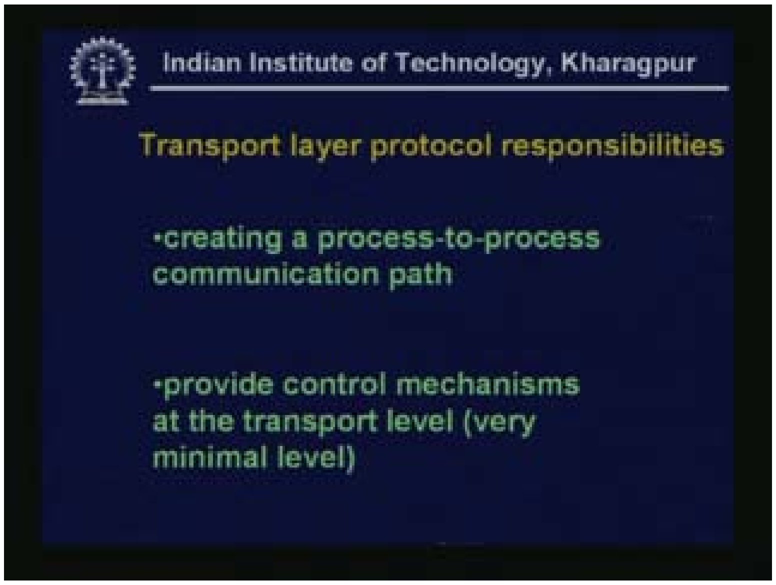
Computer Networks

Prof. S. Ghosh
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecturer # 30
UDP and Client Server

Good day, today we will start our discussion about transport layer protocols. There are two dominant protocols the UDP and TCP. We will take them up one by one. Let us look at UDP in this lecture and TCP in the next one. UDP stands for User Datagram Protocol.

(Refer Slide Time: 1:09)



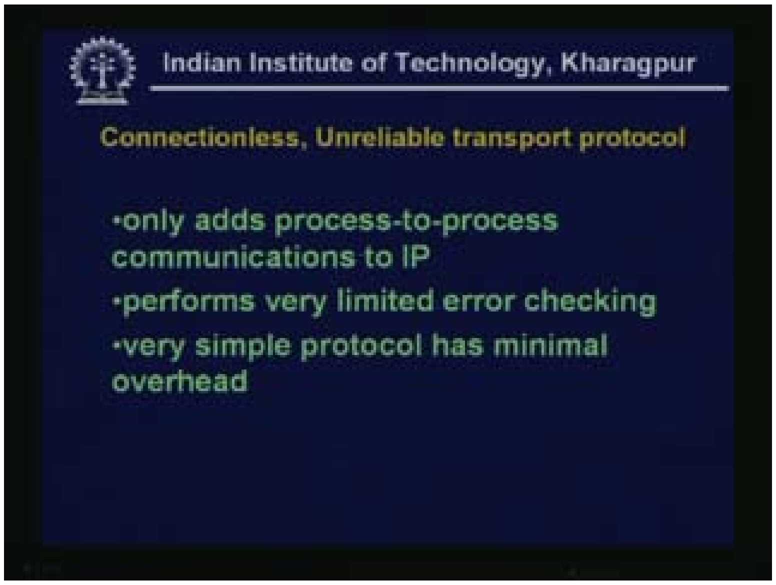
Indian Institute of Technology, Kharagpur

Transport layer protocol responsibilities

- creating a process-to-process communication path
- provide control mechanisms at the transport level (very minimal level)

This is a transport layer protocol and has got the following responsibilities:
First of all, it creates a process to process communication path. Till now we have talked about the network layer and the job of the network layer is to connect a distant machine to another distant machine. So it is a machine to machine communication whereas now we are talking about process to process communication. So in this particular source machine may be some application process is running which is trying to connect to the other distant machine for some job. So this process has to connect to a corresponding process there which may be a particular application server on one side and application client on the other side whatever the applications may be. This is a process to process communication path. This also provides control mechanisms at the transport level. The control mechanism in the case of UDP is very minimal.

