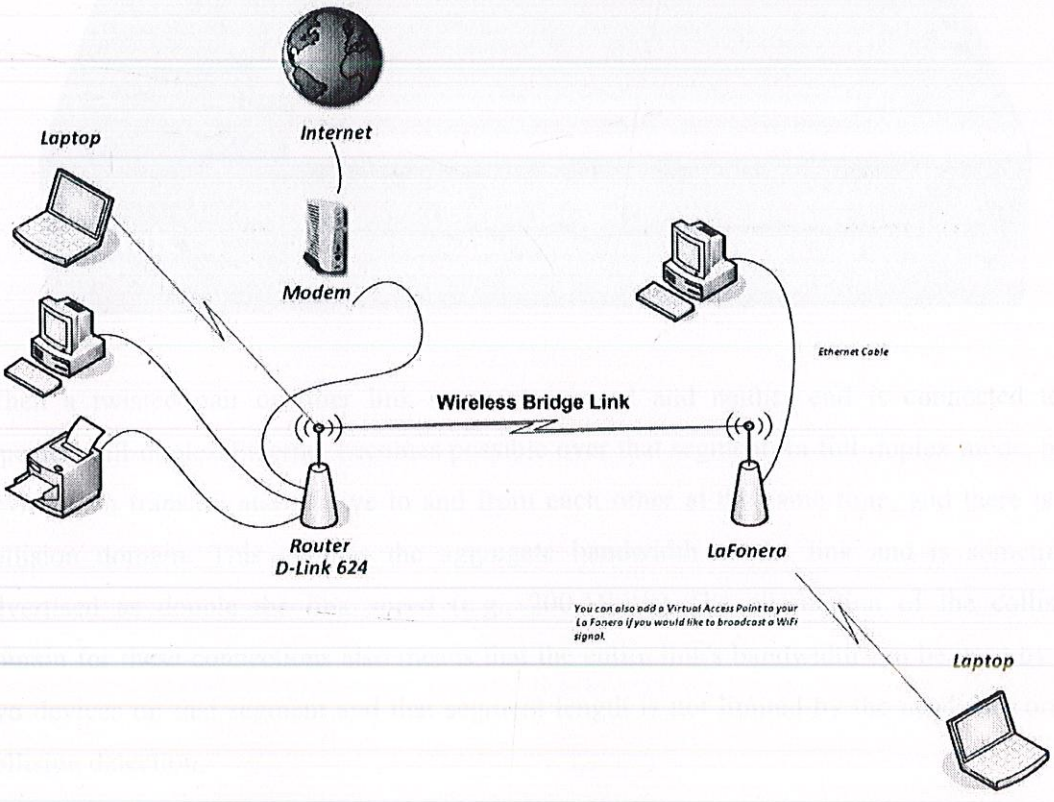
To alleviate these problems, bridging was created to communicate at the data link layer while isolating the physical layer. With bridging, only well-formed Ethernet packets are forwarded from one Ethernet segment to another; collisions and packet errors are isolated.

Prior to discovery of network devices on the different segments, Ethernet bridges (and switches) work somewhat like Ethernet repeaters, passing all traffic between segments. However, as the bridge discovers the addresses associated with each port, it forwards network traffic only to the necessary segments, improving overall performance. Broadcast traffic is still forwarded to all network segments. Bridges also overcame the limits on total segments between two hosts and allowed the mixing of speeds, both of which became very important with the introduction of Fast Ethernet.

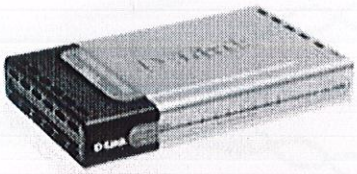
Early bridges examined each packet one by one using software on a CPU, and some of them were significantly slower than repeaters at forwarding traffic, especially when handling many ports at the same time. This was in part because the entire Ethernet packet would be read into

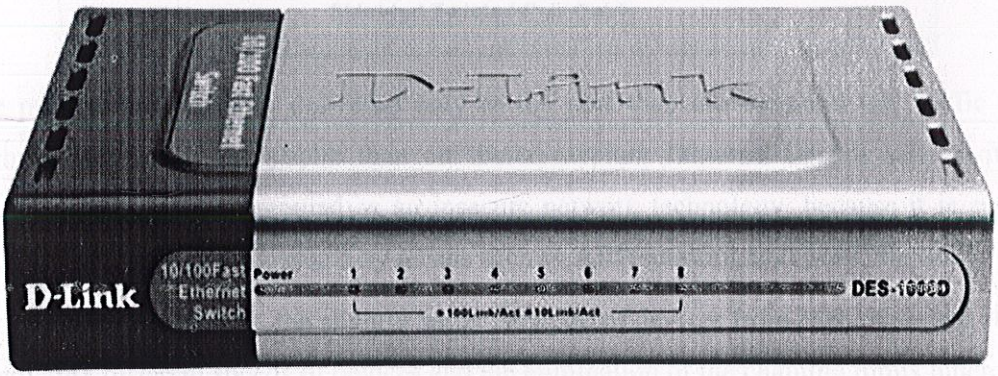
a buffer, the destination address compared with an internal table of known MAC addresses, and a decision made as to whether to drop the packet or forward it to another or all segments.

In 1989, the networking company Kalpana introduced their EtherSwitch, the first Ethernet switch. This worked somewhat differently from an Ethernet bridge, in that only the header of the incoming packet would be examined before it was either dropped or forwarded to another segment. This greatly reduced the forwarding latency and the processing load on the network device. One drawback of this cut-through switching method was that packets that had been corrupted would still be propagated through the network, so a jabbering station could continue to disrupt the entire network. The eventual remedy for this was a return to the original store and forward approach of bridging, where the packet would be read into a buffer on the switch in its entirety, verified against its checksum and then forwarded, but using more powerful application-specific integrated circuits. Hence, the bridging is then done in hardware, allowing packets to be forwarded at full wire speed.

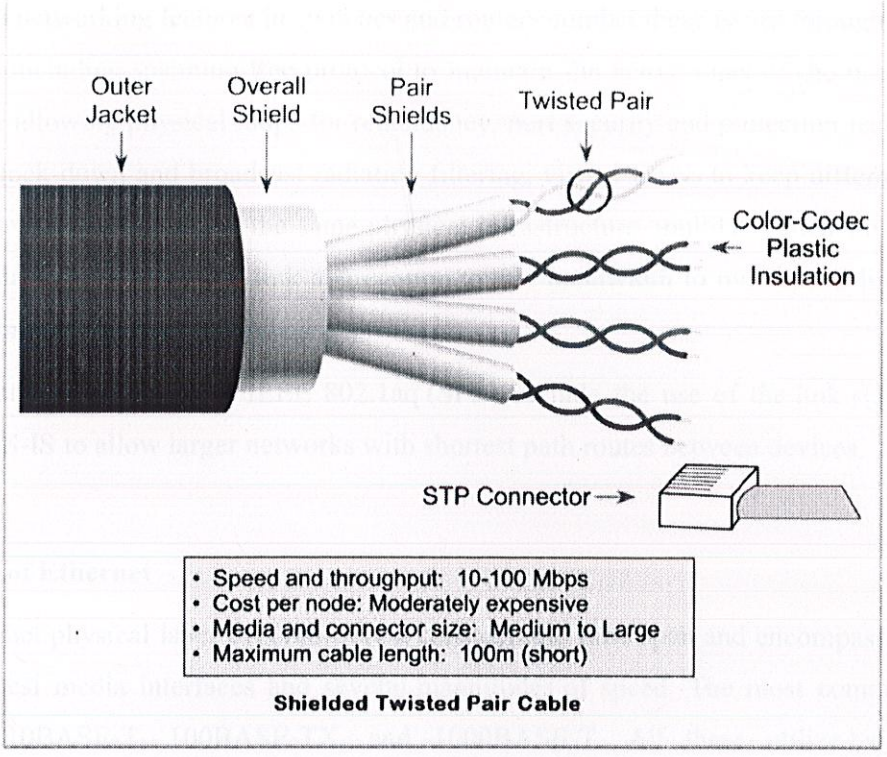


D-Link 8-Port Switch





When a twisted pair or fiber link segment is used and neither end is connected to a repeater, full-duplex Ethernet becomes possible over that segment. In full-duplex mode, both devices can transmit and receive to and from each other at the same time, and there is no collision domain. This doubles the aggregate bandwidth of the link and is sometimes advertised as double the link speed (e.g., 200 Mbit/s). The elimination of the collision domain for these connections also means that the entire link's bandwidth can be used by the two devices on that segment and that segment length is not limited by the need for correct collision detection.



Shielded Twisted Pair Cable

Since packets are typically delivered only to the port they are intended for, traffic on a switched Ethernet is less public than on shared-medium Ethernet. Despite this, switched Ethernet should still be regarded as an insecure network technology, because it is easy to subvert switched Ethernet systems by means such as ARP spoofing and MAC flooding.

The bandwidth advantages, the slightly better isolation of devices from each other, the ability to easily mix different speeds of devices and the elimination of the chaining limits inherent in non-switched Ethernet have made switched Ethernet the dominant network technology.

Advanced networking

Simple switched Ethernet networks, while a great improvement over repeater-based Ethernet, suffer from single points of failure, attacks that trick switches or hosts into sending data to a machine even if it is not intended for it, scalability and security issues with regard to broadcast radiation and multicast traffic, and bandwidth choke points where a lot of traffic is forced down a single link.

Advanced networking features in switches and routers combat these issues through a number of means including spanning-tree protocol to maintain the active links of the network as a tree while allowing physical loops for redundancy, port security and protection features such as MAC lock down and broadcast radiation filtering, virtual LANs to keep different classes of users separate while using the same physical infrastructure, multilayer switching to route between different classes and link aggregation to add bandwidth to overloaded links and to provide some measure of redundancy.

Recent networking advances IEEE 802.1aq (SPB) include the use of the link-state routing protocol IS-IS to allow larger networks with shortest path routes between devices.

Varieties of Ethernet

The Ethernet physical layer evolved over a considerable time span and encompasses quite a few physical media interfaces and several magnitudes of speed. The most common forms used are 10BASE-T, 100BASE-TX, and 1000BASE-T. All three utilize twisted pair cables and 8P8C modular connectors.

They run at 10 Mbit/s, 100 Mbit/s, and 1 Gbit/s, respectively. Fiber optic variants of Ethernet offer high performance, electrical isolation and distance (tens of kilometers with some versions). In general, network protocol stack software will work similarly on all varieties.

Ethernet frames

A data packet on the wire is called a frame. A frame begins with Preamble and Start Frame Delimiter, following which each Ethernet frame features an Ethernet header featuring source and destination MAC addresses. The middle section of the frame consists of payload data including any headers for other protocols (e.g., Internet Protocol) carried in the frame. The frame ends with a 32-bit cyclic redundancy check, which is used to detect any corruption of data in transit.

Autonegotiation

Autonegotiation is the procedure by which two connected devices choose common transmission parameters, such as speed and duplex mode. Autonegotiation was first introduced as an optional feature for Fast Ethernet, but it is also backward compatible with 10BASE-T. Autonegotiation is mandatory for Gigabit Ethernet.

4.2.4 Router

A **router** is a device that forwards data packets across computer networks. Routers perform the data "traffic directing" functions on the Internet. A router is a microprocessor-controlled device that is connected to two or more data lines from different networks. When a data packet comes in on one of the lines, the router reads the address information in the packet to determine its ultimate destination. Then, using information in its routing table, it directs the packet to the next network on its journey. A data packet is typically passed from router to router through the networks of the Internet until it gets to its destination computer. Routers also perform other tasks such as translating the data transmission protocol of the packet to the appropriate protocol of the next network, and preventing unauthorized access to a network by the use of a firewall.

The most familiar type of routers are home and small office routers that simply pass data, such as web pages and email, between the home computers and the owner's cable or DSL modem, which connects to the Internet (ISP). However more sophisticated routers range from enterprise routers, which connect large business or ISP networks up to the powerful core routers that forward data at high speed along the optical fiber lines of the Internet backbone.

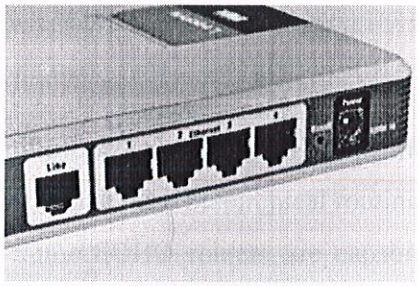
D-link Router

Applications

When multiple routers are used in interconnected networks, the routers exchange information about destination addresses, using a dynamic routing protocol. Each router builds up a table listing the preferred routes between any two systems on the interconnected networks. A router has interfaces for different physical types of network connections, (such as copper cables, fiber optic, or wireless transmission). It also contains firmware for different networking protocol standards. Each network interface uses this specialized computer software to enable data packets to be forwarded from one protocol transmission system to another.

Routers may also be used to connect two or more logical groups of computer devices known as subnets, each with a different sub-network address. The subnets addresses recorded in the router do not necessarily map directly to the physical interface connections. A router has two stages of operation called planes:

Control plane: A router records a routing table listing what route should be used to forward a data packet, and through which physical interface connection. It does these using internal pre-configured addresses, called static routes.

