Internet Technology Fundamentals

by Ted Kosan

Part of The Professor And Pat series
(professorandpat.org)

Table of Contents

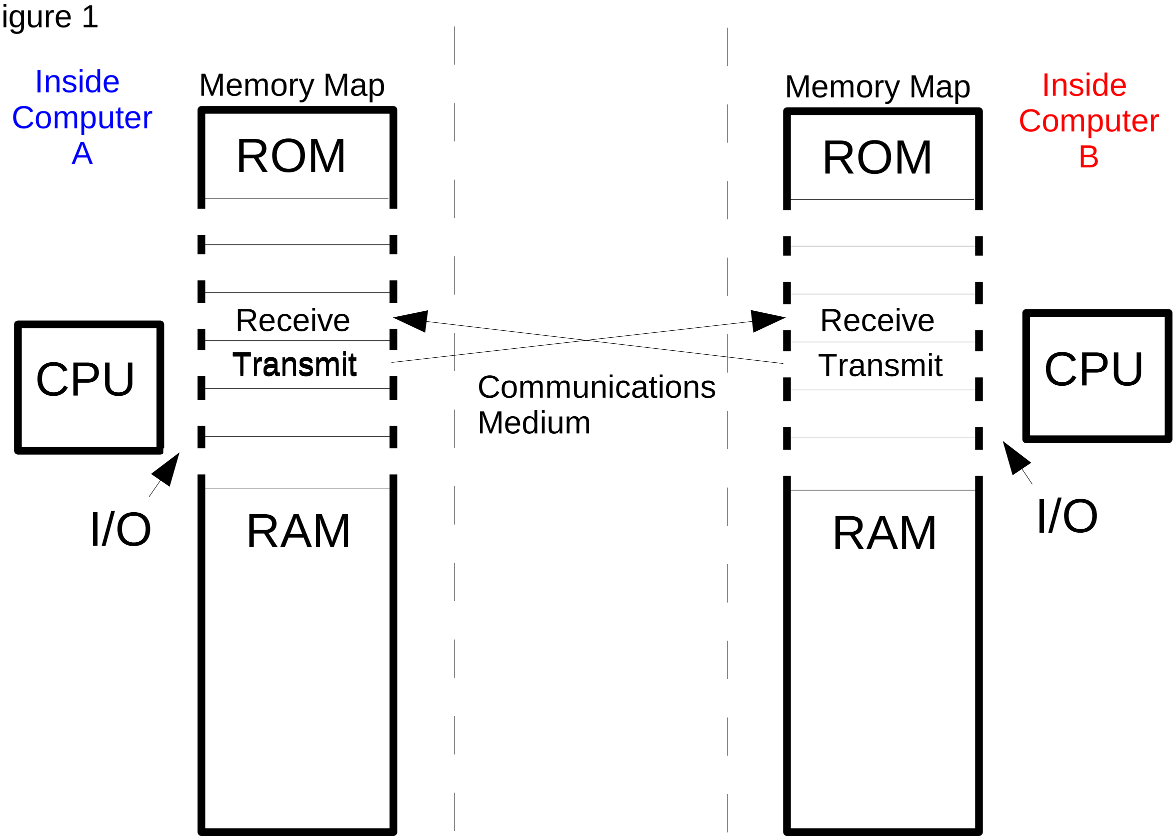
1 How does one computer communicate with another computer?

1.1 The Internet is currently one of the most important technologies of our civilization, and its importance will only increase in the future. The Internet is expanding so quickly that projections show almost all computing devices will eventually be connected to it. Therefore, it is essential for anyone who has a desire to become deeply involved with computers to understand how Internet-related technologies work.

1.2 Understanding the history of how the Internet was created is also important, but we will not be discussing this history here because it has been documented elsewhere. I highly recommend that you do an Internet search on the history of the Internet and read some of the articles you find. I assure you that it will be an excellent investment of your time.

1.3 We are going to approach the topic of how Internet-related technologies work by going back to the model of how a computer works (which was discussed in the "Computer Systems: Gateways To Cyberspace" book) and extending it to show how two computers can be connected to each other so

Figure 1



that they can communicate.

1.4 In the “Computer Systems: Gateways To Cyberspace” book, we discussed how a computer's memory map holds three kinds of memory (RAM, ROM and I/O) and that I/O memory was how a computer communicated with devices outside itself, like keyboards and printers. Since most computers are external to each other, I/O locations are also the mechanism that is used to allow one computer to communicate with another computer. Figure 1 shows two computers which are labeled "A" and "B". Both computers are using one I/O location as a **transmit location** and one I/O location as a **receive location**.