(Refer Slide Time: 2:58)


Indian Institute of Technology, Kharagpur

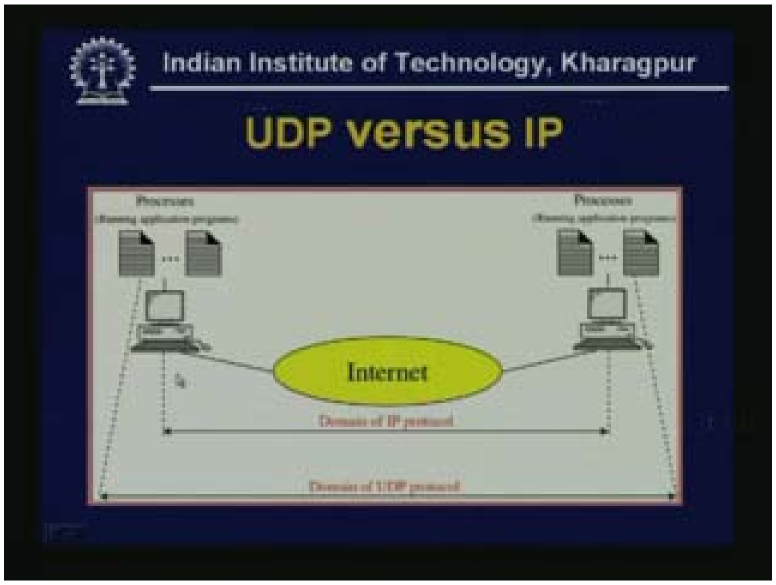
Connectionless, Unreliable transport protocol

- only adds process-to-process communications to IP
- performs very limited error checking
- very simple protocol has minimal overhead

UDP is a connectionless unreliable transport protocol. But why would we try to have an unreliable protocol? This protocol is not exactly unreliable but it does not do anything extra for reliability making it a very light weight protocol. So its overhead cost is very low. In many cases it may be a very reasonable thing to have where you do not expect lot of errors or you do not really care if some error occurs from time to time. In such cases you may use UDP. Functions of UDP:

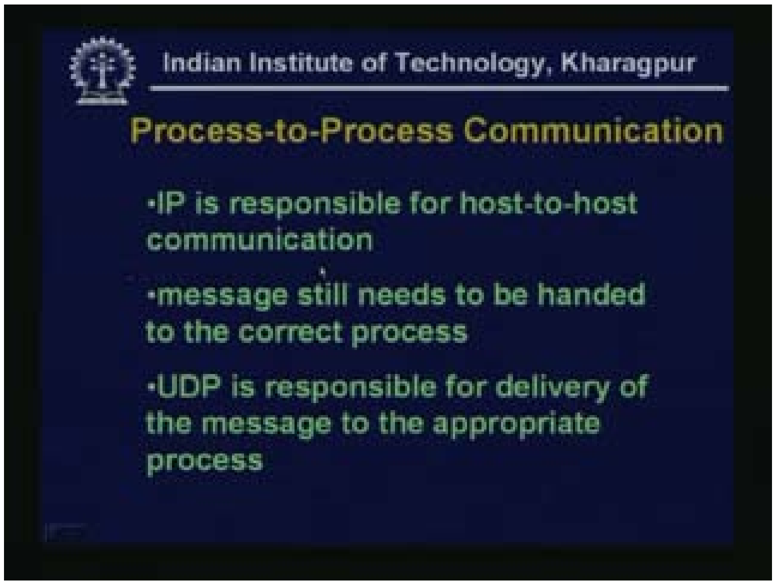
- It only adds process to process communications to IP.
- Performs very limited error checking.
- Very simple protocol and has minimal overhead. This is the main advantage. It has very minimal overhead.

(Refer Slide Time: 3:20)


Indian Institute of Technology, Kharagpur
UDP versus IP
Processes
(Running application programs)
Processes
(Running application programs)
Internet
Domain of IP protocol
Domain of UDP protocol

When you are talking about the IP protocol you are talking about the connection from one machine to another. But in this particular machine a number of application programs or processes may be running. This UDP protocol connects one process to another process, so does TCP and that is the job of the transport layer. Although UDP is a connectionless protocol the job of the transport layer is to make some virtual connection or some virtual communication channel available to the corresponding processes.