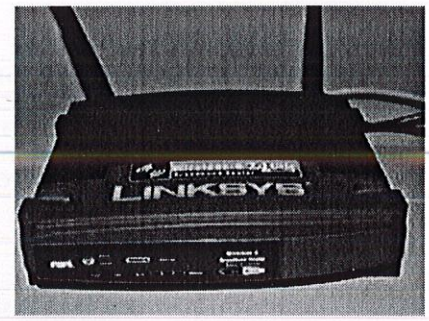


Forwarding plane: The router forwards data packets between incoming and outgoing interface connections. It routes it to the correct network type using information that the packet header contains. It uses data recorded in the routing table control plane.

Routers may provide connectivity within enterprises, between enterprises and the Internet, and between internet service providers (ISPs) networks. The largest routers (such as the Cisco CRS-1 or Juniper T1600) interconnect the various ISPs, or may be used in large enterprise networks. Smaller routers usually provide connectivity for typical home and office networks. Other networking solutions may be provided by a backbone Wireless Distribution System (WDS), which avoids the costs of introducing networking cables into buildings.

Enterprise routers

All sizes of routers may be found inside enterprises. The most powerful routers are usually found in ISPs, academic and research facilities. Large businesses may also need more powerful routers to cope with ever increasing demands of intranet data traffic. A three-layer model is in common use, not all of which need be present in smaller networks.



Access routers, including 'small office/home office' (SOHO) models, are located at customer sites such as branch offices that do not need hierarchical routing of their own. Typically, they are optimized for low cost. Some SOHO routers are capable of running alternative free Linux-based firmwares like Tomato, OpenWrt or DD-WRT.

Distribution

Distribution routers aggregate traffic from multiple access routers, either at the same site, or to collect the data streams from multiple sites to a major enterprise location. Distribution routers are often responsible for enforcing quality of service across a WAN, so they may have considerable memory installed, multiple WAN interface connections, and substantial onboard data processing routines. They may also provide connectivity to groups of file servers or other external networks.

Security

External networks must be carefully considered as part of the overall security strategy. Separate from the router may be a firewall or VPN handling device, or the router may include these and other security functions. Many companies produced security-oriented routers, including Cisco Systems' PIX and ASA5500 series, Juniper's Netscreen, Watchguard's Firebox, Barracuda's variety of mail-oriented devices, and many others.

Core

In enterprises, a core router may provide a "collapsed backbone" interconnecting the distribution tier routers from multiple buildings of a campus, or large enterprise locations. They tend to be optimized for high bandwidth.

Internet connectivity and internal use

Routers intended for ISP and major enterprise connectivity usually exchange routing information using the Border Gateway Protocol (BGP).RFC 4098 standard defines the types of BGP-protocol routers according to the routers' functions:

Edge router: Also called a Provider Edge router, is placed at the edge of an ISP network. The router uses External BGP to EBGP protocol routers in other ISPs, or a large enterprise Autonomous System.

Subscriber edge router: Also called a Customer Edge router is located at the edge of the subscriber's network, it also uses EBGP protocol to its provider's Autonomous System. It is typically used in an (enterprise) organization.

Inter-provider border router: Interconnecting ISPs is a BGP-protocol router that maintains BGP sessions with other BGP protocol routers in ISP Autonomous Systems.

Core router: A core router resides within an Autonomous System as a back bone to carry traffic between edge routers.

Within an ISP: In the ISPs Autonomous System, a router uses internal BGP protocol to communicate with other ISP edge routers, other intranet core routers, or the ISPs intranet provider border routers.

"Internet backbone:" The Internet no longer has a clearly identifiable backbone, unlike its predecessor networks. See default-free zone (DFZ). The major ISPs system routers make up what could be considered to be the current Internet backbone core. ISPs operate all four types of the BGP-protocol routers described here. An ISP "core" router is used to interconnect its edge and border routers. Core routers may also have specialized functions in virtual private networks based on a combination of BGP and Multi-Protocol Label Switching protocols.

Port forwarding: Routers are also used for port forwarding between private internet connected servers.

Voice/Data/Fax/Video Processing Routers: Commonly referred to as access servers or gateways, these devices are used to route and process voice, data, video, and fax traffic on the internet. Since 2005, most long-distance phone calls have been processed as IP traffic (VOIP) through a voice gateway,. Voice traffic that the traditional cable

networks once carried. Use of access server type routers expanded with the advent of the internet, first with dial-up access, and another resurgence with voice phone service.

Forwarding

For pure Internet Protocol (IP) forwarding function, a router is designed to minimize the state information associated with individual packets. The main purpose of a router is to connect multiple networks and forward packets destined either for its own networks or other networks. A router is considered a Layer 3 device because its primary forwarding decision is based on the information in the Layer 3 IP packet, specifically the destination IP address. This process is known as routing. When each router receives a packet, it searches its routing table to find the best match between the destination IP address of the packet and one of the network addresses in the routing table. Once a match is found, the packet is encapsulated in the Layer 2 data link frame for that outgoing interface. A router does not look into the actual data contents that the packet carries, but only at the layer 3 addresses to make a forwarding decision, plus optionally other information in the header for hint on, for example, QoS. Once a packet is forwarded, the router does not retain any historical information about the packet, but the forwarding action can be collected into the statistical data, if so configured.

Forwarding decisions can involve decisions at layers other than layer 3. A function that forwards based on layer 2 information is properly called a bridge. This function is referred to as layer 2 bridging, as the addresses it uses to forward the traffic are layer 2 addresses (e.g. MAC addresses on Ethernet).

Besides making decision as which interface a packet is forwarded to, which is handled primarily via the routing table, a router also has to manage congestion, when packets arrive at a rate higher than the router can process. Three policies commonly used in the Internet are tail drop, random early detection (RED), and weighted random early detection (WRED). Tail drop is the simplest and most easily implemented; the router simply drops packets once the length of the queue exceeds the size of the buffers in the router. RED probabilistically drops datagrams early when the queue is exceeds a pre-configured size of the queue until a pre-configured max when it becomes tail drop. WRED requires a weight on the average

queue size to act upon when the traffic is about to exceed the pre-configured size, so that short bursts will not trigger random drops.

Another function a router performs is to decide which packet should be processed first when multiple queues exist. This is managed through quality of service (QoS), which is critical when Voice over IP is deployed, so that delays between packets do not exceed 150ms to maintain the quality of voice conversations.

Yet another function a router performs is called policy-based routing where special rules are constructed to override the rules derived from the routing table when a packet forwarding decision is made.

These functions may be performed through the same internal paths that the packets travel inside the router. Some of the functions may be performed through an application-specific integrated circuit (ASIC) to avoid overhead caused by multiple CPU cycles, and others may have to be performed through the CPU as these packets need special attention that cannot be handled by an ASIC.

4.2.5 Switch

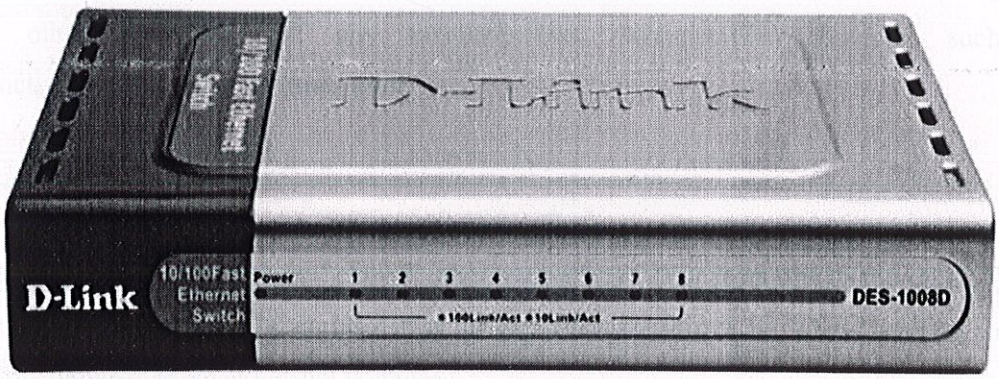
In electronics, a **switch** is an electrical component that can break an electrical circuit, interrupting the current or diverting it from one conductor to another. The most familiar form of switch is a manually operated electromechanical device with one or more sets of electrical contacts. Each set of contacts can be in one of two states: either 'closed' meaning the contacts are touching and electricity can flow between them, or 'open', meaning the contacts are separated and non-conducting.

A switch may be directly manipulated by a human as a control signal to a system, such as a computer keyboard button, or to control power flow in a circuit, such as a light switch. Automatically operated switches can be used to control the motions of machines, for example, to indicate that a garage door has reached its full open position or that a machine tool is in a position to accept another work piece. Switches may be operated by process variables such as pressure, temperature, flow, current, voltage, and force, acting as sensors in a process and used to automatically control a system.

For example, a thermostat is a temperature-operated switch used to control a heating process. A switch that is operated by another electrical circuit is called a relay. Large switches may be remotely operated by a motor drive mechanism.

Some switches are used to isolate electric power from a system, providing a visible point of isolation that can be pad-locked if necessary to prevent accidental operation of a machine during maintenance, or to prevent electric shock.

D-Link 8-Port Switch



In circuit theory

In electronics engineering, an ideal switch describes a switch that:

- has no current limit during its ON state
- has infinite resistance during its OFF state

- has no voltage drop across the switch during its ON state
- has no voltage limit during its OFF state
- has zero rise time and fall time during state changes
- switches only once without "bouncing" between on and off positions

Practical switches have loss and limitation. The ideal switch is often used in circuit analysis as it greatly simplifies the system of equations to be solved, however this can lead to a less accurate solution.

Contacts

In the simplest case, a switch has two conductive pieces, often metal, called *contacts* that touch to complete (make) a circuit, and separate to open (break) the circuit. The contact material is chosen for its resistance to corrosion, because most metals form insulating oxides that would prevent the switch from working. Contact materials are also chosen on the basis of electrical conductivity, hardness (resistance to abrasive wear), mechanical strength, low cost and low toxicity.

Sometimes the contacts are plated with noble metals. They may be designed to wipe against each other to clean off any contamination. Nonmetallic conductors, such as conductive plastic, are sometimes used.

Contact terminology

A pair of contacts is said to be "*closed*" when current can flow from one to the other. When the contacts are separated by an insulating air gap, they are said to be "*open*", and no current can flow between them at normal voltages.

Switches are classified according to the arrangement of their contacts in electronics. Electricians installing building wiring use different nomenclature, such as "*one-way*", "*two-way*", "*three-way*" and "*four-way*" switches, which have different meanings in North American and British cultural regions as described in the table below.

In a push-button type switch, in which the contacts remain in one state unless actuated, the contacts can either be *normally open* until closed by operation of the switch, or *normally closed* and opened by the switch action. A switch with both types of contact is called a *changeover switch*. These may be "*make-before-break*" which momentarily connect both circuits, or may be "*break-before-make*" which interrupts one circuit before closing the other.

The terms *pole* and *throw* are also used to describe switch contact variations. The number of "*poles*" is the number of separate circuits which are controlled by a switch. For example, a "*2-pole*" switch has two separate identical sets of contacts controlled by the same knob. The number of "*throws*" is the number of separate positions that the switch can adopt. A single-throw switch has one pair of contacts that can either be closed or open. A double-throw switch has a contact that can be connected to either of two other contacts; a triple-throw has a contact which can be connected to one of three other contacts, etc.

These terms give rise to abbreviations for the types of switch which are used in the electronics industry such as "*single-pole, single-throw*" (SPST) (the simplest type, "on or off") or "*single-pole, double-throw*" (SPDT), connecting either of two terminals to the common terminal. In electrical power wiring (i.e. House and building wiring by electricians) names generally involving the suffixed word *"-way"* are used; however, these terms differ between British and American English and the terms *two way* and *three way* are used in both with different meanings.

Power switching

When a switch is designed to switch significant power, the transitional state of the switch as well as the ability to stand continuous operating currents must be considered. When a switch is in the on state its resistance is near zero and very little power is dropped in the contacts; when a switch is in the off state its resistance is extremely high and even less power is dropped in the contacts. However when the switch is flicked the resistance must pass through a state where briefly a quarter (or worse if the load is not purely resistive) of the load's rated power is dropped in the switch.

For this reason, power switches intended to interrupt a load current have spring mechanisms to make sure the transition between on and off is as short as possible regardless of the speed at which the user moves the rocker.

Power switches usually come in two types. A momentary on-off switch (such as on a laser pointer) usually takes the form of a button and only closes the circuit when the button is depressed.

A regular on-off switch (such as on a flashlight) has a constant on-off feature. Dual-action switches incorporate both of these features.

Inductive loads

When a strongly inductive load such as an electric motor is switched off, the current cannot drop instantaneously to zero; a spark will jump across the opening contacts. Switches for inductive loads must be rated to handle these cases. The spark will cause electromagnetic interference if not suppressed; a snubber network of a resistor and capacitor in series will quell the spark.

Actuator

The moving part that applies the operating force to the contacts is called the *actuator*, and may be a toggle or *dolly*, a rocker, a push-button or any type of mechanical linkage *(see photo)*.

Biased switches

The momentary push-button switch is a type of biased switch. The most common type is a "push-to-make" (or normally-open or NO) switch, which makes contact when the button is pressed and breaks when the button is released. Each key of a computer keyboard, for example, is a normally-open "push-to-make" switch. A "push-to-break" (or normally-closed or NC) switch, on the other hand, breaks contact when the button is pressed and makes contact when it is released. An example of a push-to-break switch is a button used to release a door held open by an electromagnet.

Toggle switch

A toggle switch is a class of electrical switches that are manually actuated by a mechanical lever, handle, or rocking mechanism.

Toggle switches are available in many different styles and sizes, and are used in countless applications. Many are designed to provide, e.g., the simultaneous actuation of multiple sets of electrical contacts, or the control of large amounts of electric current or mains voltages.