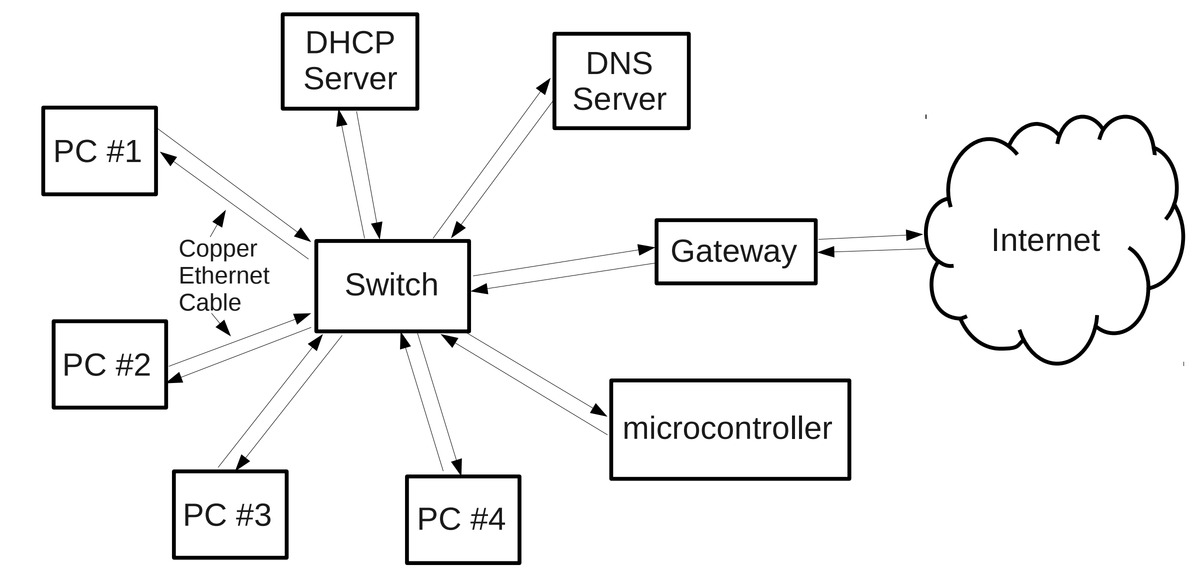
1.5 Computer "**A**'s" **transmit** location is attached to computer "**B**'s" **receive** location (and computer "**B**'s" **transmit** location is attached to computer "**A**'s" **receive** location) using a **communications medium**. A **communications medium** is something that is able to carry information from one point to another. In the case of the model in Figure 1, when computer "**A**'s" CPU places a number in computer "**A**'s" transmit location, the communications medium (represented by the right-pointing arrow) copies this number to computer "**B**'s" receive location. The communications medium represented by the left-pointing arrow will copy numbers that are placed into "**B**'s" transmit location into "**A**'s" receive location.

1.6 There are many kinds of communications mediums, including copper wires, fiber optic cables, and wireless radio signals. One of the most popular communications mediums for networking PCs is called **Ethernet** and most PCs which are sold today include an Ethernet interface. No matter which communications medium a device uses, however, they all perform the same task of copying numbers from the I/O memory locations of one computer to the I/O locations of another computer.

2 How do multiple computers communicate with each other?

2.1 When only two computers need to communicate, the situation is simple because the information that leaves one computer is sent to the other computer and vice versa. But what about the situation where multiple computers need to communicate with each other? There are a number of ways to solve this problem, and one of the more common ways is shown in Figure 2:

Figure 2 Local Area Network (LAN)



2.2 Figure 2 shows multiple computers connected to what is called a **Local Area Network** or **LAN**. A **LAN** consists of multiple computers that are physically close to each other (usually in the same room or in the same building) and attached to each other using some kind of communications medium. In Figure 2, the computers are attached to a device called a **switch** with copper Ethernet cables.

2.3 Computers on a network communicate with each other using **messages**, and sending a message is similar to sending a letter through the mail. The purpose of a **switch** is to look at each message that is sent into it, determine which computer the message is being sent to, and then sending the message to that computer.

2.4 There is a problem with the model in Figure 2, however, because the names that are associated with each computer on the network would not be suitable for uniquely identifying them if their numbers would be increased into the hundreds or thousands. Beyond this, the cloud on the right side of the figure represents the Internet, and the millions of computers (which are also called **hosts**) that are currently attached to it. Messages can also be sent to these computers and received from them, but only if each computer on the Internet is uniquely identified in some way. Beyond this, rules for how the messages are to be exchanged must also exist.

3 The TCP/IP protocol suite

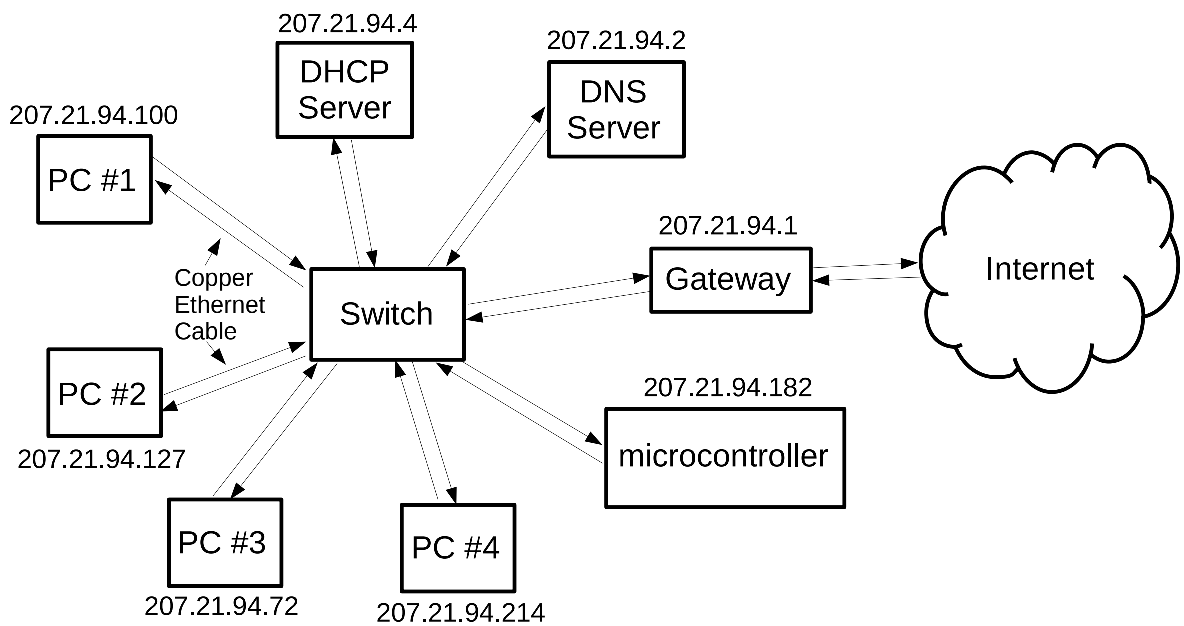
3.1 Two problems that needed to be solved before the Internet could be created were that each computer needed to be uniquely identified, and communications rules (also called **protocols**) needed to be developed which determined how the messages were to be exchanged. With respect to the Internet, a **protocol** can be defined as "**a set of rules that define an exact format for communication between systems.**" (www.unitedyellowpages.com/internet/terminology.html). When a number of protocols are used together, they are called a **protocol suite**.

