# UNIT 1 INTRODUCTION TO TCP/IP

**Structure**



## 1.0 INTRODUCTION

Transmission Control Protocol (TCP)/Internet Protocol (IP) is a set of protocols developed to allow computers of all sizes from different vendors, running different operating systems, to communicate or to share resources across a network. A packet-switching network research project was started by the USA Government in late 1960s. This in 1990s, became the most widely used form of computer networking. This project centered around ARPANET. ARPANET is the best known TCP/IP network.

TCP/IP is the principal UNIX networking protocol. This was designed to provide a reliable end-to-end byte stream over an unreliable internetwork. TCP is a connection-oriented protocol while IP is a connection less protocol.

TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual-circuit that two processes can use to communicate.

IP (Internet Protocol) provides a connection less and unreliable delivery system. It considers each datagram independently.

UDP (user datagram protocol) is a connectionless and unreliable protocol. It adds a checksum to IP for the contents of the datagram and pass members. These are used to give a client/server model.

## 1.1 OBJECTIVES

After going through this unit, you will be able to understand the following:

- Difference between various types of network protocols
- Internet addressing
- Domain Name System
- Client/Server model

## 1.2 TCP/IP LAYERING

**Transmission Control Protocol (TCP)**: Provides a reliable data stream service to network application programs. The various gateways uses TCP to communicate with mail programs on other TCP/IP nodes. Third-party applications written with the TCP/UDP programming interface can also use TCP.

**User Datagram Protocol (UDP)**: Provides an unreliable datagram service to network applications. The Third-party applications written with the TCP/UDP programming interface can also use UDP.

| | |
|---|---|
| APPLICATION | Telnet, FTP, e-mail etc. |
| TRANSPORT | TCP, UDP |
| NETWORK | IP, ICMP, IGMP |
| Physical Layer + Datalink Layer | Device Drivers and Interface Card |

**Figure 1: Four Layers of TCP/IP protocol Suite**

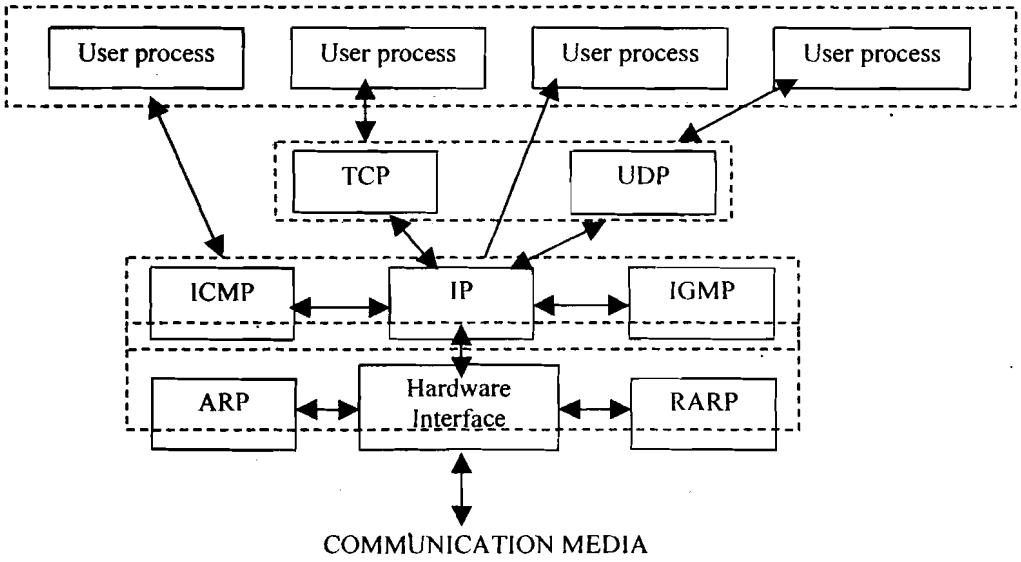


**Figure 2: Various Protocols at different Layers in TCP/IP protocol**

In the TCP/IP architecture, application protocols such as SMTP and FTP perform session and presentation layer functions. A distinct session layer or presentation layer does not exist.

The protocols in each layer provide services to the protocols in the layer above it. For example, TCP relies on IP to move TCP messages through the network.