The word "toggle" is a reference to a kind of mechanism or joint consisting of two arms, which are almost in line with each other, connected with an elbow-like pivot. However, the phrase "toggle switch" is applied to a switch with a short handle and a positive snap-action, whether it actually contains a toggle mechanism or not. Similarly, a switch where a definitive click is heard is called a "positive on-off switch.

Special types

Switches can be designed to respond to any type of mechanical stimulus: for example, vibration (the *trembler switch*), tilt, air pressure, fluid level (the *float switch*), the turning of a key (*key switch*), linear or rotary movement (the *limit switch* or *microswitch*), or presence of a magnetic field (the *reed switch*).

Mercury tilt switch

The mercury switch consists of a drop of mercury inside a glass bulb with 2 or more contacts. The two contacts pass through the glass, and are connected by the mercury when the bulb is tilted to make the mercury roll on to them.

This type of switch performs much better than the ball tilt switch, as the liquid metal connection is unaffected by dirt, debris and oxidation, it wets the contacts ensuring a very low resistance bounce-free connection, and movement and vibration do not produce a poor contact. These types can be used for precision works.

It can also be used where arcing is dangerous (such as in the presence of explosive vapor) as the entire unit is sealed.

Knife switch

Knife switches consist of a flat metal blade, hinged at one end, with an insulating handle for operation, and a fixed contact. When the switch is closed, current flows through the hinged pivot and blade and through the fixed contact. Such switches are usually not enclosed. The knife and contacts are typically formed of copper, steel, or brass, depending on the application. Fixed contacts may be backed up with a spring. Several parallel blades can be operated at the same time by one handle. The parts may be mounted on an insulating base with terminals for wiring, or may be directly bolted to an insulated switch board in a large assembly. Since the electrical contacts are exposed, the switch is used only where people cannot accidentally come in contact with the switch or where the voltage is so low as to not present a hazard.

Knife switches are made in many sizes from miniature switches to large devices used to carry thousands of amperes. In electrical transmission and distribution, gang-operated switches are used in circuits up to the highest voltages.

The disadvantages of the knife switch are the slow opening speed and the proximity of the operator to exposed live parts. Metal-enclosed safety disconnect switches are used for isolation of circuits in industrial power distribution. Sometimes spring-loaded auxiliary blades are fitted which momentarily carry the full current during opening, then quickly part to rapidly extinguish the arc.

Footswitch

A footswitch is a rugged switch which is operated by foot pressure. An example of use is for the control of an electric sewing machine. The foot control of an electric guitar is also a switch.

Reversing switch

A DPDT switch has six connections, but since polarity reversal is a very common usage of DPDT switches, some variations of the DPDT switch are internally wired specifically for polarity reversal. These crossover switches only have four terminals rather than six. Two of

the terminals are inputs and two are outputs. When connected to a battery or other DC source, the 4-way switch selects from either normal or reversed polarity. Such switches can also be used as intermediate switches in a multi-way switching system for control of lamps by more than two switches.

Light switches

In building wiring, light switches are installed at convenient locations to control lighting and occasionally other circuits. By use of multiple-pole switches, control of a lamp can be obtained from two or more places, such as the ends of a corridor or stairwell.

Electronic switches

A relay is an electrically operated switch. Many relays use an electromagnet to operate a switching mechanism mechanically, but other operating principles are also used.

The analogue switch uses two MOSFET transistors in a transmission gate arrangement as a switch that works much like a relay, with some advantages and several limitations compared to an electromechanical relay.

The power transistor(s) in a switching voltage regulator, such as a power supply unit, are used like a switch to alternately let power flow and block power from flowing.

Many people use metonymy to call a variety of devices "switches" that conceptually connect or disconnect signals and communication paths between electrical devices, analogous to the way mechanical switches connect and disconnect paths for electrons to flow between two conductors. Since the advent of digital logic in the 1950s, the term *switch* has spread to a variety of digital active devices such as transistors and logic gates whose function is to change their output state between two logic levels or connect different signal lines, and even computers, network switches, whose function is to provide connections between different ports in a computer network. The common feature of all these usages is they refer to devices that control a binary state: they are either *on* or *off*, *closed* or *open*, *connected* or *not connected*.

CHAPTER 5

TROBULESHOOTING AND PROBLEM SOLVING

Customer support inquires about software and network settings takes time and costs money, both for the customer and the vendor giving the support. Doing Troubleshooting and problem solving automatically can lower the need for inquires, and when needed, raise the quality of the contact for the customer.

The newest operating systems (Windows 7 and Mac OS X Snow Leopard) have good automatic error diagnostic facilities for generic network problems, but older versions like Windows XP has only limited capability.

The prototype(from which I took the text) look at the same problem, but also take it a step further and test the availability and connection quality to a vendor's specific network service.

For solving generic network problems and when having the newest version of the operating system the benefit of a separate tool is not great. But when a company requires more specific network testing related to their product, and possible retrieve other valuable information for the customer support about the computer setup, a custom developed tool has its benefits.

5.1.1. Introduction

Although computers have become easier to use and handle, the average user's technical knowledge of the computer has gone down as now also 'normal' people and not only technology 'geeks' use them. Thus many people need help when there is some malfunction as now everyone is not interested in knowing everything there is to know about a computer. The 'normal' people just want it to work. So this leads to need of customer support. Such support over telephone or via e-mail is common but it is tedious for the customer service

agent to understand the problem and correct it while only communicating by voice or text and not see the content of the computer screen.

It would be of great value to have a tool that can do routine troubleshooting and tell the user and the support personnel what the problem actually is.

5.1.2. Glossary

ARP (Address Resolution Protocol) – is a method for retrieving a host's link layer (hardware) address by providing the network layer (often IP) address.

DHCP (Dynamic Host Configuration Protocol) – Standard for automatically configure network settings on hosts on a network instead of doing it manually. If the DHCP server is unavailable when a host requests settings it will become somewhat like an orphaned child, one possible cause of loss of the internet connection.

DNS (Domain Name System) – Is a distributed database mapping names to IP addresses. If all of the configured DNS servers of a host fail it cannot contact other hosts (e.g. web servers) via domain names, only by directly providing the IP address.

ICMP (Internet Control Message Protocol) – Is a core protocol in the IP suite. It is not used for user/application data transfer, instead for error messages to help the network function. One notable function is to send a"ping", meaning to transmit an "echo request" (type 8 message) and expect an "echo reply" (type 0 message) in return. The sender measures the round trip time of this.

JNI (Java Native Interface) – Allows java code running in the java virtual machine to communicate with operating system specific native code, both to call and be called by.

JNLP (Java Network Launching Protocol) – specifies how a java web start application should launch. It defines a XML schema to this end.

JRE (Java Runtime Environment) – is composed of the java virtual machine (JVM) and class libraries. JVM executes java bytecode and the class libraries implement the java application programming interface (API).

Native code – Opposed to Java's platform independent source code and byte code (the code that the JRE understands) native code mean source code and compiled code aiming at a specific platform (operating system and instruction set architecture).

NetBIOS (Network Basic Input/Output System) – is an API aimed at providing services on a network. There are several adaptions using different network protocols as carrier.

Operator – A company that has a branded poker client developed by bwin and have their own 1st line customer service but which infrastructure in the poker network is hosted on the Ongame network. There are around 25 operators not counting bwin's own brand on the network.

Test – When referring to the prototype it means a plug-in that does some investigation of the computer or related things, e.g. the network connection test.

Tool – When referring to the prototype it means a plug-in that does some "work" without the intent to investigate anything.

WINS (Windows Internet Name Service) – is a Microsoft implementation of the name service for computer names offered by NetBIOS.

Problem Description and Rationale

Internet service provider companies can monitor disturbances in their own network but for laymen this kind of information is not available. To correct network problems it takes often quite bit of experience and most users never get the time to learn about this topic. But unlike

many other fields the computers has a very good for doing automatic repair of its own – the computer can be controlled by programs and not only humans.

Network diagnostic is one field where many repairs can be performed by the computer but some kind of repairs need the ability to affect objects in the physical world. Here an interaction is required to guide the user. Functionality for more-or-less automatic network diagnostic, to find and correct network problems, is already implemented in modern versions of desktop and laptop operating systems like Microsoft Windows and Apple Mac OS X. But it is not always the case that they provide enough information about the problem

5.1.3 Goals

- To investigate how network connection problems can be diagnosed and what facilities are provided by today's operating systems.
- Find out what problems that is common among the customers, both network-wise and others.
- Identify the most important areas of functionality for a diagnostic tool running on the customer's computer.
- To develop a prototype of diagnostic program that is aimed to the average customer for diagnosing network problems. A second aim is to incorporate more functionality from suggestions in previous surveys. The application should be implemented with Java as an application that can be retrieved via Java web start. The focus is on network connection problems, other possible suggested needs are secondary.

5.1.4 Delimitations

The prototype should be made to help inexperienced users with network related problems that they may encounter. The aim is not at network professionals and their methods and tools for solving complex network problems. Only automatic ("press a button") diagnostic is required of the prototype, no need to attempt to "repair" a computer's perhaps faulty network configuration settings.

Tools in Today's Operating Systems

Windows XP

Windows XP can only inform the user that the cable is not connected (or that the interface is not associated with a wireless network) or that it has "limited or no connectivity", often reported when it has not gotten an IP address from a DHCP server.

Microsoft introduced the "repair" function with Windows XP. When a network connection is not working it does a number of things to try to fix the problem. But it does not present possible causes if it fails, it only tells the user at which point it failed. These are the steps that are performed while repairing:

- Transmitting a DHCP broadcast message asking for address renewal.
- Note that it does not release the current address first as that could worsen the situation if the computer does not receive a DHCP lease response from any server.
- Flushes the ARP cache, which is the cache of previous learned mappings between IP addresses and network interface addresses (MACaddresses).
- Flushes the NetBIOS name cache and reloads static entries.
- Re-register the computer with a WINS server (if such exists).
- Flushes the DNS cache, the mappings between IP addresses and fully qualified domain names.
- Re-register the computer with a (dynamic) DNS server.

There is also another tool from Microsoft called "Network Diagnostics for Windows XP" released for free in 2006. This tool purports to do the following tests:

- Test the IP address configuration.
- Test the default gateway.
- Test Winsock.
- Test DNS.

- Test software firewall.
- Validate the internet connection.

After testing it can do some repairing actions or suggest to the user what to do. It provides a log file with much information about what has been tested and the results.

However it has at least one drawback. When using two network adapters it stopped testing and just directed the user to customer support with the warning message "This machine has more than one Ethernet or more than one Wireless adapter". I think that is an unnecessary limitation on the software.

The introduction of the repair function was a significant improvement compared to not having any. But it lacks functionality with which to inform the user about possible steps to take if it is unsuccessful in the repair process.

The separate tool had some of this functionality. Also it is not system-wide; the user has to select a specific network connection (which needs to be enabled first). As there are some strange "network connections" that sometimes are shown in the network connections list (i.e. IEEE 1394 and "Microsoft TV/Video connection") the user might select the wrong one when trying to repair it. This level of technical competence should not be expected of the average user.

Windows Vista

Vista has a slightly more refined network diagnostic suite. The user does not need to begin by selecting a specific network connection; instead there is a "diagnose and repair" link in the "network and sharing center". It tries to find the problem and repair it. If correcting the connection state requires a physical presence (e.g. "connect the network cable" or "restart the home router") the software gives appropriate instructions to the user.

Windows 7

In Windows 7 the diagnosis of network problems has been reworked. A "Windows troubleshooting platform" has been developed to support many different tests (not only network related) including option to extend it with self-made tests.

In the troubleshooting guide the user has to select which test to run, but the choices are easy to understand. Namely, the user can choose from the following tests to run from the "network and sharing center" (where only the network related tests are displayed):

- Internet Connection – tries to reach Microsoft's web page or a page provided by the user.
- Shared Folders – the user provides some network file share where a problem exists that requires software guided diagnosis.
- Home Group – troubleshoot Microsoft's improved ad hoc network service in windows 7 that gives streamlined access to resources on other windows 7 computers at home.
- Network Adapter – if nothing is working this is the guide to start with, it diagnoses the network adapter's settings etc.
- Incoming Connections – checks things regarding the software firewall so programs or services can work.
- Connection to a Workspace Using DirectAccess – DirectAccess (written without a space between the words) is a VPN technology based on IPv6.

Mac OS X – Leopard & Snow Leopard

OS X has a similar network diagnostic as Windows 7. The user has to select which network adapter to test and then it reports status for a few different items that are labeled Ethernet, Network settings, ISP (internet service provider), Internet and lastly, Server. When testing wireless connection the Ethernet label is replaced with Airport and Airport settings.

Ubuntu Linux 8.10 – Intrepid Ibex

There does not seem to be any automatic network diagnostic utility shipped with Ubuntu. There is a network tool that is just a GUI frontend to traditional programs like ping, traceroute, nslookup etc. The operating system's help files contain some troubleshooting guides for networking.

Possibilities to Diagnose From a Single Network Host

- First step in diagnosing a connection is to see if there actually is any connection at all, thus testing *reachability*. But this is not sufficient in itself; the user most probably already knows that there is a problem. To help the user alleviate the problem the program, needs to provide information about the (probable) location or cause. The key points are:
  - Physical connection – is the network cable plugged in or is the wireless network card associated with a wireless network?
  - Presence on the network – does the machine have an IP address at all?
  - Inter network capability – can the default gateway be reached?
  - Vicinity – try different sites on internet, if none of them works the problem is probably close to the default gateway, like in the internet service provider's network.
  - Destination – can the on game network's servers be reached?
  - Name resolution – perhaps the DNS server (-s) used by the client has a problem with resolving the name of the game servers. This problem differs from the others, since in this case there is no problem with the direct path between the client's computer and operator.
- After the above established criteria are verified to work, the quality of the connection should be determined. Factors that play a role in the quality of a connection are:

  • Message loss – if a data packet gets lost en route it can sometimes be detected with an ICMP message (type 3 destination unreachable and type 12 time exceeded), but still it has to be retransmitted and then the measure of quality goes down and the total time for the message to reach the server increases. If a program is not written robustly message losses can make it work very poorly.