3.2 The protocol suite that was developed for the Internet is called **TCP/IP**, and its name is a combination of the names of the two most heavily used protocols in the suite (**TCP** stands for **Transmission Control Protocol** and **IP** stands for **Internet Protocol**). The **Internet Protocol** defines a way to uniquely identify computers on the Internet using an addressing system. IP version 4 (**IPv4**), which is currently the most widely used version of the IP protocol, consists of **4 numbers between 0 and 255 separated from each other by a dot**. Examples of IP address include 207.21.94.50, 54.3.59.2, and 204.74.99.100. All the addresses from 0.0.0.0 to 255.255.255.255 create an **address space** that contains 4,294,967,296 addresses.

3.3 IP version 6 (**IPv6**) is the newest version of the IP protocol and it has an address space that contains 340,282,366,920,938,463,463,374,607,431,768,211,456 addresses! The transition from IPv4 to IPv6 has begun, but it is moving slowly. A large number of hosts on the Internet will continue to use the IPv4 protocol for a long time and therefore IPv4 is what we will use in this document.

3.4 Figure 3 contains the same model of a network that was shown in Figure 2 but with **IPv4** addresses assigned to each computer:

Figure 3 Local Area Network (LAN) with IPv4 addresses



3.5 If PC #3 needed to send a message to PC #4, the IP address of PC #4 (which is 207.21.94.214) would be placed into the message. The IP address of the sender (207.21.94.72) is also placed into the message in case PC #4 wanted to send a reply (this is similar to placing a return address on a letter). PC #3 will then send the message to the switch, the switch will look at the message's destination address and then pass the message to PC #4.

3.6 If one of the computers on this local network needs to send a message to a computer which is not on the LAN, then the message is sent to the **gateway** computer, and the gateway will then route the message to the Internet.

3.7 During the base Gentoo installation procedure (and also the GUI installation procedure) you have been logging directly into the superuser's account because you needed the extra privileges that this account possessed. **Normally**, however, you should log into a Linux system using your **user account** and then log into the **superuser** account (using the **su** command) only when you need the superuser account's extra privileges.

3.8 Log into the computer you installed Gentoo Linux on **using your user account** and lets determine its **IP address**. Execute an **ifconfig** command in order to determine the IP address of your machine:

```
kosan1 / # ifconfig
eth0      Link encap:Ethernet  HWaddr 00:16:D4:0B:1A:3A
          inet addr:206.21.94.132  Bcast:206.21.94.255  Mask:255.255.255.0
```

```
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:26727 errors:0 dropped:0 overruns:0 frame:0
          TX packets:22929 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:26221365 (25.0 Mb)  TX bytes:4167216 (3.9 Mb)
          Interrupt:18

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:14 errors:0 dropped:0 overruns:0 frame:0
          TX packets:14 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:756 (756.0 b)  TX bytes:756 (756.0 b)
```

3.9 The IP address of my machine is 206.21.94.132 and it was obtained from the DHCP (Dynamic Host Configuration Server) which is on the LAN that my machine is attached to.

4 Clients and servers

4.1 On LANs and on the Internet, there are a number of ways for communications between computers to be organized and these organizations are often called **architectures**. One architecture is called **Peer-to-Peer** (P2P), and it treats computers on the network as equals that exchange information with each other. An example of a P2P application is instant messaging.

4.2 Another architecture that is used with networked computers is called **Client-Server**. With a Client-Server architecture, a **server** is a computer that accepts requests from other computers on the network, performs the work that was requested, and returns the results of the work to the requester. A **client** is a computer that sends a request to a server, receives a response, and then makes use of the information that was contained in the response.

4.3 In the LAN shown in Figure 3, there are two servers (a DHCP server and a DNS server) and five clients. The servers will be discussed in the next two sections.

5 DHCP

5.1 **DHCP** stands for **Dynamic Host Configuration Protocol** and its purpose is to allow computers on a LAN to automatically be configured when they are booted up with the information they need to access the network. This information includes an **IP address**, the **address of the gateway**, and the **address of a DNS server**. We have already discussed what an IP address is and what a gateway is. DNS servers will be covered in the next section.

5.2 What you might be wondering at this point is how a computer that doesn't have an IP address yet (because it is booting up) is able to use the network to contact the DHCP server to obtain an IP address. This problem is solved by having the booting computer send a DHCP **broadcast** message to the LAN. Broadcast messages are not sent to any specific machine on a LAN. Instead, broadcast messages are sent to the LAN as a whole and all the computers that are on the LAN receive the message.

5.3 If a DHCP request message is broadcast to the LAN, the DHCP server will receive the request at the same time that the rest of the computers do. The other computers will read the contents of the message, see that it contains a DHCP request, and then they will ignore it. The DHCP server, however, will read the contents of the message, see that the message was meant for it, and send DHCP configuration information back to the sender.