(Refer Slide Time: 4:05)



Indian Institute of Technology, Kharagpur

Process-to-Process Communication

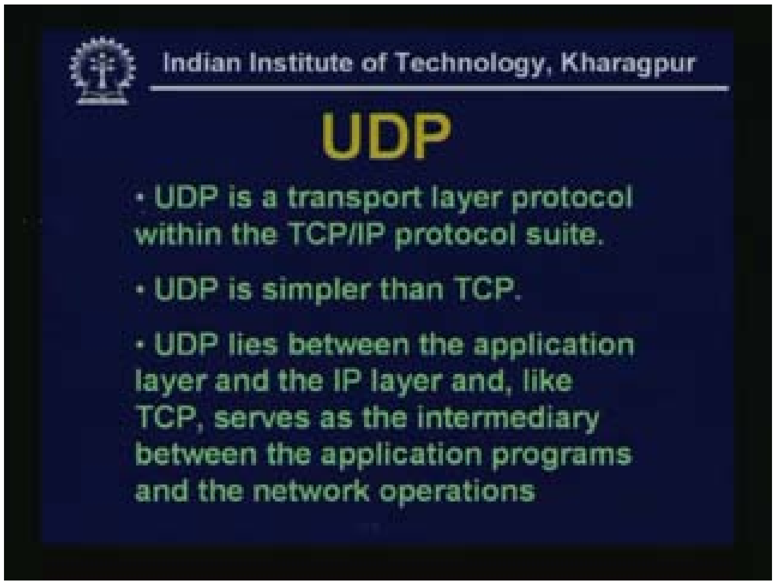
- IP is responsible for host-to-host communication
- message still needs to be handed to the correct process
- UDP is responsible for delivery of the message to the appropriate process

So to summarize:
- IP is responsible for host to host communication.

- Message still needs to be handed to the correct process.
- UDP is responsible for delivery of the message to the appropriate process. Actually, not only you need to understand that there is some kind of multiplexing and de-multiplexing going on. In the same server a number of processes may be running and out of all those processes a good number of them may be using the same UDP protocol. So, when sending out a packet it is alright but when receiving a packet the UDP protocol has to determine as to which process it will go to and that is one of its jobs and the other job is to make connections.
-

There is a point over here and this is a common task between TCP and the UDP. Our computer network is mostly a packet switched network whereas as far as applications are concerned mostly they do not really bother about packets. They may produce a chunk of data to be communicated to the other side. They may even produce a stream of data to be communicated to the other side so it does not work with packets. So somebody has to take this stream of data from the application layer and chop them into small packets and that is a job of the transport layer. So, both the TCP and UDP do that.

(Refer Slide Time: 6:05)



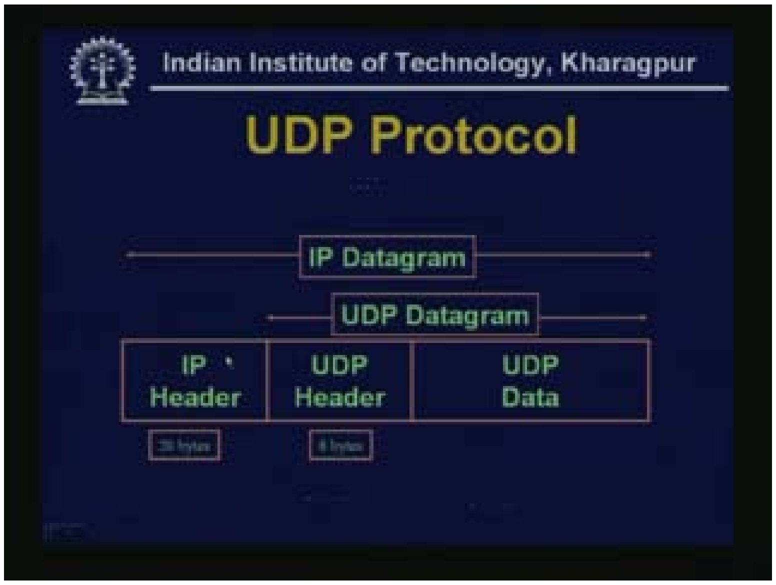
Indian Institute of Technology, Kharagpur

UDP

- UDP is a transport layer protocol within the TCP/IP protocol suite.
- UDP is simpler than TCP.
- UDP lies between the application layer and the IP layer and, like TCP, serves as the intermediary between the application programs and the network operations