• Round trip time – how long time does it take for a message to reach the server and a response to travel back to the sender under normal working conditions? If it takes too long time the program will feel unresponsive.

• Bandwidth – how many bytes of useful data can be sent and received within a given timeframe?

• Message integrity – the message itself should not be altered en route. This problem is not so common thanks to automatic error detection, but when a message gets corrupted the error can-not be corrected but instead will be retransmitted. In some situations this situation can be misinterpreted as a message loss problem.

Possibilities to Diagnose Between Two Network Hosts

Internet service providers can passively monitor the amount of traffic that passes through their routers and thus find a bottleneck in their network, but such information is not available to the general public. So to measure the available bandwidth across the internet two hosts need to cooperate in some way. It can be as simple as downloading a large file from a server to see the speed of the path between them. But this is generally a bad idea for at least three reasons:

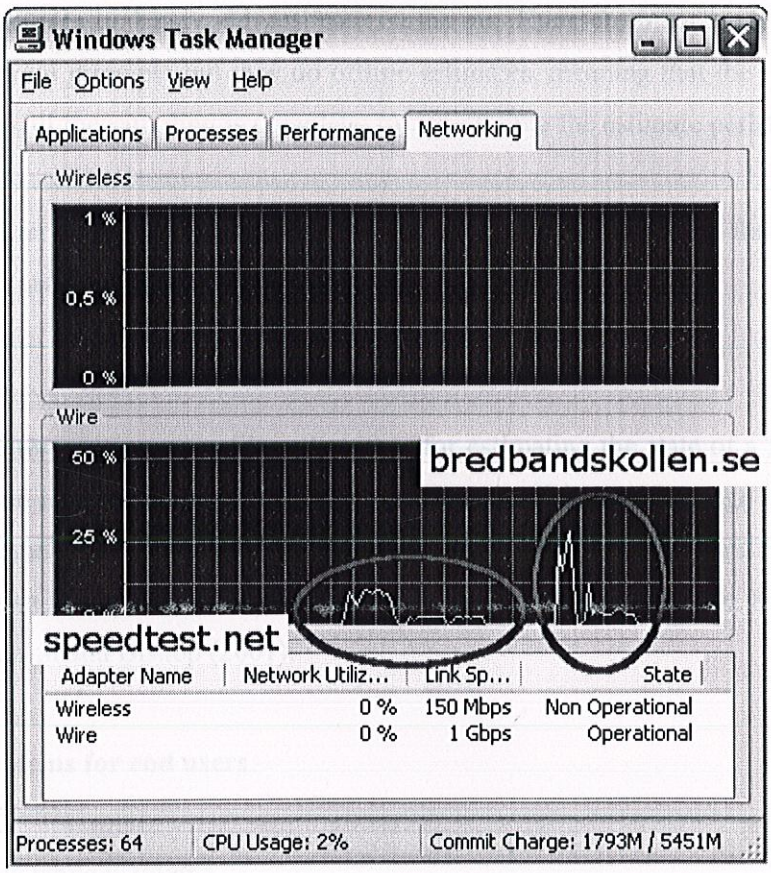
- If either the client or the server pays per transferred data it can cost a lot in the end, especially for the server owner if a lot of clients test their bandwidth in this way.

- The server can become congested by all the bandwidth testers, so the bottleneck is always at the server. The goal is to measure the speed on the path between the hosts, not to create a bottleneck at one of them.

- A client does not want to fill (possible all) their available bandwidth for a test, it creates disturbances, e.g. using streamed real-time media at the same time would probably not work (unless some quality of service is employed.

It seems common internet sites for speed testing employ the method of filling the available bandwidth and measuring per second throughput. In figure, the result is shown from running the bandwidth test on the internet site www.speedtest.net and on www.bredbandskollen.se. Notice that the test is conducted on a gigabit Ethernet connection, so even though the highest usage was only 25% that is 250 Mbps.

Figure



Thus we want a better way to measure the available bandwidth. Eklin et al. (2006) proposed an algorithm they call "BART" but there are others of this kind, e.g. pathChirp

BART (Bandwidth Available in Real-Time)

BART is a method that analyzes the currently available (unused) bandwidth in a packet-switched network. The algorithm works by sending a series of network packets with a known time delay between each packet (the rate). It tries to find the minimum amount of time

separation where packets arrive at the same time separation as they were sent. If the packets arrive at a lower rate it means that they have experienced traffic congestion on some hop en route.

The hop that had the biggest congestion is the overall bottleneck of the whole route. There amount of packets that needs to be sent to get a good estimate are few hence the BART algorithm solves both previously stated problems about bandwidth testing.

The algorithm analyzes the collected data after each "packet train" has been completely received and updates its bandwidth estimate on the go. There are other methods that works by the packet train principle but they do offline estimates, meaning that the results are only presented when all the sampling is complete, by which time the estimate perhaps can already by outdated.

The Kalman filter algorithm is used in BART to get a nice average of all the measurements (even though it can be of a very short time scale).

Kalman Filter

The Kalman filter is a recursive filter-algorithm for estimating the state of a linear dynamic system based on a series of noisy measurements. It is used in a wide range of applications, usually very some sensor reading of the real world is not totally accurate and the process that is measured also has a degree of uncertainty. A good first guide to the Kalman filter is provided by Welch and Bishop (2001).

Common problems for end users

1. Disconnected while playing

The single biggest problem occurs when an end user gets disconnected for an unknown reason while entered in a session with the server. The difficulty is to know where the error occurred, if it was the end users internet connection causing the problem or at the servers

2. Location of log files in the file system

The client stores its log files in a location that is the correct path for this kind of information according to the design guidelines on a Windows operating system, but this directory is hard to find for a novice customer. The folder location in question is the local application data folder of the windows user's profile. On a normal Windows XP installation (Vista and 7 stores this information somewhat differently) this folder is located in

"C:\Documents and Settings\<username>\Local Settings\Application Data"

but it may differ as an administrator can change this path. On top of this the last two folders, "Local Settings\Application", have the hidden folder file system attribute. Many users do not show hidden files and folders in Windows Explorer.

3. Log files content and size

Currently it seems that the content of the log files is not so helpful when trying to find a solution to a disconnection problem. Partly because of lack of information to determine the error, and partly that parts of it can be hard to understand for people that have not developed the system (such as the customer service agents).

E.g. on Windows the error codes given by the windows sockets (winsock) object when a network socket has been abruptly terminated give quite good information about the loss of connection, but this is not recorded.

The logger produces quite large log files in terms of bytes, this makes the log files hard to handle as it is normal procedure for the customer service agent to request the user to send the log file, but some e-mail systems can have a limit of a few megabytes for attachments.

Graphical User Interface at end user

As the target audience for the prototype is a novice to average skilled computer user it has to be simple and obvious what can be done with it. It should not be required of the user to study a user manual; instead everything needs to be intuitive.

As it is built as a plug-in architecture with distinct tests and tools it is natural to create a list of them in the main window, from where the user can start a test or a tool.

Plug-ins provides their graphical user interface; it is not something made available through the core program.

Distribution model

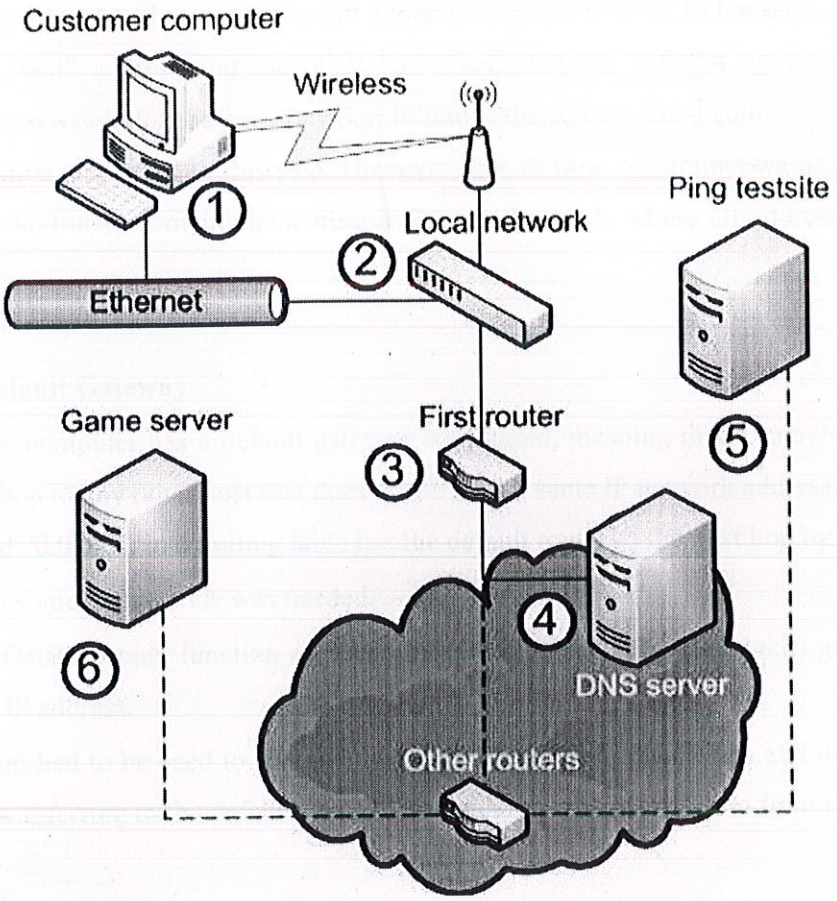
It's a Java application. It should be in a single JAR-file and when signed (applied a digital signature) it can be distributed via Java web start and do privileged operations, not being sandboxed like ordinary applets or web start applications.

Java web start is a distribution method where a HTTP server hosts a .jnlp file with a custom internet media type (MIME type) called application/xjava-jnlp-file. When requested by a web browser (typed into the address bar or referred via a link) it will automatically check if the right version of the Java runtime environment is installed. If not, it offers the user to install it.

After this it downloads the application and installs it. The advantage is that the Java runtime environment can be "bundled" with the application; one does not have to request the user to visit some other web site to download a Java runtime environment.

Network Diagnostic Test

Most parts of this test require functionality that only can be provided by the custom made native code for each platform, here each stage is explained and a brief mention about with what it was implemented on each platform. Each stage's corresponds with the numbers in order to make it easier to follow the text.



Step 1 – Medium Present

Check if the network cable is plugged in (both ends of it) or if it is a wireless network card it checks if it has been associated with any network.

Implementation:

On Windows the native code uses isNetworkAlive function exported from the System Event Notification Service (Sensapi.h).

Step 2 – Valid IP Address

Check if the computer has any IP address assigned to it, no matter if it is static or dynamic, except the loopback address or a self-assigned dynamic address.

The Java function InetAddress.getLocalHost() is used although it only can return 1 address, even if the computer has several assigned. This leads some problems E.g. the computer has two network interface cards, a wired and a wireless network. It can be that both have their

medium present and both use dynamic IP address assignment (DHCP) but only one has been assigned a "real" address and the other has a self-assigned 169.254.x.y address. In this situation it is possible for the Java function to return the self-assigned address. This can give a false negative test result in this step. However, due to time constraints we have chosen to use this method instead of implementing it from native code where all addresses could be retrieved.

Step 3 – Default Gateway

Check if the computer has a default gateway configured, meaning that it knows the address of the first hop to any other host that does not share the same IP network address. The default route, called "0.0.0.0", in a routing table has the default router as the next hop by definition.

For this test some native code was needed.

Windows: GetBestRoute function exported in the IP helper API (iphlpapi.h) gives the best route to an IP address.

OS X: Sysctl had to be used to retrieve the routing table from the system and then manually find the row referring to the default route and extract the next hop column from that table.

Step 4 – Name Server

Check if the computer has any IP address to some DNS server and test if it responds.

Native code for this:

Windows: GetNetworkParams function exported in the IP helper API (iphlpapi.h) return various parameter about the network configuration, among those is a linked list with DNS servers.

The shell command scutil is executed which can list DNS servers and the output is grabbed by the application.

The actual test of the server is straight forward. A datagram package is constructed, that conforms to the DNS specification, and sent via UDP. It is a simple query and as the content of the response (what the name resolved to) is not interesting, it only checks that the right ID marker exists in the returned package.

The rationale for testing reachability with the DNS protocol instead of ICMP echo requests is that ICMP might be blocked, or the actual DNS server software is not running but the server hardware and operating system works at the given time.

As Windows and OS X are very similar in terms of socket programming (heritage from BSD sockets) and since Java doesn't support unsigned data types (which makes it harder to construct the package where one needs precise control over which bits are set and which are not) the implementation of this is done in native C code. Both platforms use the same methods and when the code differs slightly preprocessor directives are used to select between Windows and OS X code.

Java has methods for resolving names built into the standard library. Those records can have been cached previously. The only reliable way to really know that the request was sent to the server is to do it "by hand".

Step 5 – Connection to the Internet

This test sends ICMP echo requests ("ping") to a number of hosts on the internet. The addresses' names imply that they are intended for anyone to use as ping sites.