5.4 Earlier we saw how the **ifconfig** command could be used to list the IP address that the DHCP server gave your machine. You can also determine what address the DHCP server gave for the **gateway** on your network by using the **netstat** command:

```
kosan1 / # netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
0.0.0.0         206.21.94.1     0.0.0.0         UG        0 0          0 enp0s3
127.0.0.0       0.0.0.0         255.0.0.0       U         0 0          0 lo
206.21.94.0     0.0.0.0         255.255.255.0   U         0 0          0 enp0s3
```

5.5 This listing shows that the address for the **gateway** on the network I am using is **206.21.94.1**.

6 DNS

6.1 Each of the millions of computers on the Internet can be accessed using their IP addresses. For example, the IP address of the server that contains the main Shawnee State University website is **146.85.50.73**. You can access this website by launching a web browser and then entering **http://146.85.50.73/** in the URL bar.

6.2 It is difficult for humans to remember numerous numbers, however, so a **system for associating names with IP address numbers** was created for the Internet. The name of the system is **DNS**, and it stands for **Domain Name System**. A name that is associated with one or more IP address is called a **domain name**, and a **domain name** that has a given machine's **hostname** at its beginning (and a period at its end) is called a **fully qualified domain name**. Examples of domain names are:

gentoo.org
yahoo.com
sourceforge.net
google.com
java.net
wikipedia.com

6.3 Examples of fully qualified domain names are:

kiwi.gentoo.org.
loon.gentoo.org.
wren.gentoo.org.

6.4 DNS is implemented as a large database that is distributed across the whole Internet. Domain names need to be registered with a **domain name registry** organization before they will be entered into the DNS system. Examples of domain name registry companies include godaddy.com, networksolutions.com, and register.com.

6.5 The DNS server on the LAN in Figure 3 has three functions. The first function is to accept messages that contain **domain names** from clients and to return the **IP address** that are associated with these names. When a user types in a domain name like **shawnee.edu** into a browser's URL bar, the browser cannot contact a the server yet because it does not know its IP address. The operating system that the browser is running on will therefore send the domain name to the DNS server (using the DNS server's IP address that it obtained through DHCP), and the DNS server will respond with one or more IP address that are associated with the **shawnee.edu** domain name. The system will then use one of these IP address to contact the Shawnee State University server.

6.6 A Gentoo Linux system holds DNS server IP addresses in the **/etc/resolv.conf** file. Take a moment to look inside this file on your system to see which DNS servers your system has been configured with.

6.7 The second function that a local DNS server has is to **define** the **domain name** to **IP address** mappings for the machines on the local network. If a remote computer on the Internet wants to know the IP address for a machine on the local network, and its DNS server does not know the mapping, the remote DNS server will contact the local **authoritative** DNS server to ask what the mapping is. The remote DNS server will then remember this mapping for a certain time in case machines on the remote network need to know the mapping in the future.

6.8 The third function that a DNS server has is to take messages that contain **IP addresses** and return the **domain names** that are associated with these addresses.

6.9 Now that you know what a DNS server does, lets **emerge** a program that will allow us to **query our local DNS servers**. The program is called **dig**, and it is contained in the **bind-tools package**. The superuser account must be used to execute the emerge command. The superuser account is entered from a user account using the **su** (superuser) command:

```
tkosan@kosan1 / $ su
Password: ***********
kosan1 / #
```

10.1 Notice that the command prompt indicates that a person is in their user account by showing the user's name and the machine's name (in green) along with a dollar '$'. After entering the superuser's account using the **su** command, only the machine's name is shown (in red) along with a number sign '**#**' instead of a dollar '$' sign.

10.2 Now that you are in the superuser's account, emerge the bind-tools program:

```
kosan1 / # emerge bind-tools
```

10.3 After the package has been emerged, lets ask **dig** what IP addresses are associated with the **shawnee.edu** domain name:

```
kosan1 / # dig shawnee.edu

; <<>> DiG 9.9.4 <<>> shawnee.edu
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22307
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;shawnee.edu.                  IN     A

;; ANSWER SECTION:
shawnee.edu.           72159 IN     A      146.85.50.73
```

```
;; Query time: 15 msec
;; SERVER: 206.21.94.6#53(206.21.94.6)
;; WHEN: Sun Apr 06 20:29:53 EST 2014
;; MSG SIZE  rcvd: 45
```

10.4 The dig program indicates that one IP address is associated the **shawnee.edu** domain name. If we want to see what fully qualified domain names are associated with this IP address, we can have **dig** find out for us by executing a **dig -x 146.85.50.73** command:

```
kosan1 / # dig -x 146.85.50.73

; <<>> DiG 9.9.4 <<>> -x 146.85.50.73
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53876
;; flags: qr rd ra; QUERY: 1, ANSWER: 3, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;73.50.85.146.in-addr.arpa.   IN    PTR

;; ANSWER SECTION:
73.50.85.146.in-addr.arpa. 86400 IN PTR   shawnee.edu.
73.50.85.146.in-addr.arpa. 86400 IN PTR   www.shawnee.edu.
73.50.85.146.in-addr.arpa. 86400 IN PTR   omniupdate.shawnee.edu.

;; Query time: 17 msec
;; SERVER: 206.21.94.6#53(206.21.94.6)
;; WHEN: Sun Apr 06 20:30:42 EST 2014
;; MSG SIZE  rcvd: 111
```

11 Processes and ports

11.1 Now that we have discussed some of the more important technologies that are related to the Internet, it is time talk about what happens when IP messages (referred to as messages from now on) arrive at a computer, and what generates messages before they are sent from a computer.

11.2 Almost all modern personal computers can have multiple programs running on them concurrently. Here is a list of programs that may be running concurrently on a typical user's computer:

- Web browser.
- Instant message client.
- Word processor.
- File download utility.
- Audio file player.
- Computer game.