## 1.3 THE TCP/IP STACK

In order for your computer to connect to the Internet, it must be installed with a correctly configured TCP/IP stack. A TCP/IP stack is the software that provides an implementation of the TCP/IP suite of protocols. Conceptually, this software works in

the middle, between the TCP/IP applications running on your computer and your computer's network hardware and associated drivers. Some TCP/IP applications come with their own built-in TCP stack, however, most applications expect that you will be running a stand-along stack. The later method is preferred, as you need only configure the stack once, and many applications can use it.

On Macintosh computers, the standard TCP/IP stack is called Open Transport. There are many stacks available for IBM compatible computers. Most TCP/IP applications that run under Microsoft Windows are written to be Winsock compliant. This means that they should work with any stack that provides standard Winsock compatibility. For Windows 95 or Windows NT, the Microsoft TCP stack that comes as part of the package is sufficient. Users using the older versions of Microsoft windows will need a TCP stack.

### 1.3.1 The TCP Level

Two separate protocols are involved in handling TCP/IP datagrams. TCP (the "transmission control protocol") is responsible for breaking up the message into datagram, reassembling them at the other end, resending anything that gets lost, and putting things back in the right order. IP (the "internet protocol") is responsible for routing individual datagrams. It may seem like TCP is doing all the work. And in small networks that is true. However, in the Internet, simply getting a datagram to its destination can be a complex job. A connection may require the datagram to go through several networks, a couple of Ethernets there, a series of 56K baud phone lines to another NSFnet site, and more Ethernets on another campus. Keeping track of the routes to all of the destinations and handling incompatibilities among different transport media turns out to be a complex job. Note that the interface between TCP and IP is fairly simple. TCP simply hands IP a datagram with a destination. IP doesn't know how this datagram relates to any datagram before it or after it.

To transfer any datagram TCP keeps track of multiple connections to a given system. Clearly it is not enough to get a datagram to the right destination. TCP has to know which connection this datagram is part of. This task is referred to as "demultiplexing." In fact, there are several levels of demultiplexing going on in TCP/IP. The information needed to do this demultiplexing is contained in a series of "headers". A header is simply a few extra octets tacked onto the beginning of a datagram by some protocol in order to keep track of it. It's lot like putting a letter into an envelope and putting an address on the outside of the envelope. Here is an overview of the headers that get stuck on a message that passes through a typical TCP/IP network:

We start with a single data stream say a file you are trying to send to some other computer:

.....

TCP breaks it up into manageable chunks. (In order to do this, TCP has to know how large a datagram your network can handle. Actually, the TCP's at each end say how big a datagram they can handle, and then they pick the smallest size).

.... .... .... .... .... .... .... ....

TCP puts a header at the front of each datagram. This header actually contains at least 20 octets, but the most important ones are a source and destination "port number" and a "sequence number". The port numbers are used to keep track of different conversations. Suppose 3 different people are transferring files. Your TCP might allocate port numbers 1000, 1001, and 1002 to these transfers. When you are sending a datagram, this becomes the "source" port number, since you are the source of the datagram. Of course the TCP at the other end has assigned a port number of its own for the conversation.

Your TCP has to know the port number used by the other end as well. (It finds out when the connection starts, as we will explain below). It puts this in the "destination port numbers will be reversed, since then it will be the source and you will be the destination. Each datagram has a sequence number. This is used so that the other end can make sure that it gets the datagram in the right order, and that it hasn't missed any. (See the TCP specification for details.) TCP doesn't number the datagrams, but the octets. So if there are 500 octets of data in each datagram, the first datagram might be numbered 0, the second 500, the next 1000, the next 1500, etc. Finally, We will mention the Checksum. This is a number that is computed by adding up all the octets in the datagram (more or less-see the TCP spec). The result is put in the header. TCP at the other end computes the checksum again. If they disagree, then something bad happened to the datagram in transmission, and it is thrown away. So here's what the datagram looks like now.

<table>
<tr><td colspan="8">Source Port</td><td>Destination Port</td></tr>
<tr><td colspan="9">Sequence Number</td></tr>
<tr><td colspan="9">Acknowledgement Number</td></tr>
<tr><td>Data Offset</td><td>Reserved</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Window</td></tr>
<tr><td colspan="8">Checksum</td><td>Urgent Pointer</td></tr>
<tr><td colspan="9">your data ... next 500 octets<br>.....</td></tr>
</table>

**Figure 3: Datagram**

If we abbreviate the TCP header as "T", the whole file now looks like this:

T.... T.... T.... T.... T.... T.... T....