Host address name Location

ping.sunet.se Sweden

ping.port80.se Sweden

ping.bahnhof.se Sweden

Step 6 – Connection to Server

Check if the program can connect to the servers. As ICMP echo request can be blocked a stream socket (over TCP) is opened to the server software that the client usually connects to (with the same port). As TCP stream sockets do handshaking this tool will know that the server is reachable. After the handshake is completed the stream socket is closed down, no testing using the protocol (e.g. login procedure) is performed.

The Quality of a Network Connection

The result from the above steps in the test mostly makes sense to a professional. To help a novice user to understand all this the combined result is presented with a "score" (not a competition "score" but a judgment) which is presented both in text and symbolically, on a scale from green to red depending on the score.

The score is calculated from all steps in the network connection test. Steps 1 through 4 have results that are Boolean, either it works or not. Thus they can terminate the connection test immediately. Steps 5 and 6 reduce the score, but the testing process continues.

The actual score is maintained internally as a double precision floating point value. It starts out as 1.0 and decreases with falling connection quality. The final quality judgment is based on this number; it is mapped to a textual score.

The levels are listed in Table

Quality levels and their required minimum quality value

Label	Minimum
Perfect	0.95
Good	0.85
Acceptable	0.75
Poor	0.55
Unusable	0.35
No connection	N/A

Available Bandwidth Test

The bandwidth test is not integrated with the network connection test due to uncertainty about how well it would work – if it does not measure up correctly there is no reason to have it in the multi-step network connection test. Instead it is listed as a separate test on the main menu of the application.

The goal of this part of the project, implementing an efficient bandwidth test, was only to see if we could do it and test such a method, not to integrate it coherently with the rest of the prototype, which is a much bigger task. It was sufficient as it is a prototype and a real developed application would need a completely rewritten server part anyway as it was not meant to support more than one client.

Future development

This section lists smaller things that were not completed for the prototype but should be if the code will be developed towards a final product.

For the program to run correctly in 64-bit JRE it also has to include 64-bit versions of dynamic linked library files for each such platform.

Currently only 32-bit is used.

- The OS X has support for detecting if the network medium is present or not, but I did not found out how to use it until later when I had already skipped that feature for OS X. The information can be retrieved via IOCTL system call. That is how ifconfig shell command does it. Studying the nearly uncommented open source code for ifconfig and understanding it probably gives the solution.
- According to a web page, the solution to the problem with being unable to delete the native dynamic loaded library file is to write a custom Java class loader, use it to load the native library. Later at shutdown all references to objects of the native class should be dropped and also to this class loader. Running the garbage collector twice after that was claimed to remove the file lock and make deletion possible.
- Use native libraries to get all IP addresses for all network interfaces. Then keep track of them and when the default router address is found that IP address should be checked to be within the same network address part of at least one of the previously found host IP addresses.
- This program is not a shell script and thus should not rely on executing system commands and parsing the output. One such case where that rule is broken is for finding the name servers (DNS) on OS X. The scutil command is used, but the

System Configuration framework API, SCDynamicStore, can be used to do queries to the system about various runtime parameters instead.

- Optimize some network tests to be done in parallel. E.g. the game server test is not sensitive in measuring time, it just needs to open a number of stream socket sessions which are totally independent of each other. If there will be a great number of servers to check this can cut down time for tests to complete, in the case where some servers timeouts.
- The user interface of the start window was intended to contain a list of available test & tools. But the list is quite short (two enabled and two disabled mockup buttons) so it could be expanded to have all shown at the same time instead of a scrollable surface.
- For a live release it needs a possibility to be branded to different operators on the poker network, perhaps by detecting which operator's client is installed and show that brand.
- Fine-tune the bandwidth testing so it produces more stable and reliable results.
- Improve the router testing in the network test by doing a trace route to some location and thus finding out the second router (hop). Often people have a home router. The next hop is often at the ISP and thus it can be used to detect if a user has a local problem with the ISP.
- Verify the content of the DNS response to make the test not only detecting that the DNS responds but it also has contact with the rest of the DNS infrastructure (i.e. it can connect to other DNS servers.

Prototype Program

Network Connection Test

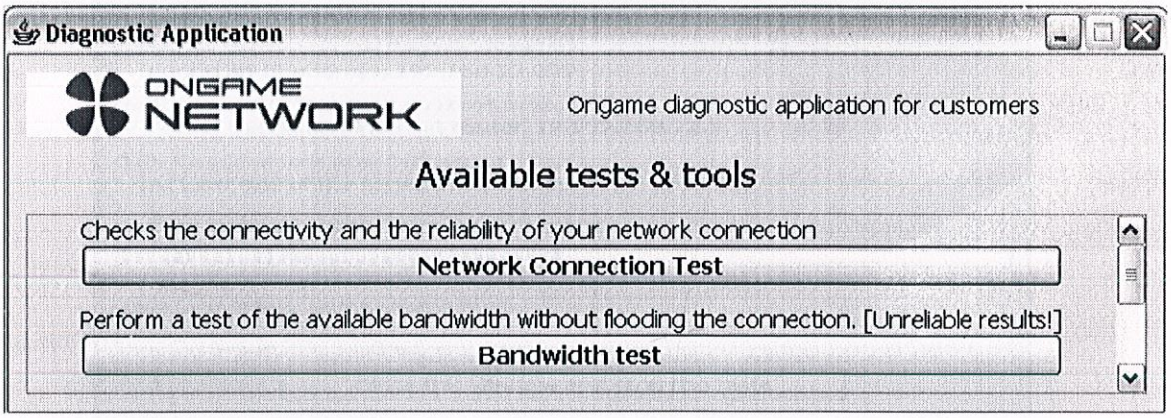
Albeit it could produce inconsequent results as mentioned in the previous section about mismatches of host IP and gateway IP addresses I think the network connection test works well. But the graphical interface is not so well designed, lacking human computer interaction user perspective thinking.

Bandwidth Test

This test was not completed as I could not get my implementation to produce any reliable output. That is because of the problem with getting it fine-tuned to work well and lack of a big real world test to verify that it works. It is possible to complete it but for any commercial release it should be considered that it is intellectual property of another company, which perhaps can license it to bwin.

Graphical User Interface

As a mockup the Ongame network logo is used in the main window instead of any operator's brand, no need for that in a prototype.

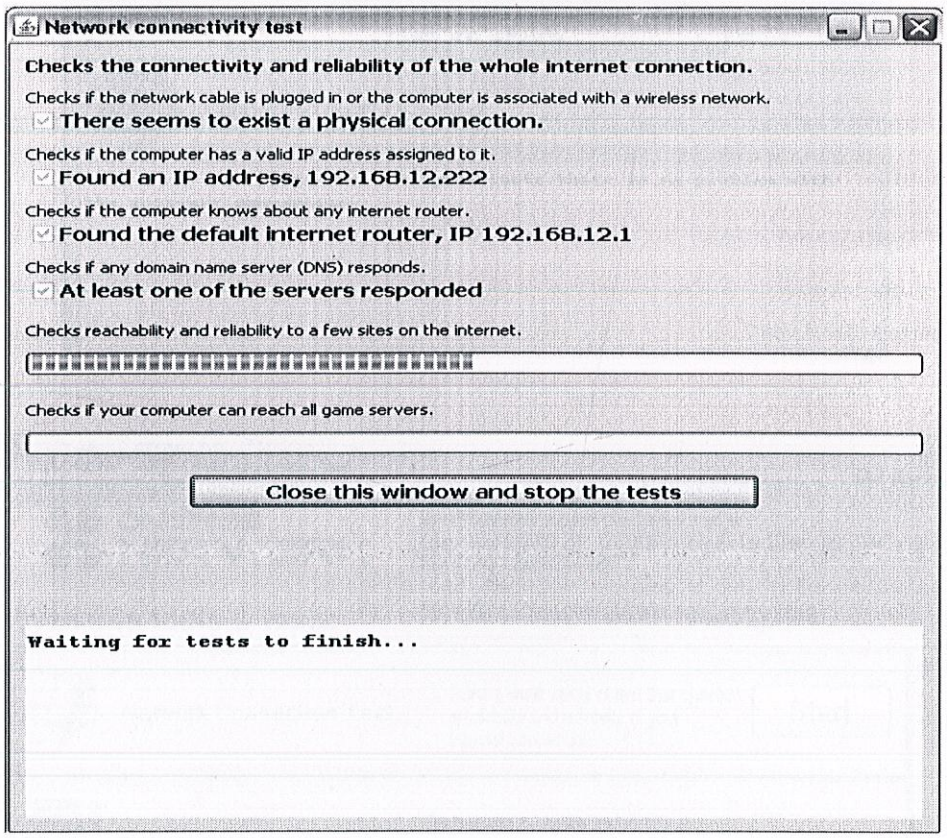


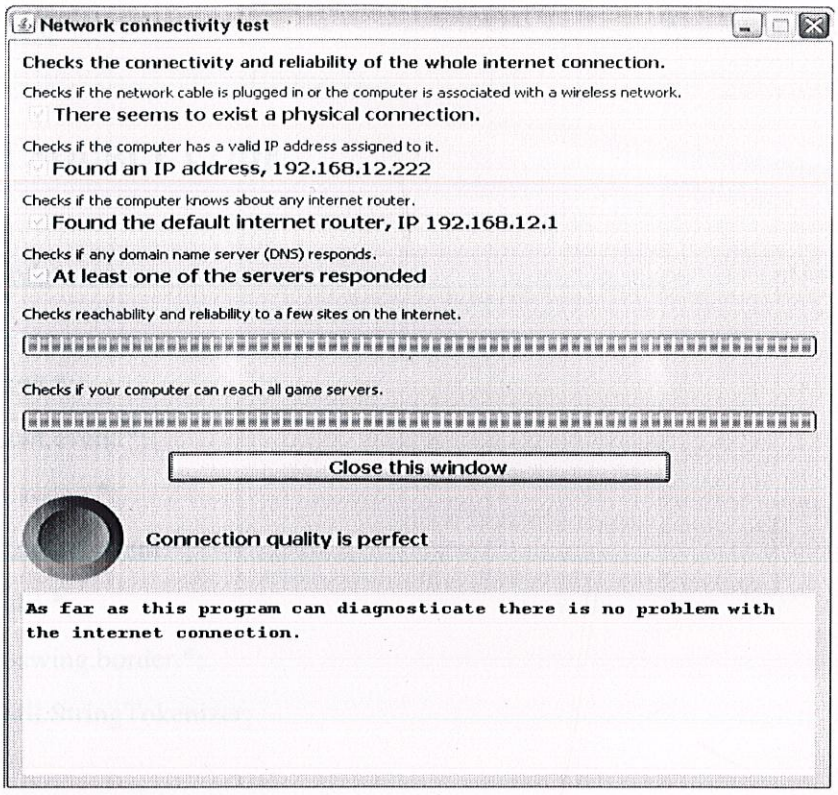
Clicking the button labeled "Network Connection Test" in takes the user to the screen shown in Figure where the actual test starts right away. The first three tests can be done in an instant as they are simple lookup requests to the local computers configuration. The fourth test is also usually done without seeing it, as the DNS server is often very close (in context of time) to the host. The fifth test of reachability for internet ping sites takes a longer time as the program waits a little while between each ICMP echo response and the next ICMP echo request.

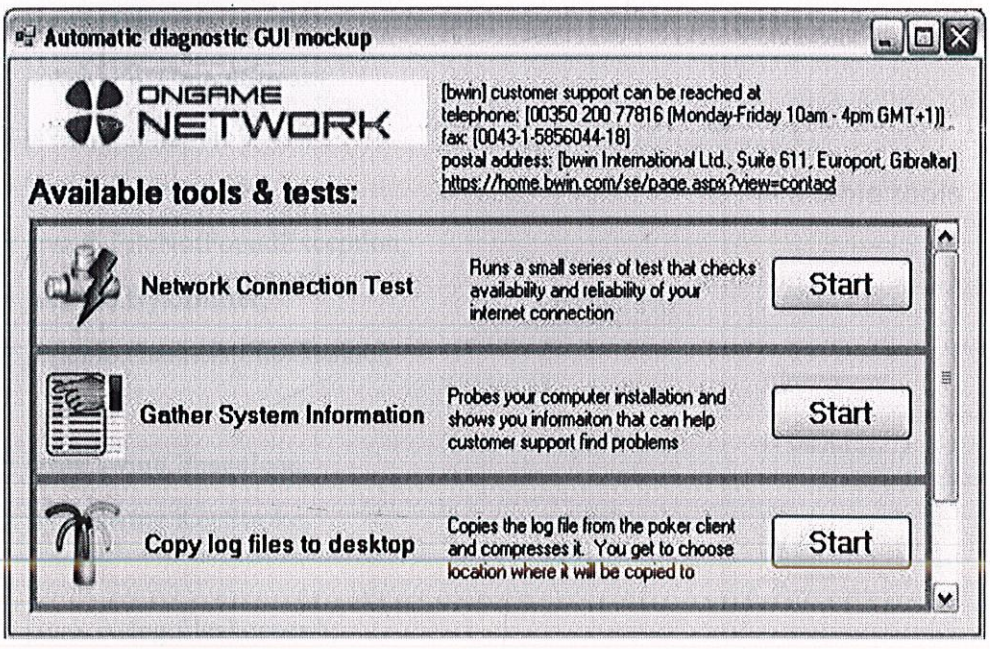
If any of the basic tests fails, the text area in the bottom of the window will show a (hopefully helpful) description of the probable problem.