11.3 In Linux, running programs are called **processes**, and a list of all the **processes** that are currently running on a Linux system can be obtained by executing a **ps -e** command:

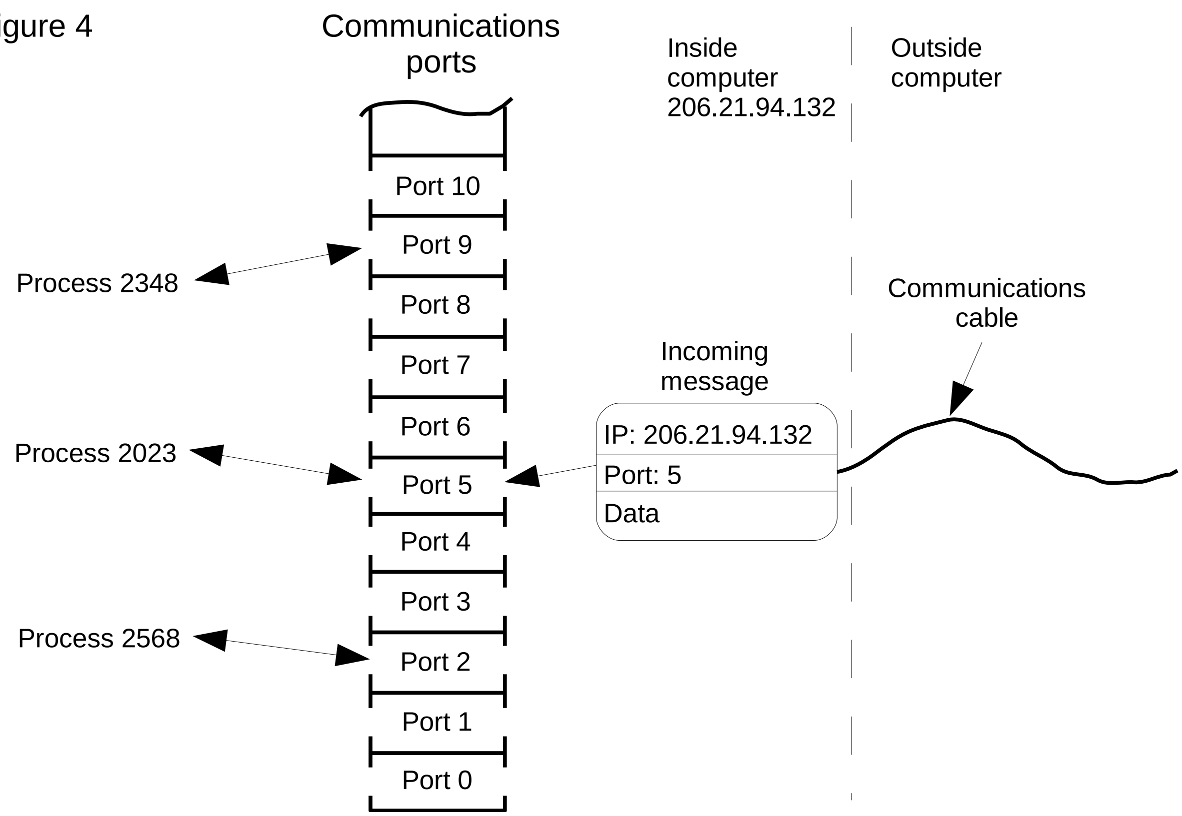
```
kosan1 / #  ps -e
  PID TTY          TIME CMD
    1 ?        00:00:00 init
    2 ?        00:00:00 ksoftirqd/0
    3 ?        00:00:00 events/0
    4 ?        00:00:00 khelper
    5 ?        00:00:00 kthread
    8 ?        00:00:00 kblockd/0
    9 ?        00:00:00 kacpid
   55 ?        00:00:00 kseriod
   58 ?        00:00:00 khubd
  145 ?        00:00:00 pdflush
  146 ?        00:00:00 pdflush
  147 ?        00:00:00 kswapd0
  148 ?        00:00:00 aio/0
  149 ?        00:00:00 cifsoplockd
  150 ?        00:00:00 cifsdnotifyd
  753 ?        00:00:00 kpsmoused
  814 ?        00:00:00 kjournald
  925 ?        00:00:00 udevd
 3532 ?        00:00:00 syslog-ng
 3947 ?        00:00:00 dhclient
 4168 ?        00:00:00 cron
 4245 tty1     00:00:00 login
 4246 tty2     00:00:00 agetty
 4247 tty3     00:00:00 agetty
 4258 tty4     00:00:00 agetty
 4259 tty5     00:00:00 agetty
 4260 tty6     00:00:00 agetty
 4285 tty1     00:00:00 bash
 4289 ?        00:00:00 sshd
 4292 pts/0    00:00:00 bash
 4296 tty1     00:00:00 startx
 4312 tty1     00:00:00 xinit
 4313 tty7     00:00:12 X
 4317 tty1     00:00:00 jwm
 4349 tty1     00:00:00 sh
 4350 tty1     00:00:00 xload
 4352 tty1     00:00:00 rxvt
 4353 pts/1    00:00:00 bash
 4356 tty1     00:00:00 mozilla-launche
 4365 tty1     00:00:12 firefox-bin
 4381 tty1     00:00:00 soffice
 4390 tty1     00:00:12 soffice.bin
 4496 pts/0    00:00:00 ps
```

11.4 If you look towards the bottom of this list you can see that my computer is currently running X Windows, the jwm window manager, the rxvt terminal emulator, a bash shell (that is attached to the terminal emulator), the firefox browser, and the open office word processor. Notice that the **ps** command included itself in the list because it was running at the moment that the list was created.

11.5 There are four columns in this listing. Each process is given a unique **Process ID** (PID) number when the process is created, and these numbers are listed in the **PID** column. The **TTY** column indicates whether or not a process is attached to a terminal, and if it is, what terminal it is attached to. The **TIME** column indicates how much CPU time the process has used so far in hours, minutes and seconds .