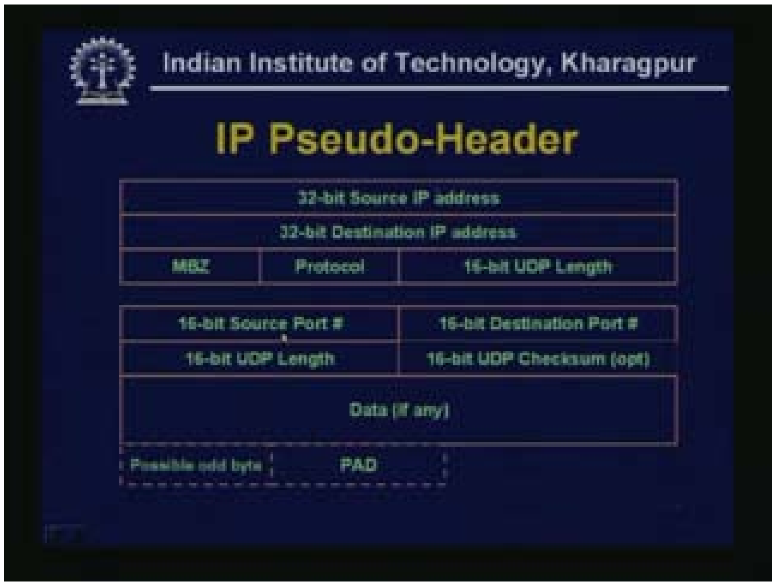
- UDP is a transport layer protocol within the TCP/IP protocol suite.
- It is simpler than TCP. TCP has higher overhead, it is more reliable than UDP. If you want to have reliability and still want to use UDP then you have to take care of reliability in some other layer may be application layer or something.
- UDP lies between the application layer and the IP layer and like TCP serves as the intermediary between the application programs and the network operations.

(Refer Slide Time: 6:40)



If this is your IP datagram, the IP datagram will have an IP header and the payload for the IP would be the entire UDP datagram. The UDP datagram would have the UDP data which it derives from the application and the UDP header.

(Refer Slide Time: 9:10)



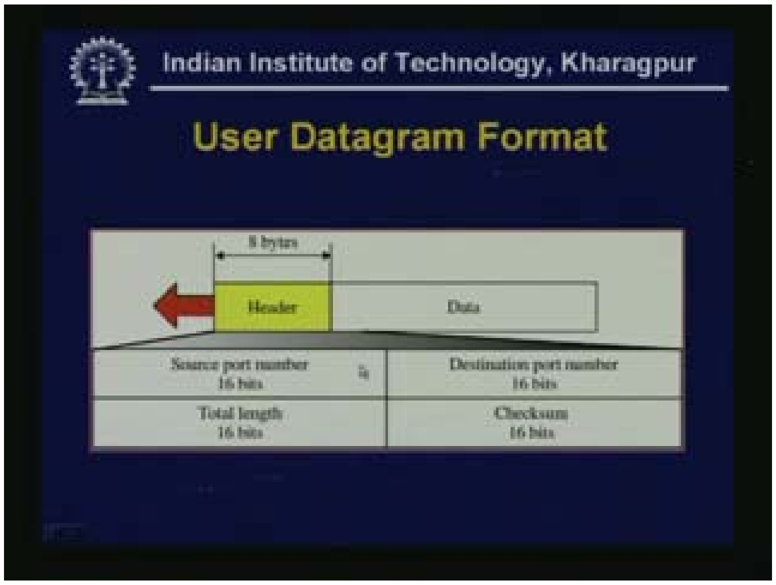
When a particular application wants to communicate to another application or when a process is trying to communicate to another process which is in a remote machine, in that case it will want to talk to some particular machine. Up to whatever we have seen, a particular machine means a particular IP address. Now this IP address is handled by the IP layer that means by the network layer on both the machines. Then how does the network layer get this IP address? As such it is supposed to get the IP address, the destination source and destination IP address from the top but there is a transport layer between the IP layer and the application layer.

We are talking mostly about the TCP/IP protocol stack so we are not considering talking about OSI protocol stack and presentation layer for the time being. The destination IP address is known to the application but it has to be communicated to the network layer. The transport layer has got nothing to do with the IP addresses therefore the application layer does communicate the source and destination IP addresses to the transport layer and the transport layer passes it on to the network layer and does not do anything with it except considering it for the check sum. So this is known as IP pseudo header because this is not a real header for the UDP protocol or TCP.

This IP pseudo header contains the 32-bit source IP address, 32-bit destination IP address and then this 16-bit UDP length etc. So this is the header which is passed on to the IP layer and this is in the same stack in the same machine. These headers are actually meant for communication between peers that means the transport layer, the UDP on this machine and the UDP on that machine will communicate via this UDP header so this is the proper header. And the pseudo header just takes that information about IP addresses etc from above and passes it below.

What the real UDP header uses to communicate with its peer on the other machine contains a 16-bit source port number and 16-bit destination port number. Remember, these were 32-bit source IP address and here these are 32-bit destination IP address. There is also the port number here which we will see later. We have 16-bit source port number, 16-bit destination port number, 16-bit UDP length, optional 16-bit UDP checksum which may be optional and data if any, possible odd bytes and a pad.