When the test finishes a "button" (a symbolic icon, not interactive GUI element) is shown, together with a short text about the connection quality.

The button color reflects the result of the quality test, as people are used to interpreting signs using colors. Red (no connection) is often used to signal a warning and green (perfect connection) is common to signal that something is okay. The button's colors spans from red to green via yellow in six steps.




Network connectivity test
Checks the connectivity and reliability of the whole internet connection.
Checks if the network cable is plugged in or the computer is associated with a wireless network.
There seems to exist a physical connection.
Checks if the computer has a valid IP address assigned to it.
Found an IP address, 192.168.12.222
Checks if the computer knows about any internet router.
Found the default internet router, IP 192.168.12.1
Checks if any domain name server (DNS) responds.
At least one of the servers responded
Checks reachability and reliability to a few sites on the internet.
Checks if your computer can reach all game servers.
Close this window
Connection quality is perfect
As far as this program can diagnosticate there is no problem with the internet connection.


Automatic diagnostic GUI mockup
ONGAME NETWORK
[bwin] customer support can be reached at
telephone: [00350 200 77816 (Monday-Friday 10am - 4pm GMT+1)]
fax: [0043-1-5856044-18]
postal address: [bwin International Ltd., Suite 611, Europort, Gibraltar]
https://home.bwin.com/se/page.aspx?view=contact
Available tools & tests:
Network Connection Test
Runs a small series of test that checks availability and reliability of your internet connection
Start
Gather System Information
Probes your computer installation and shows you informaiton that can help customer support find problems
Start
Copy log files to desktop
Copies the log file from the poker client and compresses it. You get to choose location where it will be copied to
Start

CHAPTER 6

PROJECT SOURCE CODE

6.1.1 GUI Code

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.*;
import javax.swing.border.*;
import java.util.StringTokenizer;

import java.io.*;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;

import javax.swing.ImageIcon;
import javax.swing.KeyStroke;

import javax.swing.filechooser.*;

class writefilemulnewdiag extends JFrame
```

```
{
    public writefilemulnewdiag(String st)
    {
        String dirname="E:/Pro/";
        String filename="diag_"+st+".txt";
        String[] tests={"New Diagnosis Created Successfully."};
        File afile=new File(dirname,filename);
        FileOutputStream o=null;
        try
        {
            o=new FileOutputStream(afile,true);
        }
        catch(FileNotFoundException e)
        {
            e.printStackTrace(System.err);
            System.exit(1);
        }
        FileChannel oc=o.getChannel();

        int maxlength=0;
        for(String s:tests)
        {
            if(maxlength<s.length())
            maxlength=s.length();
        }

        ByteBuffer buf=ByteBuffer.allocate(2*maxlength+4);
        try
        {
```

```
                for(String s:tests)
                {
                        buf.putInt(s.length()).asCharBuffer().put(s);
                        buf.position(buf.position()+2*s.length()).flip();
                        oc.write(buf);
                        buf.clear();
                }
                o.close();
            }
            catch(IOException e)
            {
                e.printStackTrace(System.err);
                System.exit(1);
            }
        }
}

class writefilemulnewdiagpro extends JFrame
{
    public writefilemulnewdiagpro(String st)
    {
        String dirname="E:/Pro/";
        String filename="diagpro_"+st+".txt";
        String[] tests={"New Diagnosis Profile Created Successfully."};
        File afile=new File(dirname,filename);
        FileOutputStream o=null;
        try
        {
            o=new FileOutputStream(afile,true);
```

```
}
catch(FileNotFoundException e)
{
        e.printStackTrace(System.err);
        System.exit(1);
}
FileChannel oc=o.getChannel();

int maxlength=0;
for(String s:tests)
{
        if(maxlength<s.length())
        maxlength=s.length();
}

ByteBuffer buf=ByteBuffer.allocate(2*maxlength+4);
try
{
        for(String s:tests)
        {
                buf.putInt(s.length()).asCharBuffer().put(s);
                buf.position(buf.position()+2*s.length()).flip();
                oc.write(buf);
                buf.clear();
        }
        o.close();
}
catch(IOException e)
{
```

```
                    e.printStackTrace(System.err);
                    System.exit(1);
                }
        }
}

class writefilemuldiagreport extends JFrame
{
        public writefilemuldiagreport(String st)
        {
                String dirname="E:/Pro/";
                String filename="diagreport_"+st+".txt";
                String[] tests={"New Diagnosis Report Created Successfully."};
                File afile=new File(dirname,filename);
                FileOutputStream o=null;
                try
                {
                        o=new FileOutputStream(afile,true);
                }
                catch(FileNotFoundException e)
                {
                        e.printStackTrace(System.err);
                        System.exit(1);
                }
                FileChannel oc=o.getChannel();

                int maxlength=0;
                for(String s:tests)
                {
```

```
                if(maxlength<s.length())
                maxlength=s.length();
            }

            ByteBuffer buf=ByteBuffer.allocate(2*maxlength+4);
            try
            {
                for(String s:tests)
                {
                    buf.putInt(s.length()).asCharBuffer().put(s);
                    buf.position(buf.position()+2*s.length()).flip();
                    oc.write(buf);
                    buf.clear();
                }
                o.close();
            }
            catch(IOException e)
            {
                e.printStackTrace(System.err);
                System.exit(1);
            }
        }
}

class readfile extends JFrame
{

    public readfile(String x)
    {
```

```
String filename=x;
FileInputStream in=null;
try
{
        in=new FileInputStream(filename);
}
catch(FileNotFoundException e)
{
        e.printStackTrace(System.err);
        System.exit(1);
}

FileChannel ic=in.getChannel();
ByteBuffer buf=ByteBuffer.allocate(1000);
try
{
        while(ic.read(buf)!=-1)
        {

System.out.println(((ByteBuffer)(buf.flip())).asCharBuffer().toString());
                buf.clear();
        }
        in.close();
}
catch(IOException e)
{
        e.printStackTrace(System.err);
        System.exit(1);
}
```

```
            //System.exit(0);
        }
}

class filechooser extends JFrame implements ActionListener
{
        JFileChooser ch=new JFileChooser();
        JButton b=new JButton("Select File");
        JTextField t=new JTextField(30);
        public filechooser()
        {
                super();
                setTitle("Open");
                Container cp=getContentPane();
                cp.setLayout(new FlowLayout());
                cp.add(b);
                cp.add(t);
                b.addActionListener(this);
        }
        public void actionPerformed(ActionEvent e)
        {
                int r=ch.showOpenDialog(null);
                File f=ch.getSelectedFile();
                if(r==JFileChooser.APPROVE_OPTION)
                {
                        t.setText("Selected : "+f.getPath());
                        JFrame xx=new readfile(f.getPath());
                }
                else if(r==JFileChooser.CANCEL_OPTION)
```

```
            {
                dispose();
            }
        }
}

class pbthread extends Thread
{
    JProgressBar pbar;
    pbthread(JProgressBar pb)
    {
        pbar=pb;
    }
    public void run()
    {
        int min=0;
        int max=50;
        pbar.setMinimum(min);
        pbar.setMaximum(max);
        pbar.setValue(0);
        for(int i=min;i<=max;i++)
        {
            pbar.setValue(i);
            try
            {
                sleep(500);
            }
            catch(InterruptedException e)
            {}
```

```
            }
        }
}

class prob extends JFrame implements ActionListener
 {
        JProgressBar progress;
        JButton button;
        JLabel label1;
        JPanel topPanel;
        public prob()
        {
                setTitle("Testing Arena");
                setSize(310,130);
                setBackground(Color.gray);

                topPanel=new JPanel();
                topPanel.setPreferredSize(new Dimension(310,130));
                getContentPane().add(topPanel);


                label1=new JLabel("Press Start to begin Testing...");
                topPanel.add(label1);

                progress=new JProgressBar();
                progress.setPreferredSize(new Dimension(300,20));
                progress.setMinimum(0);
                progress.setMaximum(100);
                progress.setValue(0);
```

```
        progress.setBounds(20,35,260,20);
        progress.setBackground(Color.white);
        progress.setForeground(Color.blue);
        topPanel.add(progress);

        button=new JButton("Start");
        topPanel.add(button);
        button.addActionListener(this);
}

public void actionPerformed(ActionEvent e)
{
        if(e.getSource()==button)
        {
                button.setEnabled( false );
                for(int i=1;i<101;i++)
                {
                        Bogus(i);
                        label1.setText(i+" % Completed");
                        Rectangle labelRect=label1.getBounds();
                        labelRect.x=0;
                        labelRect.y=0;
                        label1.paintImmediately(labelRect);

                        progress.setValue(i);
                        Rectangle progressRect=progress.getBounds();
                        progressRect.x=0;
                        progressRect.y=0;
                        progress.paintImmediately(progressRect);
```

```
                }
            }
            label1.setText("Test(s) Completed. Check Report(s)");
            topPanel.setBackground(Color.yellow);
            String strr=JOptionPane.showInputDialog(null,"Enter the Diagnosis Report
Name"," ",JOptionPane.PLAIN_MESSAGE);
            JFrame ff=new writefilemuldiagreport(strr);
            JOptionPane.showMessageDialog(null,"Diagnosis Report saved
!!","Diagnosis",JOptionPane.INFORMATION_MESSAGE);
            dispose();
        }

        public void Bogus(int i)
        {
            Random random=new Random(i);
            for(int iV=0;iV<random.nextFloat()*100000;iV++)
            {
                System.out.println("iValue="+iV);
            }
        }
}

class loadFileDiag extends JFrame
{
        JTextArea t=new JTextArea();
        loadFileDiag()
        {
            setTitle("Diagnosis");
            Container content=getContentPane();
```

```
                JScrollPane sPane=new JScrollPane (t);
                FileDialog fd=new FileDialog(this, "Load File", FileDialog.LOAD );
                fd.show();
                //fd.setFile      ("*.java;*.txt");
                String file=fd.getDirectory();
                if(file == null)
                return;

                setBackground(Color.gray);
                content.setPreferredSize(new Dimension(600,400));
                content.add(t);

                try
                {
                        FileInputStream fis=new FileInputStream(fd.getFile());
                        byte[] data=new byte[fis.available()];
                        fis.read(data);
                        t.setText(new String(data));
                }
                catch (IOException e)
                {
                        t.setText("Could not load file...");
                }
        }
}

class loadFileDiagPro extends JFrame
{
        JTextArea t=new JTextArea();
```

```
loadFileDiagPro()
{
        setTitle("Diagnosis Profile");
        Container content=getContentPane();
        JScrollPane sPane=new JScrollPane (t);
        FileDialog fd=new FileDialog(this, "Load File", FileDialog.LOAD );
        fd.show();
        //fd.setFile      ("*.java;*.txt");
        String file=fd.getDirectory();
        if(file == null)
        return;

        setBackground(Color.gray);
        content.setPreferredSize(new Dimension(600,400));
        content.add(t);

        try
        {
                FileInputStream fis=new FileInputStream(fd.getFile());
                byte[] data=new byte[fis.available()];
                fis.read(data);
                t.setText(new String(data));
        }
        catch (IOException e)
        {
                t.setText("Could not load file...");
        }
    }
}
```

```
class loadFileDiagReport extends JFrame
{
        JTextArea t=new JTextArea();
        loadFileDiagReport()
        {
                setTitle("Diagnosis Report");
                Container content=getContentPane();
                JScrollPane sPane=new JScrollPane (t);
                FileDialog fd=new FileDialog(this, "Load File", FileDialog.LOAD );
                fd.show();
                //fd.setFile      ("*.java;*.txt");
                String file=fd.getDirectory();
                if(file == null)
                return;

                content.setBackground(Color.gray);
                content.setPreferredSize(new Dimension(600,400));
                content.add(t);

                try
                {
                        FileInputStream fis=new FileInputStream(fd.getFile());
                        byte[] data=new byte[fis.available()];
                        fis.read(data);
                        t.setText(new String(data));
                }
                catch (IOException e)
                {
```

```
                    t.setText("Could not load file...");
                }
        }
}

class menu8 extends JFrame implements ActionListener,ItemListener
{
        String str=null;
        JCheckBox c1,c2,c3,c4,c5;
        JLabel l,l1,l2,sel;
        JTextArea t;
        JButton b1,b2;
        JPanel p,pp;
        int r=0,s=0,q=0,u=0,v=0;

        public menu8()
        {
                setTitle("Diagnostic Server Application");
                setSize(800,600);

                addWindowListener(new WindowAdapter()
                {
                        public void windowClosing(WindowEvent evt)
                        {
                                System.exit(0);
                        }
                });

                JMenuBar mbar=new JMenuBar();
```

```
ImageIcon iconNew1=new ImageIcon("NewD.png");
ImageIcon iconNew2=new ImageIcon("LastD.png");
ImageIcon iconNew3 = new ImageIcon("NewDRe.png");
ImageIcon iconNew4 = new ImageIcon("LastDRe.png");
ImageIcon iconNew5 = new ImageIcon("saveD.png");
ImageIcon iconNew6 = new ImageIcon("saveDRe.png");

ImageIcon iconNew7 = new ImageIcon("cut.png");
ImageIcon iconNew8 = new ImageIcon("copy.png");
ImageIcon iconNew9 = new ImageIcon("paste.png");
ImageIcon iconNew10 = new ImageIcon("selectall.png");

ImageIcon iconNew11 = new ImageIcon("viewDR.png");
ImageIcon iconNew12 = new ImageIcon("viewDP.png");
ImageIcon iconNew13 = new ImageIcon("viewLDR.png");