11.6 When a message arrives at a computer from the network, the computer must decide which process to give the message to. The way that the TCP/IP protocol solves this problem is with software-based communications **ports**.

Figure 4

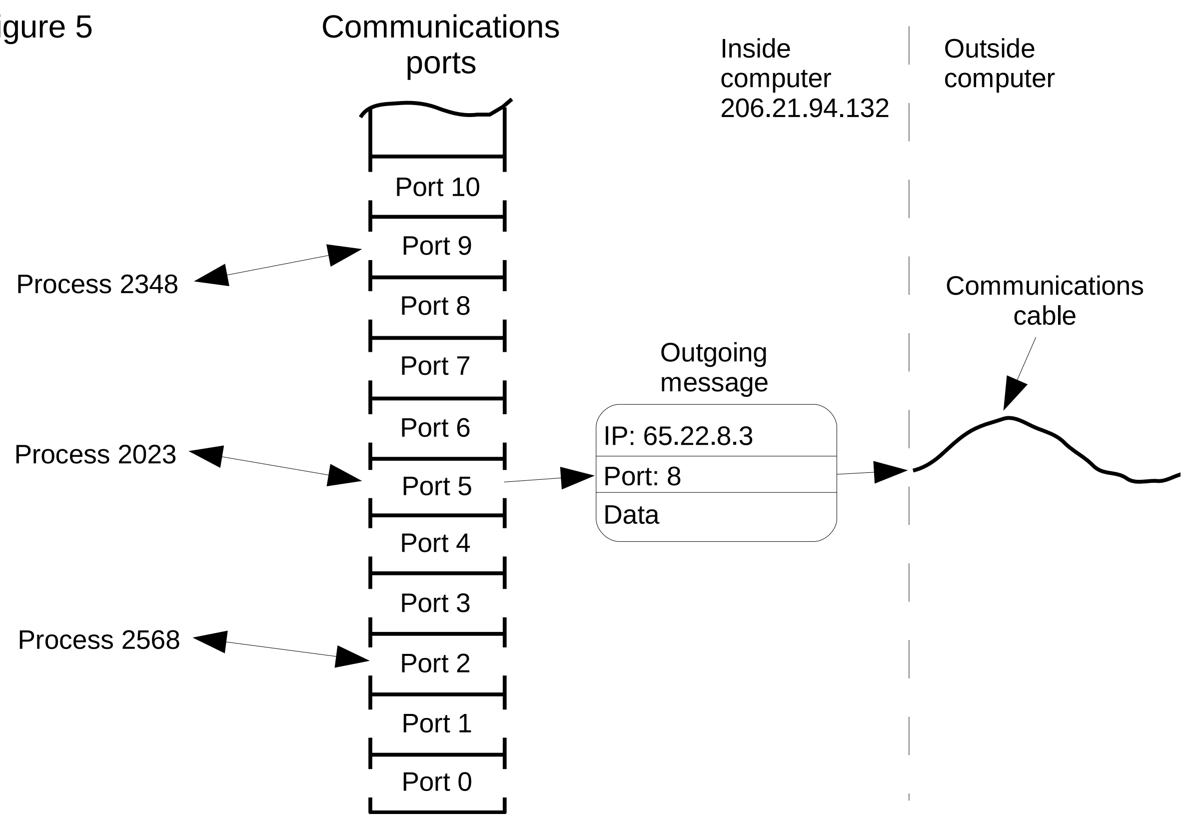


11.7 Figure 4 shows the inside and the outside of a computer that is connected to a network and which has an IP address of **206.21.94.132**. The communications ports are placed between the processes that are running on the left and the network connection on the right. Each port is given a unique number with the lowest port number being **0** and the highest port number being **65535**. Each message that arrives from the network has a port number included in it so that the system knows which port to send the message to.

11.8 In Figure 4, a message that has **port 5** as its destination port has arrived from the network and therefore the system will place this message into **port 5**. **Process 2023** has been bound to **port 5**. When the system sends the message to this port, **process 2023** will take the message and then do something with the information it contains.

11.9 Figure 5 shows a message from process **2023** being sent to another computer on the network which has an IP address of **65.22.8.3**. When this messages arrives at the destination computer, it will place the message into it's **port 8**. If there is a process on the destination computer that is bound to **port 8**, it will receive this message.

Figure 5



12 Well known ports, registered ports, and dynamic ports

12.1 Now that you know what ports are and how processes are bound to them, you may be wondering how people determine which processes should be bound to which ports. An organization called **IANA** (Internet Assigned Numbers Authority) is responsible for various number schemes associated with the Internet, and one of them is the TCP/IP port scheme. IANA has divided the **0 - 65535** port range into the following three address blocks:

0 - 1023 -> Well Known Ports.

1024 - 49151 -> Registered Ports.

49152 - 65535 -> Dynamic and or Private Ports.