You will note that there are items in the header that we have not described above. They are generally involved with managing the connection. In order to make sure the datagram has arrived at its destination, the recipient has to send back an "acknowledgement". This is a datagram whose "Acknowledgement number" field is filled in. For example, sending a pack with an acknowledgement of 1500 indicates that you have received all the data up to octet number 1500. If the sender doesn't get an acknowledgement within a reasonable amount of time, it sends the data again. The window is used to control how much data can be in transit at any one time. It is not practical to wait for each datagram to be acknowledged before sending the next one. That would slow things down too much. On the other hand, you can't just keep sending, or a fact computer might overrun the capacity of a slow one to absorb data. Thus each end indicates how much new data it is currently prepared to absorb by putting the number of octets in its "Window" field. As the computer receives data, the amount of space left in its window decreases. When it goes to zero, the sender has to stop. As the receiver processes the data, it increases its window, indicating that it is ready to accept more data. Often the same datagram can be used to acknowledge receipt of a set of data and to give permission for additional new data (by an updated window). The "Urgent" field allows one end to tell the other to skip ahead in its processing to a particular octet. This is often useful for handling asynchronous events, for example when you type a control character or other command that interrupts output.

### 1.3.2 The IP Level

TCP sends each of these datagram to IP. Of course it has to tell IP the Internet address of the computer at the other end. Note that this is all IP is concerned about. It doesn't care about what is in the datagram, or even in the TCP header. IP's job is simply to find

a route for the datagram and get it to the other end. In order to allow gateways or other intermediate systems to forward the datagram, it adds its own header. The main things in this header are the source and destination Internet address (32-bit addresses, like 128.6.4.194), the protocol number, and another checksum. The source Internet address is simply the address of your machine. (This is necessary so the other end knows where the datagram came from). The destination Internet address is the address of the other machine. (This is necessary so any gateways in the middle know where you want the datagram to go). The protocol number tells IP at the other end to send the datagram to TCP. Although most IP traffic uses TCP, there are other protocols that can use IP, so you have to tell IP which protocol to send the datagram to. Finally, the checksum allows IP at the other end to verify that the header wasn't damaged in transit. Note that TCP and IP have separate checksums. IP needs to be able to verify that the header didn't get damaged in transit, or it could send a message to the wrong place. For reasons not worth discussing here, it is both more efficient and safer to have TCP compute a separate checksum for the TCP neader and data. Once IP has tacked on its header, here's what the message looks like:

<table>
<tr><td>Version</td><td>IHL</td><td>Type of Service</td><td colspan="2">Total Length</td></tr>
<tr><td colspan="3">Identification</td><td>Flags</td><td>Fragment Offset</td></tr>
<tr><td colspan="2">Time to Live</td><td>Protocol</td><td colspan="2">Header Checksum</td></tr>
<tr><td colspan="5">Source Address</td></tr>
<tr><td colspan="5">Destination Address</td></tr>
<tr><td colspan="5">TCP header, then your data .....</td></tr>
</table>

**Figure 4: Message Format**

If we represent the IP header by an "I", your file now looks like this:

IT.... IT.... IT.... IT.... IT.... IT.... IT....

Again, the header contains some additional fields that have not been discussed. Most of them are beyond the scope of this unit. The flags and fragment offset are used to keep track of the pieces when a datagram has to be split up. This can happen when datagram are forwarded through a network for which they are too big. (This will be discussed a bit more below.) The *time to live* is a number that is decremented whenever the datagram passes through a system. When it goes to zero, the datagram is discarded. This is done in case a loop develops in the system somehow. Of course this should be impossible, but well-designed networks are built to cope with "impossible" conditions.

At this point, it's possible that no more headers are needed. If your computer happens to have a direct phone line connecting it to the destination computer, or to a gateway, it may simply send the datagram out on the line (though likely a synchronous protocol such as HDLC would be used, and it would add at least a few octets at the beginning and end).