ImageIcon iconNew14 = new ImageIcon("contents.png");
ImageIcon iconNew15 = new ImageIcon("update.png");
ImageIcon iconNew16 = new ImageIcon("about.png");


JMenu mFile=new JMenu("File");
JMenu mEdit=new JMenu("Edit");
JMenu mView=new JMenu("View");
JMenu mHelp=new JMenu("Help");
mFile.setMnemonic('f');
mEdit.setMnemonic('e');
mView.setMnemonic('v');
```

```
mHelp.setMnemonic('h');

JMenuItem mi,mix,miy,miz;
mi=(JMenuItem)mEdit.add(new JMenuItem("Cut",iconNew7));

mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_T,ActionEvent.CTRL_M
ASK));
mi.addActionListener(this);

mix=(JMenuItem)mEdit.add(new JMenuItem("Copy",iconNew8));

mix.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Y,ActionEvent.CTRL_
MASK));
mix.addActionListener(this);

miy=(JMenuItem)mEdit.add(new JMenuItem("Paste",iconNew9));

miy.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_P,ActionEvent.CTRL_M
ASK));
miy.addActionListener(this);

miz=(JMenuItem)mEdit.add(new JMenuItem("Select All",iconNew10));


miz.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C,ActionEvent.CTRL_
MASK));
miz.addActionListener(this);

mEdit.add(new JSeparator());
```

```
JMenuItem mi1,mi1x,mi1y,mi1z,mi1r,mi1s;
mi1=(JMenuItem)mFile.add(new JMenuItem("New Diagnosis",iconNew1));

mi1.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N,ActionEvent.CTRL_
MASK));
mi1.addActionListener(this);

mi1x=(JMenuItem)mFile.add(new JMenuItem("Open
Diagnosis",iconNew2));

mi1x.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_L,ActionEvent.CTRL_
MASK));
mi1x.addActionListener(this);

mi1y=(JMenuItem)mFile.add(new JMenuItem("New Diagnosis
Profile",iconNew3));

mi1y.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_A,ActionEvent.CTRL_
MASK));
mi1y.addActionListener(this);

mi1z=(JMenuItem)mFile.add(new JMenuItem("Open Diagnosis
Profile",iconNew4));

mi1z.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_B,ActionEvent.CTRL_
MASK));
mi1z.addActionListener(this);

mi1r=(JMenuItem)mFile.add(new JMenuItem("Save Diagnosis",iconNew5));
```

```
        mi1r.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_S,ActionEvent.CTRL_
MASK));
                mi1r.addActionListener(this);

                mi1s=(JMenuItem)mFile.add(new JMenuItem("Save Diagnosis
Profile",iconNew6));

        mi1s.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_R,ActionEvent.CTRL_
MASK));
                mi1s.addActionListener(this);


                JMenuItem mi2,mi2x,mi2y;
                mi2=(JMenuItem)mView.add(new JMenuItem("Diagnosis
Report",iconNew11));

        mi2.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D,ActionEvent.CTRL_
MASK));
                mi2.addActionListener(this);

                mi2x=(JMenuItem)mView.add(new JMenuItem("Diagnosis
Profile",iconNew12));

        mi2x.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_E,ActionEvent.CTRL_
MASK));
                mi2x.addActionListener(this);

                mi2y=(JMenuItem)mView.add(new JMenuItem("Last Diagnosis
Report",iconNew13));
```

```
        mi2y.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F,ActionEvent.CTRL_
MASK));
                mi2y.addActionListener(this);


                JMenuItem mi3,mi3x,mi3y;
                mi3=(JMenuItem)mHelp.add(new JMenuItem("Contents",iconNew14));

        mi3.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_H,ActionEvent.CTRL_
MASK));
                mi3.addActionListener(this);

                mi3x=(JMenuItem)mHelp.add(new JMenuItem("Update",iconNew15));

        mi3x.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_I,ActionEvent.CTRL_
MASK));
                mi3x.addActionListener(this);

                mi3y=(JMenuItem)mHelp.add(new JMenuItem("About",iconNew16));

        mi3y.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_G,ActionEvent.CTRL_
MASK));
                mi3y.addActionListener(this);

                mbar.add(mFile);
                mbar.add(mEdit);
                mbar.add(mView);
                mbar.add(mHelp);
```

```
setJMenuBar(mbar);

l=new JLabel("DIAGNOSTIC SERVER APPLICATION");
l1=new JLabel("Please Tick the tests wherever applicable: ");
sel=new JLabel();

c1=new JCheckBox("End System Flaw Test");
c2=new JCheckBox("Path Flaw Test");
c3=new JCheckBox("Tester Flaw Test");
c4=new JCheckBox("No Path or Target Flaw Test");
c5=new JCheckBox("Insufficient Buffering Test");

c1.addItemListener(this);
c2.addItemListener(this);
c3.addItemListener(this);
c4.addItemListener(this);
c5.addItemListener(this);

b1=new JButton("Run Selected Test(s)");
b1.addActionListener(this);

b2=new JButton("Schedule Selected Test(s)");
b2.addActionListener(this);


p=new JPanel();
p.setLayout(new GridLayout(10,10,200,10));
p.setBackground(Color.gray);
```

```
l.setForeground(Color.white);
l.setFont(new Font("Arial",Font.BOLD,30));
l1.setForeground(Color.black);
l1.setFont(new Font("Times New Roman",Font.PLAIN,20));
c1.setForeground(Color.black);

c1.setFont(new Font("Georgia",Font.PLAIN,15));
c2.setForeground(Color.black);
c2.setFont(new Font("Georgia",Font.PLAIN,15));
c3.setForeground(Color.black);
c3.setFont(new Font("Georgia",Font.PLAIN,15));
c4.setForeground(Color.black);
c4.setFont(new Font("Georgia",Font.PLAIN,15));
c5.setForeground(Color.black);
c5.setFont(new Font("Georgia",Font.PLAIN,15));

b1.requestFocus();
b2.requestFocus();

b1.setBorder(new EtchedBorder());
b2.setBorder(new EtchedBorder());

p.add(l,"North");
p.add(l1,"Center");
p.add(c1,"South");
p.add(c2,"South");
p.add(c3,"South");
p.add(c4,"South");
p.add(c5,"South");
```

```
        p.add(b1,"South");
        p.add(b2,"South");
        p.add(sel,"South");

        Container cPane=getContentPane();
        cPane.add(p);
}

public void itemStateChanged(ItemEvent e)
{
        Object obj=e.getSource();
        if(obj==c1 && c1.isSelected())
        {
                r=r+1;
                sel.setText(c1.getText() + " is selected.");
        }
        else if(obj==c2 && c2.isSelected())
        {
                s=s+1;
                sel.setText(c2.getText() + " is selected.");
        }
        else if(obj==c3 && c3.isSelected())
        {
                q=q+1;
                sel.setText(c3.getText() + " is selected.");
        }
        else if(obj==c4 && c4.isSelected())
        {
                u=u+1;
```

```
                sel.setText(c4.getText() + " is selected.");
            }
            else if(obj==c5 && c5.isSelected())
            {
                v=v+1;
                sel.setText(c5.getText() + " is selected.");
            }

            System.out.println(r+" "+s+" "+q+" "+u+" "+v);
        }

        public void actionPerformed(ActionEvent ae)
        {
            int val=0;
            String strr,opt[]={"Save","Open","Exit"};
            String cmd=ae.getActionCommand();
            Object source=ae.getSource();
            if(source==b1)
            {
                val=JOptionPane.showConfirmDialog(null," Do you wish to run the
test(s)                ?                ","                Test
Run",JOptionPane.YES_NO_CANCEL_OPTION,JOptionPane.QUESTION_MESSAGE);
                if(val==0)
                {

        if(c1.isSelected()||c2.isSelected()||c3.isSelected()||c4.isSelected()||c5.isSelected())
                    {
                        JFrame ww=new prob();
                        ww.setBounds(100,100,350,160);
```

```
                    ww.setVisible(true);
                }
                else
                JOptionPane.showMessageDialog(null,"Please Select a
Test!","Test",JOptionPane.ERROR_MESSAGE);
            }
            else
            JOptionPane.showMessageDialog(null,"Test(s) Terminated!!","Job
Cancelled",JOptionPane.INFORMATION_MESSAGE);
        }
        else if(source==b2)
        {
            val=JOptionPane.showOptionDialog(null," Do you wish to save the
Schedule ? ","
",JOptionPane.OK_CANCEL_OPTION,JOptionPane.QUESTION_MESSAGE,null,opt,null)
;
            if(val==0)
            {
                strr=JOptionPane.showInputDialog(null,"Enter the Schedule
Task Name"," ",JOptionPane.PLAIN_MESSAGE);
    JOptionPane.showMessageDialog(null,"Test(s) Schedule Saved !!","Task
Schedule",JOptionPane.INFORMATION_MESSAGE);
            }
            else if(val==1)
            JOptionPane.showMessageDialog(null,"No Saved Schedule
!!","Schedule",JOptionPane.INFORMATION_MESSAGE);
            else
            JOptionPane.showMessageDialog(null,"Schedule Terminated !!","Job
Cancelled",JOptionPane.INFORMATION_MESSAGE);
        }
```

```
if(cmd.equals("Cut"))
{
        str=t.getSelectedText();
        t.replaceRange(" ",t.getSelectionStart(),t.getSelectionEnd());
}
else if(cmd.equals("Copy"))
str=t.getSelectedText();
else if(cmd.equals("Paste"))
t.insert(str,t.getSelectionStart());
else if(cmd.equals("New Diagnosis"))
{
        strr=JOptionPane.showInputDialog(null,"Enter New Diagnosis
Name",JOptionPane.PLAIN_MESSAGE);
        JFrame ff=new writefilemulnewdiag(strr);
        JOptionPane.showMessageDialog(null,"Diagnosis Created
!!","Diagnosis",JOptionPane.INFORMATION_MESSAGE);
}
else if(cmd.equals("Open Diagnosis"))
{
        JFrame nai=new loadFileDiag();
        nai.setVisible(true);
        nai.pack();
}
else if(cmd.equals("New Diagnosis Profile"))
{
        strr=JOptionPane.showInputDialog(null,"Enter New Diagnosis
Name",JOptionPane.PLAIN_MESSAGE);
        JFrame ff=new writefilemulnewdiagpro(strr);
    JOptionPane.showMessageDialog(null,"Diagnosis Created !!","Task
Schedule",JOptionPane.INFORMATION_MESSAGE);
```

```
}
else if(cmd.equals("Open Diagnosis Profile"))
{
        JFrame nai=new loadFileDiagPro();
        nai.setVisible(true);
        nai.pack();
}
else if(cmd.equals("Save Diagnosis"))
{
        val=JOptionPane.showOptionDialog(null," Do you wish to save the Diagnosis ?
","",JOptionPane.OK_CANCEL_OPTION,JOptionPane.QUESTION_MESSAGE,null,opt,null);
        if(val==0)
        {
                strr=JOptionPane.showInputDialog(null,"Enter the Diagnosis Name"," ",JOptionPane.PLAIN_MESSAGE);
                JFrame ff=new writefilemulnewdiag(strr);
    JOptionPane.showMessageDialog(null,"Diagnosis Saved !!","Diagnosis",JOptionPane.INFORMATION_MESSAGE);
        }
        else if(val==1)
        {
                JFrame nai=new loadFileDiag();
                nai.setVisible(true);
                nai.pack();
        }
        else
        JOptionPane.showMessageDialog(null,"Operation Terminated !!","Job Cancelled",JOptionPane.INFORMATION_MESSAGE);
```

```
        }
        else if(cmd.equals("Save Diagnosis Profile"))
        {
                val=JOptionPane.showOptionDialog(null," Do you wish to save the
Diagnosis                                                    Profile?
","",JOptionPane.OK_CANCEL_OPTION,JOptionPane.QUESTION_MESSAGE,null,opt,n
ull);
                if(val==0)
                {
                        strr=JOptionPane.showInputDialog(null,"Enter the Diagnosis
Profile Name"," ",JOptionPane.PLAIN_MESSAGE);
        JFrame ff=new writefilemulnewdiagpro(strr);
                        JOptionPane.showMessageDialog(null,"Diagnosis      Profile
Saved !!","Diagnosis",JOptionPane.INFORMATION_MESSAGE);
                }
                else if(val==1)
                {
                        JFrame nai=new loadFileDiagPro();
                        nai.setVisible(true);
                        nai.pack();
                }
                else
                JOptionPane.showMessageDialog(null,"Operation Terminated !!","Job
Cancelled",JOptionPane.INFORMATION_MESSAGE);
        }
        else if(cmd.equals("Diagnosis Report"))
        {
                JFrame nai=new loadFileDiagReport();
                nai.setVisible(true);
                nai.pack();
```

```
            }
            else if(cmd.equals("Diagnosis Profile"))
            {
                JFrame nai=new loadFileDiagPro();
                nai.setVisible(true);
                nai.pack();
            }
            else if(cmd.equals("Last Diagnosis Report"))
            {
                JFrame ff=new filechooser();
                ff.setBounds(100,100,400,200);
                ff.setVisible(true);
            }
            else if(cmd.equals("Contents"))
            JOptionPane.showMessageDialog(null,"Coming Soon
!!","Diagnosis",JOptionPane.INFORMATION_MESSAGE);
            else if(cmd.equals("Update"))
            JOptionPane.showMessageDialog(null,"The Application is Under
Updation","Diagnosis",JOptionPane.INFORMATION_MESSAGE);
            else if(cmd.equals("About"))
            JOptionPane.showMessageDialog(null,"Diagnostic Server Application
1.0c","Diagnosis",JOptionPane.INFORMATION_MESSAGE);
        }
        public static void main(String a[])
        {
            JFrame f=new menu8();
            f.setVisible(true);
        }
}
```

6.1.2 File Details Retrieval