12.2 Well known ports (0 - 1023)

12.2.1 A list is maintained by IANA which indicates which kinds of programs are usually bound to specific port numbers in this range. For example, **web servers** are bound to **port 80**, **SSH (secure shell) servers** are bound to **port 22**, **FTP (File Transfer Protocol) servers** are bound to **port 20**, and **DNS** servers are bound to port **53**. Here is a list of the first 25 well known ports and the full list can be obtained at http://www.iana.org/assignments/port-numbers.

```
Keyword         Decimal    Description                      References
-------         -------    -----------                      ----------
                  0/tcp    Reserved
                  0/udp    Reserved
#                          Jon Postel <postel@isi.edu>
tcpmux            1/tcp    TCP Port Service Multiplexer
tcpmux            1/udp    TCP Port Service Multiplexer
#                          Mark Lottor <MKL@nisc.sri.com>
compressnet       2/tcp    Management Utility
compressnet       2/udp    Management Utility
compressnet       3/tcp    Compression Process
compressnet       3/udp    Compression Process
#                          Bernie Volz <volz@cisco.com>
#                 4/tcp    Unassigned
#                 4/udp    Unassigned
rje               5/tcp    Remote Job Entry
rje               5/udp    Remote Job Entry
#                          Jon Postel <postel@isi.edu>
#                 6/tcp    Unassigned
#                 6/udp    Unassigned
echo              7/tcp    Echo
echo              7/udp    Echo
#                          Jon Postel <postel@isi.edu>
#                 8/tcp    Unassigned
#                 8/udp    Unassigned
discard           9/tcp    Discard
discard           9/udp    Discard
#                          Jon Postel <postel@isi.edu>
discard           9/dccp   Discard SC:DISC
```

```
#                                IETF dccp WG, Eddie Kohler <kohler@cs.ucla.edu>,
[RFC4340]
#                    10/tcp      Unassigned
#                    10/udp      Unassigned
systat               11/tcp      Active Users
systat               11/udp      Active Users
#                                Jon Postel <postel@isi.edu>
#                    12/tcp      Unassigned
#                    12/udp      Unassigned
daytime              13/tcp      Daytime (RFC 867)
daytime              13/udp      Daytime (RFC 867)
#                                Jon Postel <postel@isi.edu>
#                    14/tcp      Unassigned
#                    14/udp      Unassigned
#                    15/tcp      Unassigned [was netstat]
#                    15/udp      Unassigned
#                    16/tcp      Unassigned
#                    16/udp      Unassigned
qotd                 17/tcp      Quote of the Day
qotd                 17/udp      Quote of the Day
#                                Jon Postel <postel@isi.edu>
msp                  18/tcp      Message Send Protocol
msp                  18/udp      Message Send Protocol
#                                Rina Nethaniel <---none--->
chargen              19/tcp      Character Generator
chargen              19/udp      Character Generator
ftp-data             20/tcp      File Transfer [Default Data]
ftp-data             20/udp      File Transfer [Default Data]
ftp                  21/tcp      File Transfer [Control]
ftp                  21/udp      File Transfer [Control]
#                                Jon Postel <postel@isi.edu>
ssh                  22/tcp      SSH Remote Login Protocol
ssh                  22/udp      SSH Remote Login Protocol
#                                Tatu Ylonen <ylo@cs.hut.fi>
telnet               23/tcp      Telnet
telnet               23/udp      Telnet
#                                Jon Postel <postel@isi.edu>
                     24/tcp      any private mail system
                     24/udp      any private mail system
#                                Rick Adams <rick@UUNET.UU.NET>
smtp                 25/tcp      Simple Mail Transfer
smtp                 25/udp      Simple Mail Transfer
```

12.3 When one computer on the network wants to make use of a specific service that is running on another computer on the network, the first computer creates a message, places the port number of the desired service into the message, and then sends it to the destination computer. If a process that implements the well known service for that port is bound to the port, then this process will receive the message and perform the requested work.

12.4 The main restriction on **processes** that are bound to ports in the well known ports range is that they **must** be running with **superuser privileges**.

12.5 Registered ports (1024 - 49151)

12.5.1 **Registered ports** work similarly to **well known ports** except that the **processes** that are bound to them **do not** need to be running with **superuser privileges**. The list of **registered ports** is included in the same **IANA document** that contains the list of **well known ports**.

12.6 Dynamic/private ports (49152 - 65535)

12.6.1 These ports are used as needed, and they do not have any specific type of process associated with them. A typical use of the ports in this range is for a web browser to make an outgoing connection with a web server.

13 The SSH (Secure SHell) service

13.1 In the “Installing Gentoo Linux” book, we discussed what system services were, and then we installed a **logging** service and a **cron** service. These two services are accessed through software calls, but **some system services are bound to well known ports and they make their services available through these ports**. An example of a service that makes itself available through a well known port is the **SSH** (Secure SHell) service and it is usually bound to port **22**.

13.2 The **SSH service** allows a person to log into one computer on a network from another computer on the network. The person must know the **username** and **password** for an account on the remote machine before logging into it, and the remote machine must have a SSH service (in the form of a process) running and bound to port 22. SSH is able to provide a secure connection between the machines by encrypting the data that is passed between them.

13.3 When a system service is emerged, it usually places a small program (or script) in the **/etc/init.d** directory which will allow it to be **started**, **stopped** and **restarted**. Lets look into this directory to see what system service control scripts it contains:

```
kosan1 / # cd /etc/init.d
```

```
kosan1 init.d # pwd
/etc/init.d

kosan1 init.d # ls
bootmisc      depscan.sh    hdparm      net.enp0s3   rmnologin     sshd
checkfs       dhcpd         hostname    net.lo       rsyncd        syslog-ng
checkroot     dhcrelay      keymaps     netmount     runscript.sh  urandom
clock         functions.sh  local       nscd         shutdown.sh   vixie-cron
consolefont   gpm           localmount  numlock      slapd         xdm
crypto-loop   halt.sh       modules     reboot.sh    slurpd
```

13.4 There are a number of system service control scripts in this directory. Notice that scripts for starting, stopping and restarting the **net.np0s3** networking service, the **syslog-ng** service and the **vixie-cron** service are present here along the the script for controlling the **sshd** service. We needed to emerge the **syslog-ng** and the **vixie-cron** services ourselves but the **sshd** service is so popular that the Gentoo developers installed it by default.