### 1.3.3 The Ethernet Level

However most of our networks these days use Ethernet. So now we have to describe Ethernet's headers. Unfortunately, Ethernet has its own addresses. The people who designed Ethernet wanted to make sure that no two machines would end up with the same Ethernet address. Furthermore, they didn't want the user to have to worry about

assigning addresses. So each Ethernet controller comes with an address builtin from the factory. In order to make sure that they would never have to reuse addresses, the Ethernet designers allocated 48 bits for the Ethernet address. People who make Ethernet overlap any other manufacturer. Ethernet is a "broadcast medium". That is, it is in effect like an old party line telephone. When you send a packet out on the Ethernet, every machine on the network sees the packet. So something is needed to make sure that the right machine gets it. As you might guess, this involves the Ethernet header. Every Ethernet packet has a 14-octet header that includes the source and destination Ethernet address, and a type code. Each machine is supposed to pay attention only to packets with its own Ethernet address in the destination field. (It's perfectly possible to cheat, which is one reason that Ethernet communications are not terribly secure.) Note that there is no connection between the Ethernet address and the Internet address. Each machine has to have a table of what a Ethernet address corresponds to what Internet address. In addition to the addresses, the header contains a *type code*. The *type code* is to allow for several different protocol families to be used on the same network. So you can use TCP/IP, DECnet, Xerox NS, etc. at the same time. Each of them will put a different value in the type field. Finally, there is a checksum. The Ethernet controller computes a checksum of the entire packet. When the other end receives the packet, it recomputes the checksum, and throws the packet away if the answer disagrees with the original. The checksum is put on the end of the packet, not in the header. The final result is that your message looks like this:

| | |
|---|---|
| Ethernet destination address (first 32 bits) | |
| Ethernet dest (last 16 bits) | Ethernet source (first 16 bits) |
| Ethernet source address (last 32 bits) | |
| Type code | |
| IP header, then TCP header, then your data<br>...<br>end of your data | |
| Ethernet Checksum | |

**Figure 5: Final Message**

If we represent the Ethernet header with "E", and the Ethernet checksum with "C", your file now looks like this:

EIT....CEIT....CEIT....CEIT....CEIT....C

When these packets are received by the other end, of course all the headers are removed. The Ethernet interface removes the Ethernet header and the checksum. It looks at the type code. Since the type code is the one assigned to IP, the Ethernet device driver passes the datagram up to IP. IP removes the IP header. It looks at the IP protocol field. Since the protocol type is TCP, it passes the datagram up to TCP. TCP now looks at the sequence number. It uses the sequence numbers and other information to combine all the datagram into the original file.

This ends our initial summary of TCP/IP. Now let us have an overview of Internet addressing. The more detailed discussion will be in Unit 2.

# 1.4 INTERNET ADDRESSING

## IP Address Component

Like other network layer protocols, the IP addressing scheme is integral to the process of routing IP data through an internetwork.
Each host on a TCP/IP network is assigned a unique 32-bit logical address. The IP address is divided into two main parts; the Network Number and the Host Number.

The network number identifies the network and must be assigned by the Internet Network Information Center (InterNIC) if the network is to be part of the Internet.

The host number identifies a host in the network and is assigned by the local network administrator.

### 1.4.1 IP Address Format

The 32-bit IP address is grouped 8 bits at a time, each group of 8 bits is an octet. Each of the four octets are separated by a dot, and represented in decimal format, this is known as dotted decimal notation. Each bit in an octet has a binary weight (128, 64, 32, 16, 8, 4, 2, 1). The minimum value for an octet is 0 (all bits set to 0), and the maximum value for an octet is 255 (all bits set to 1).

The following figure shows the basic format of a typical IP address:

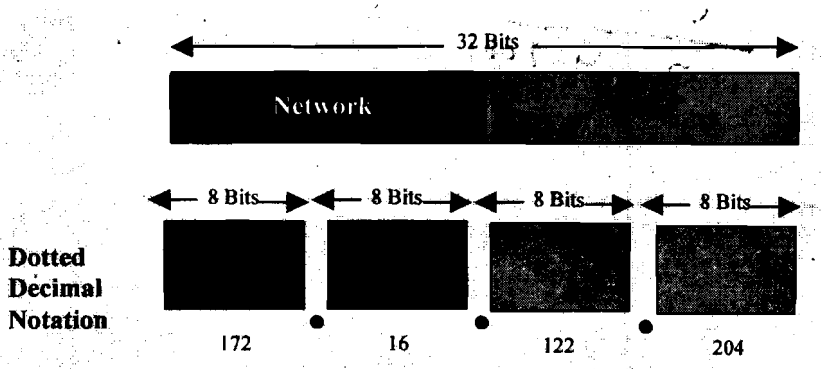


**Figure 6: Format of a Typical IP address**

### 1.4.2 IP Address Classes

'P addressing supports three different commercial address classes; Class A, Class B, and Class C.