```
import java.io.File;
import java.util.Date;

public class getmoreinfo
{
        public static void main(String a[])
        {
                File f=new File("C:/");
                System.out.println(f.getAbsolutePath() + (f.isDirectory() ? "is" : "is not") + "a
directory");
                System.out.println("The Parent of " + f.getName() + "is" + f.getParent());

                File[] contents=f.listFiles();
                if(contents!=null)
                {
                        System.out.println("\nThe "+ contents.length +"items in the directory "
+ f.getName() + "are:");
                        for(File file:contents)
                        {
                                System.out.println(file + "is a " + (file.isDirectory() ?
"directory" : "file") + "last modified on:\n" + new Date(file.lastModified()));
                        }
                }
                else
                System.out.println(f.getName() + "is not a directory.");
                System.exit(0);
        }
```

```
}


import java.awt.*;
import java.io.*;
import java.lang.*;
import java.applet.*;

public class GetProperties extends Applet
{
        String k1="OS.Name";
        String k2="OS.Architecture";
        String k3="OS.Version";

        public void paint(Graphics g)
        {
                int y=10;
                try
                {
                        String value=System.getPropety(k1);
                        g.drawString(k1 + ": " + value,10,y);
                }
                catch(Exception e)
                {
                        g.drawString("System.getProperty(" + k1 + ") : caught security
exception",10,y);
                }
                y=y+17;
```

```
            try
            {
                String value=System.getPropety(k2);
                g.drawString(k2 + ": " + value,10,y);
            }
            catch(Exception e)
            {
                g.drawString("System.getProperty(" + k2 + ") : caught security
exception",10,y);
            }
            y=y+17;


            try
            {
                String value=System.getPropety(k3);
                g.drawString(k3 + ": " + value,10,y);
            }
            catch(Exception e)
            {
                g.drawString("System.getProperty(" + k3 + ") : caught security
exception",10,y);
            }
            y=y+17;
        }
}
```

6.1.3 Network Client Test

```
import java.net.*;
import java.io.*;

abstract class NetworkClient
{
        private String host;
        private int port;
        public String getHost()
        {
                return(host);
        }
        public int getPort()
        {
                return(port);
        }

        public NetworkClient(String host, int port)
        {
                this.host = host;
                this.port = port;
        }
        public void connect()
        {
                try
                {
                        Socket client = new Socket(host, port);
                        handleConnection(client);
```

```
                client.close();
            }
            catch(UnknownHostException uhe)
            {
                System.out.println("Unknown host: " + host);
                uhe.printStackTrace();
            }
            catch(IOException ioe)
            {
                System.out.println("IOException: " + ioe);
                ioe.printStackTrace();
            }
        }

        protected abstract void handleConnection(Socket client) throws IOException;
}

class SocketUtil
{
    public static BufferedReader getReader(Socket s) throws IOException
    {
        return(new BufferedReader(new InputStreamReader(s.getInputStream())));
    }

    public static PrintWriter getWriter(Socket s) throws IOException
    {
        return(new PrintWriter(s.getOutputStream(), true));
    }
}
```

```
public class NetworkClientTest extends NetworkClient
{
        public NetworkClientTest(String host, int port)
        {
                super(host, port);
        }
        protected void handleConnection(Socket client) throws IOException
        {
                PrintWriter out = SocketUtil.getWriter(client);
                BufferedReader in = SocketUtil.getReader(client);
                out.println("Generic Network Client");
                System.out.printf("Generic Network Client:%n" + "Connected to '%s' and got
'%s' in response.%n",getHost(), in.readLine());
        }
        public static void main(String[] args)
        {
                String host = "localhost";
                int port = 8088;
                if (args.length > 0)
                {
                        host = args[0];
                }
                if (args.length > 1)
                {
                        port = Integer.parseInt(args[1]);
                }

                NetworkClientTest tester =new NetworkClientTest(host, port);
```

```
                tester.connect();
        }
}
```

6.1.4 Free Port Scan

```
import java.net.*;
import java.lang.*;
import java.io.*;

class portscan{

public static void main (String args[])
{
        try
        {
                if (args[0].trim().equals("-h"))
                {
                        System.out.println ("");
                        System.out.println ("This will Scan 192.168.1.100 all the way to port
110");
                        System.out.println ("Your ISP may not allow port Scanning I am not
Responsible if you get in trouble");
                        System.out.println("");
                        System.exit(0);
                }

                if (args.length < 2)
                {
```

```
                System.out.println (args[0]);
                System.out.println ("Usage: java telnet IPADDRESS ENDPORT");
                System.exit(-1);
            }
        }
        catch (ArrayIndexOutOfBoundsException ae)
        {
            System.out.println ("Usage: java telnet IPADDRESS ENDPORT");
            System.out.println ("Use java portscan -h for more info");
            System.exit(-1);
        }

        Socket client = null;
        PrintWriter out = null;
        BufferedReader in = null;
        int answer;
        answer = Integer.valueOf(args[1]).intValue();
        for (int port = 0; port < answer; port++)
        {
        try
        {
            client = new Socket(args[0], port);
            out = new PrintWriter(client.getOutputStream(), true);
                in = new BufferedReader(new
InputStreamReader(client.getInputStream()));
            System.out.println("Port Open: " + port);
        }
        catch (UnknownHostException e)
        {
```

```
                System.out.println("Can't find " + args[0]);
                System.exit(-1);
            }
            catch(IOException e)
            {
            }
            }

            System.out.println("");
            System.out.println ("Scanned ports 0 - " + answer);
            System.out.println ("All non-Displayed ports are CLOSED");
            }
}
```

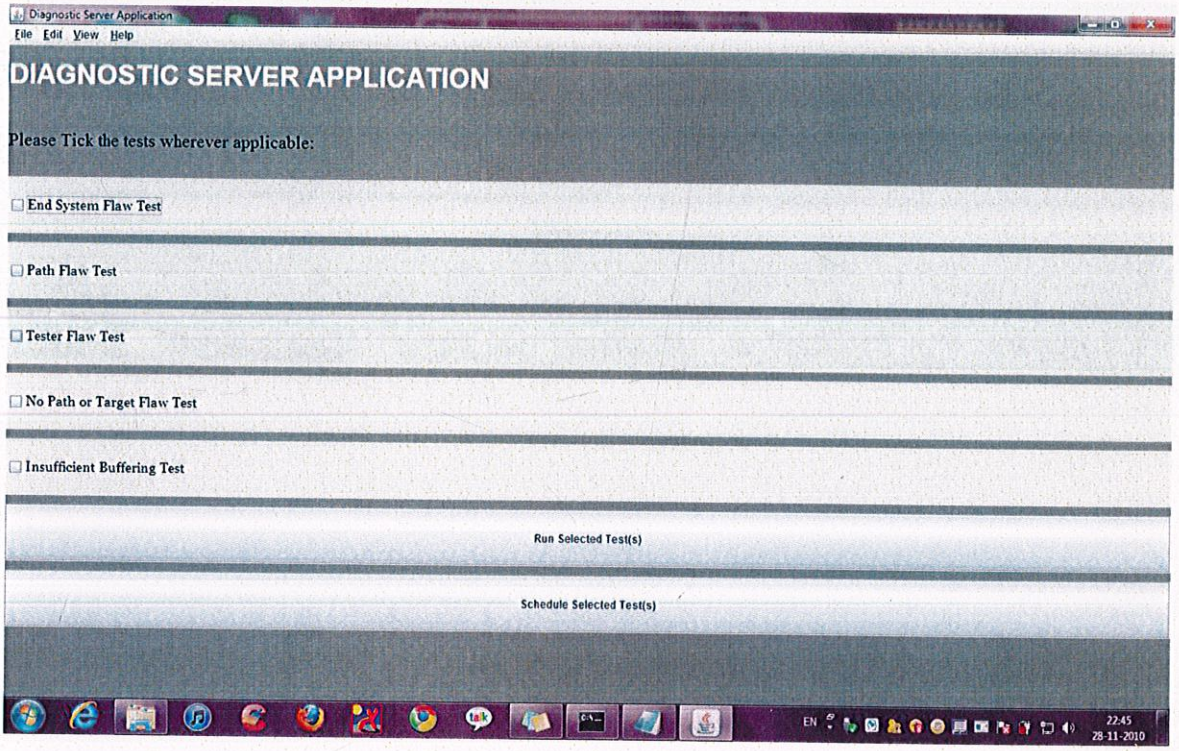
Port Scanner

```
import java.net.*;
import java.io.IOException;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

   public class PScanner {

      public static void main(String[] args) {
      InetAddress ia=null;
      String host=null;
        try {
```

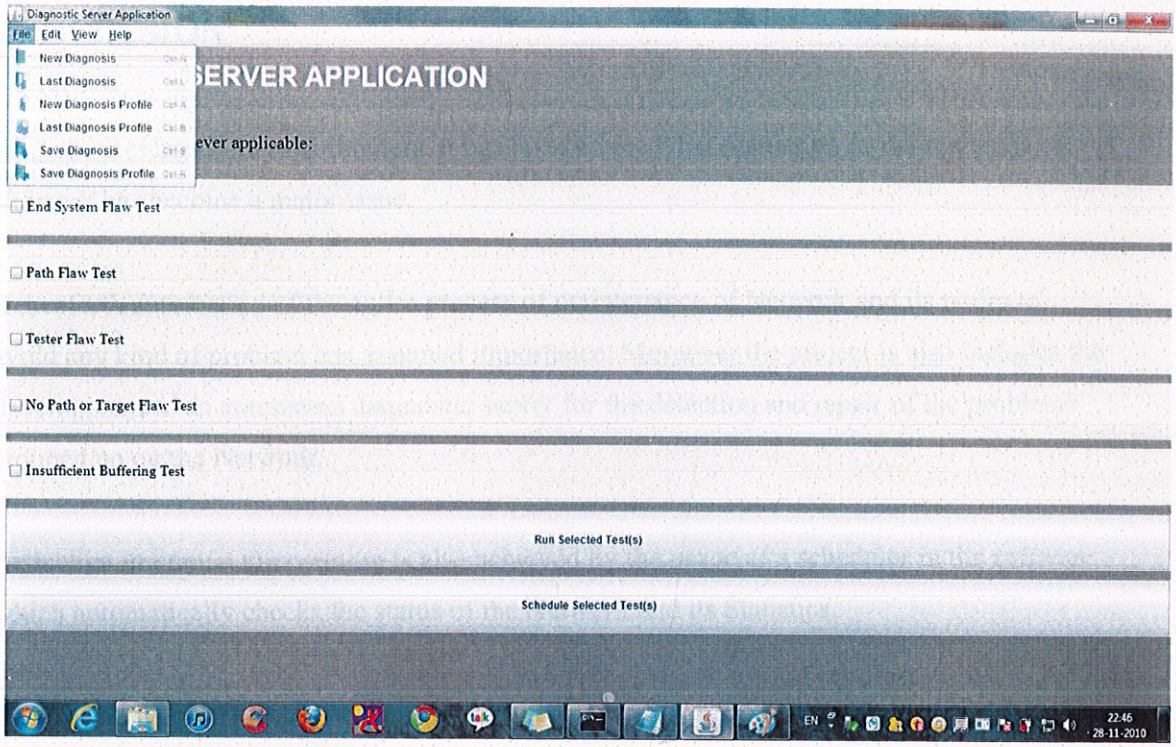
```
        host=JOptionPane.showInputDialog("Enter the Host name to scan:\n example:
xxx.com");
            if(host!=null){
            ia = InetAddress.getByName(host);
          scan(ia); }
       }
          catch (UnknownHostException e) {
          System.err.println(e );
       }
       System.out.println("Bye from NFS");
       //System.exit(0);
    }

       public static void scan(final InetAddress remote) {
       //variables for menu bar

       int port=0;
       String hostname = remote.getHostName();

          for ( port = 0; port < 65536; port++) {
             try {
             Socket s = new Socket(remote,port);
             System.out.println("Server is listening on port " + port+ " of " + hostname);
             s.close();
          }
             catch (IOException ex) {
             // The remote host is not listening on this port
             System.out.println("Server is not listening on port " + port+ " of " + hostname);
          }
```

```
        }//for ends
    }
}
```

GUI SCREENSHOT- MAIN SCREEN



GUI SCREENSHOT- SUB-MENU



CHAPTER 7

CONCLUSION

With the increased use of computers, it has been noticed that regulation or maintenance of Network has become a major issue.

Henceforth automation of the entire process of maintenance of Network and its nodes to avoid any kind of problem has assumed importance. Moreover the project is also includes the development of an automated diagnostic server for the detection and repair of the problems cropped up on the Network.

Reduction in human intervention is also achieved by the usage of a scheduler in the software which automatically checks the status of the Network and its Statistics.

The project is implemented on a small scale so as to act as a prototype.

The project is fully scalable and can be deployed on larger scale as well.

CHAPTER 8

BIBLIOGRAPHY

W. Richard Stevens, *UNIX Network Programming,* Prentice Hall, 1990.

E. R. Harold, *JAVA Network Programming,* O'Reilly.

Merlin and C. Hughes, *JAVA Network Programming,* Prentice Hall.

P. Sridharan, Advanced *JAVA Programming,* Prentice Hall.

www.cs.cf.ac.uk/Dave/HCI/

www.bytes.com/topic/java/answers/

www.java-forums.org