13.5 If you want to see which of these services are currently running on your machine, you can execute the **rc-status** command:

```
kosan1 init.d # rc-status
Runlevel: default
 net.enp0s3                                                          [  started  ]
 syslog-ng                                                           [  started  ]
 vixie-cron                                                          [  started  ]
 netmount                                                            [  started  ]
 local                                                               [  started  ]
Dynamic Runlevel: hotplugged
Dynamic Runlevel: needed
Dynamic Runlevel: manual
 sshd                                                                [  stopped  ]
```

13.6 If you want to **start** the **sshd** service, execute the **sshd** script and pass it a **start** option. After the **sshd** service has been started, execute the **rc-status** command again to verify that it has indeed been started:

```
kosan1 init.d # /etc/init.d/sshd start
* Starting sshd ...                                                       [ ok ]

kosan1 init.d # rc-status
Runlevel: default
 net.enp0s3                                                          [  started  ]
 syslog-ng                                                           [  started  ]
 vixie-cron                                                          [  started  ]
 netmount                                                            [  started  ]
 local                                                               [  started  ]
Dynamic Runlevel: hotplugged
Dynamic Runlevel: needed
```

```
Dynamic Runlevel: manual
 sshd                                                        [  started  ]
```

13.7 If you want to **stop** the service, pass the **stop** option to it and if you want to **restart** it, use the **restart** option. Finally, if you want the **sshd service** to be automatically started when the system enters the **default runlevel** (which was discussed in the “Installing Gentoo Linux” book) use the **rc-update** command:

```
kosan1 init.d # rc-update add sshd default
 * sshd added to runlevel default
```

14 Using SSH to remotely log into a machine

14.1 Now that you know how to start and stop services, lets use the SSH service to **remotely log into a Gentoo Linux machine**. If you are working with a friend, have them create an account for you on their machine and then try to remotely log into it from your system using the **ssh client program.** Also, you can ssh from your machine's normal operating system into your VirtualBox Gentoo system. If you are running Windows, you can us the **putty.exe** program for this (**see below**). **(Note: make sure your VirtualBox network connection is set to "bridged" in the VirturalBox settings)**:

```
tkosan@kosan_laptop / $ ssh tkosan@206.21.94.136
The authenticity of host '206.21.94.136 (206.21.94.136)' can't be established.
RSA key fingerprint is f6:2b:63:33:99:6c:57:37:b4:b7:2d:ba:bc:da:be:77.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '206.21.94.136' (RSA) to the list of known hosts.
Password:
Last login: Sun Mar  4 15:38:33 2007 from 10.0.1.3

tkosan@kosan1 ~ $ pwd
/home/tkosan
```

14.2 The machine that I logged into above was at IP address **206.21.94.136** and the **username** for my account on that system is **tkosan**. The **tkosan@206.21.94.136** data that I passed to the **ssh** program told it that I wanted to have it remotely log into the **tkosan** account on the machine on the network that has IP address **206.21.94.136**.

14.3 When the **ssh** client program is asked to log into a remote machine for the first time, it tells the user that it does not currently have encryption information for this host, and it asks if it should continue. Answer by typing the word "**yes**". The program then indicates that it added information about this host to a known hosts list, and it will not ask the question again in the future.

14.4 The **known hosts** list is contained in the user's home directory in a file called **.ssh/known_hosts** (notice that the .ssh directory is a hidden directory). If you experience trouble using **ssh** to log into a computer in the future, you may need to edit the **known_hosts** file or delete it so that ssh can regenerate it from scratch.

14.5 In the above listing, the **hostname** of the machine is am logging in from is **kosan_laptop** and the **hostname** of the machine I am logging into is **kosan1**. The user account I am using on both machines is **tkosan**.

14.6 If you do not have a second Gentoo Linux machine available to experiment with, you can download a program called **putty.exe** that you can install on a Windows machine, and it will allow you to remotely log into a Gentoo machine that is running the sshd service. The **putty.exe** program can be downloaded from (http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html).

14.7 When you are done using ssh to remotely log into a machine, execute an **exit** command to close the session.

15 Using scp to copy files between machines on the network

15.1 The SSH service is not only able to allow a user to log into a remote machine, it can also be used to copy files between machines on the network. The Linux client program for copying files is called **scp** (Secure Copy) and a popular **Windows scp** client, called **pscp.exe**, can be obtained from the same url that **putty.exe** was obtained from.

15.2 We can experiment with **scp** by copying a file from the local machine to a remote machine and then copying a separate file from the remote machine to the local machine. First, change into your home directory on the **local** machine and **create a file called localfile.txt using a text editor** (just place a line or two of text in this file). Then use the **scp** command (or the **pscp** command if you are on a Windows machine) as shown below to copy the file from the **local** machine to your account on the **remote** machine:

```
tkosan@kosan1 ~ $ scp localfile.txt tkosan@206.21.94.136:
Password:
localfile.txt                                    100%   16     0.0KB/s   00:00
```

15.3 Verify that the file was copied to the remote machine by using **ssh** (or **putty**) to log into it. The **first parameter** after the **scp** command indicates where the file is being **copied from** and the **second parameter** indicates where it is being **copied to**. If an IP address is not present in either the source or destination parameter, that means that the local machine is being referenced. The colon '**:**' that is placed after the IP address indicates that the file was copied into the user's home directory, and this is where you should look for the file. You can also add a **path** after the colon if you want to copy the file to a directory other than the user's home directory.

15.4 Now, create a file on the remote machine called **remotefile.txt**, exit back to the **local** machine, and then execute the following **scp** command to copy the **remotefile.txt** file from the **remote** machine to the **local** machine:

```
tkosan@kosan1 ~ $ scp tkosan@206.21.94.136:remotefile.txt .
Password:
remotefile.txt                                    100%    6     0.0KB/s   00:00
```

15.5 In this case, the **source** is the user's home directory on the **remote** machine and the **destination** is the current directory on the **local** machine. If you recall from the “Installing Gentoo Linux” book, a period '.' indicates the current directory.