In a class A address, the first octet is the network portion, so the class A address of, 10.1.25.1, has a major network address of 10. Octets 2, 3, and 4 (the next 24 bits) are for the hosts. Class A addresses are used for networks that have more than 65,536 hosts (actually, up to 16,581,375 hosts!).

In a class B address, the first two octets are the network portion, so the class B address of, 172.16.122.204, has a major network address of 172,16. Octets 3 and 4 (the next 16

bits) are for the hosts, Class B addresses are used for networks that have between 256 and 65,536 hosts.

In a class C address, the first three octets are the network portion. The class C address of, 193.18.9.45, has a major network address of 193.18.9. Octet 4 (the last 8 bits) is for hosts. Class C addresses are used for networks with less than 254 hosts.

### 1.4.3 First-Octet Rule

The class of address can be easily determined by examining the first octet of the address, and mapping that value to a class range in the table below:

The left-most (high-order) bits in the first octet indicate the network

| Address Class | First Octet in Decimal | High-Order Bits |
|---|---|---|
| Class A | 1-126 | 0 |
| Class B | 128-191 | 10 |
| Class C | 192-223 | 110 |

**Figure 7: Address Class**

For example, given an IP address of 172.31.1.2, the first octet is 172. 172 falls between 128 and 191, so 172.31.1.2 is a Class B address.

## 1.5 DOMAIN NAME SYSTEM (DNS)

- For human beings wanting to access Internet resources, names are much easier to remember than IP addresses;
- The Domain Name System (DNS) was created to provide a mapping between names for Internet resources and their associated IP addresses;
- **Characteristics of DNS:** The following are the characteristics of DNS
  - ✓ Hierarchical naming scheme

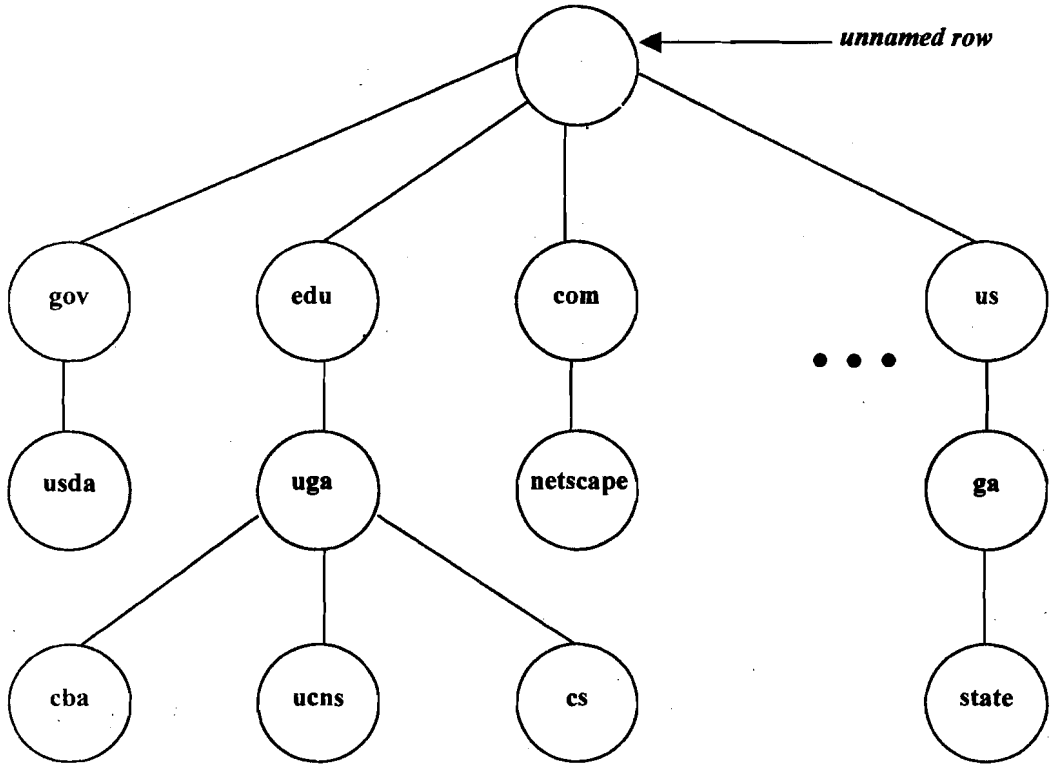


**Figure 8 : Hierarchical naming scheme**

- ✓ Delegation of authority for names
- ✓ Distributed databases of name to IP address (and IP to name) mappings

- Each name authority must operate at least two DNS database servers (name servers) for their authorized domain
- Every TCP/IP implementation has a software routine called the ***name***

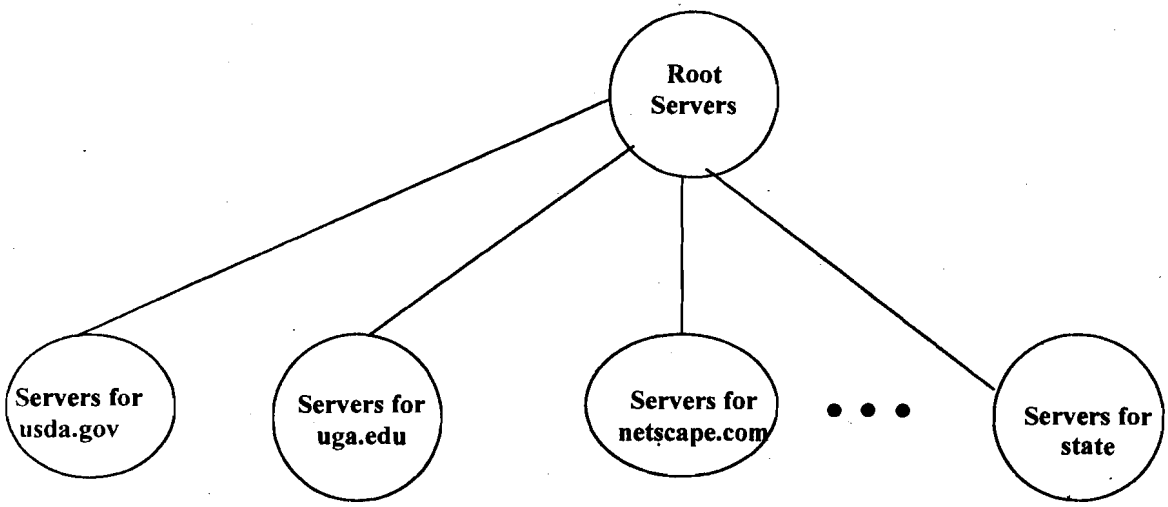


**Figure 9: Example of DNS database servers**

*resolver (NR)* to request a DNS lookup from a name server (NS)

- Two types of name resolution:

- ✓ **Recursive resolution:** NR asks NS to resolve names for which it does not have an authoritative answer by querying other name servers (**predominant method**)
- ✓ **Iterative resolution:** NR asks NS to provide the IP address of a NS that can provide an authoritative answer

- Typical name resolution process:
  1. NR receives a domain name for client TCP/IP application, formulates a DNS query and sends it to the first NS in its list;
  2. NS determines if it is the authority for the domain name;
  3. If so, it looks up the answer and sends an authoritative response back to client's NR;
  4. If not, it (typically) queries other name servers for an authoritative response, sends a non-authoritative response to the client's NR and caches the response in case it receives a NR request for the same name;
  5. The NR passes the response back to the application program and caches it for a period of time; and
  6. If the NR does not receive a response from the NS in a specified period of time, it sends the query to the next NS in its list

| IDENTIFICATION | PARAMETER* |
|---|---|
| NUMBER OF QUESTIONS | NUMBER OF ANSWERS |
| NUMBER OF AUTHORITY | NUMBER OF ADDITIONAL |
| QUESTION SECTION* | |
| ANSWER SECTION* | |
| AUTHORITY SECTION* | |
| ADDITIONAL INFORMATION SECTION* | |

**Figure 10: DNS message format**

- Pointer queries
  - √ Some TCP/IP server programs are configured to verify that an IP address has a corresponding domain name
  - √ IP address to domain name mapping is performed though a pointer query
  - √ For a given IP address of the form www.xxx.yyy.zzz, the format for a pointer query is:

**zzz.yyy.xxx.www.in-addr.arpa**

  - √ NR sends the query (with a type of PTR) to name server
  - √ If it the authoritative NS for that IP address, it returns the corresponding domain name
  - √ If not, it queries a root NS for the authoritative NS, queries that NS, and returns the response to the NR.

- Domain suffix lists
  - √ Some name resolvers append a suffix name to the domain name from *a domain suffix list* before formulating a DNS query
  - √ An example of a domain suffix list is:

**.ucns.uga.edu**
**dev.uga.edu**
**.uga.edu**
**null**

√ Some NRs automatically add domain suffixes one at a time to *all* domain name
√ Specifying a period (.) at the end of a domain name usually stops the NR from adding suffixes.

## 1.6 CLIENT/SERVER MODEL

Most of the TCP/IP applications that you will use across the Internet operate on a *client/server* model. In this model, the actual machines and applications that you use to get information (World-Wide-Web browsers, Gopher browsers, electronic mail programs, News reading programs, IRC chat programs, etc.) are the *clients*. The machines and programs that provide the information are the *servers*. It is important to understand that when using the client/server model, a specific machine can be both a client and a server. For example, if you are using your PC to browse the World Wide Web, your PC is a client. When a colleague of yours connects to that same machine to copy a file to his machine, your machine is a server.

### Clients and Servers

An application that initiates peer-to-peer communication is called a client. Most client Software consists of conventional application programs. Each time a client application executes, it contacts a server, sends a request and awaits a response. When the response arrives client continues processing. Clients are often easier to build them server and usually require no system privileges. By comparison a server is any program that waits for incoming communication requests from a client. The server receives a clients requests, performs the necessary computation, and returns the result to the client.

Because servers often need to access data computation and protocol parts that the operating system protects. Server software requires special system privileges.

Server must contain code that handles issue of:

- Authentication : verifying the identity of the client;
- Authorization : determining whether a given client is permitted to access the server supplies;
- Data Security : guaranteeing that data is not unintentionally revealed or compromised;
- Protection : guarantees the network application cannot abuse technically. A Server is a program and not a piece of hardware. For example "the computer is our file server" that means "the computer runs our file server program".

Although TCP/IP defines many standard application protocols, most commercial computer vendors supply only a handful of standard application client programs with their TCP/IP software. For example, TCP/IP software usually includes a remote terminal client that uses the standard TELNET protocol for remote log in.

An E-mail client that uses the standard SMTP protocol to transfer E-mail to a remote system.

A file transfer client uses the standard FTP protocol to transfer files between two machines.

## Check Your Progress

1) What is TCP/IP stack? Explain.

2) What is difference between TCP and UDP?

3) Describe the Ethernet frame format.

4) What is IP address format? Elaborate IP subnet addressing.

5) What are different type of IP address components? Define its classes also.

6) Describe recursion resolution in terms of DNS

Describe the DNS message format.

## 1.7 SUMMARY

This unit summarizes our discussion on TCP/IP protocol. In nutshell we can conclude that TCP/IP is the nerve center which allow different types of networks and computers to be interconnected, though the TCP works at transport level and IP works at network level. Unlike UOP, TCP/IP is a reliable connection oriented protocol. IP address (IPV4) is of 32 bit, though the latest version is of 128 bits addressing format which is IPV6 about which we study later.

Each machine is recognized by its unique outlet no or so called IP address depending on upon their class when the network designated too small for an organization it can be expanded with a technique called subnet masking or subnet addressing. This is done by borrowing bits from the host field.

DWS is a mechanism or technique which provides a mapping formats between internet resources and internet addresses.

## 1.8 MODEL ANSWERS

| | | | |
|---|---|---|---|
| Question | 1, 2, 3 | - | Refer to 1.2 – 1.3 |
| Question | 4 – 5 | - | Refer to 1.4 |
| Question | 6 – 7 | - | Refer to 1.6 |