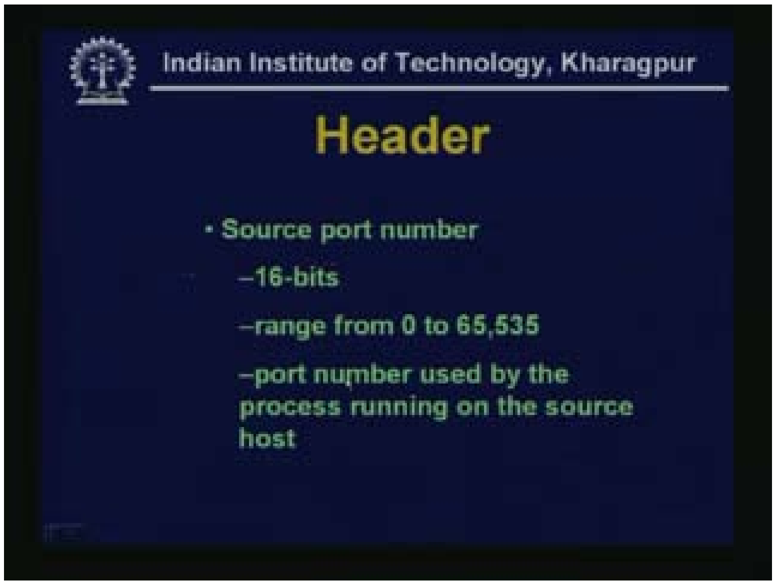
(Refer Slide Time: 10:25)


Indian Institute of Technology, Kharagpur

User Datagram Format

So this is the length of the header and data. Therefore we have the source and destination port numbers, total length and checksum.

(Refer Slide Time: 12:20)


Indian Institute of Technology, Kharagpur

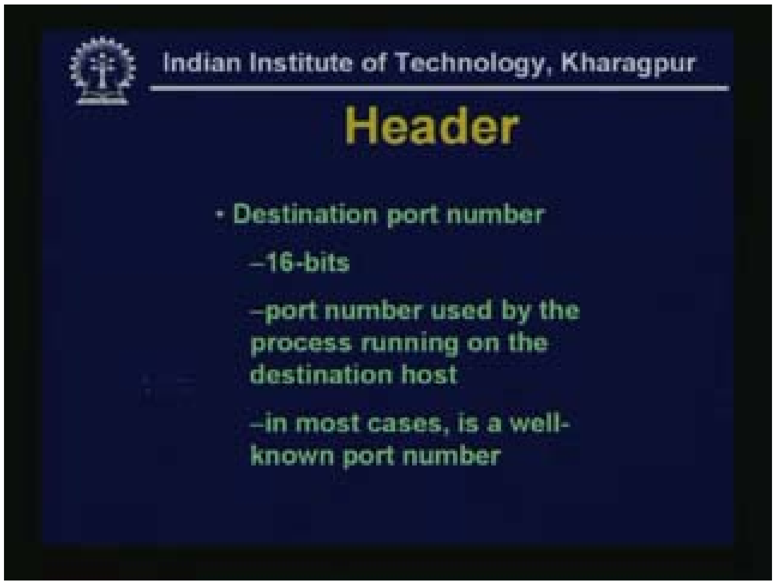
Header

- Source port number
  - 16-bits
  - range from 0 to 65,535
  - port number used by the process running on the source host

Let us go through this in detail. The source port number has
- 16-bits.
- Range from 0 to 65,535.
- Port number used by the process running on the source host.

Recall the multiplexing and de-multiplexing we discussed earlier. In the same machine number of processes may be communicating and out of all these processes a good number of them may be using the same transport protocol namely UDP while others could be using the transfer protocol TCP. When a particular packet comes, you can see whether it is a TCP packet or UDP packet. Once you make it out, you have to decide that out of all the processes using UDP for which process is this packet meant for. All these different processes are associated with different port numbers and looking at the port number the UDP decides the process for which it is meant for and this is the de-multiplexing part. But the multiplexing part comes with the source port number. Hence there is a source port number and the destination port number used by UDP for multiplexing and de-multiplexing at its own level. This is a number which is 16-bits long which will give you from 0 to 65,535. These are the port numbers and port number is used by the process running on the source host. Later on let us see in detail on how this port number is obtained.

(Refer Slide Time: 12:50)

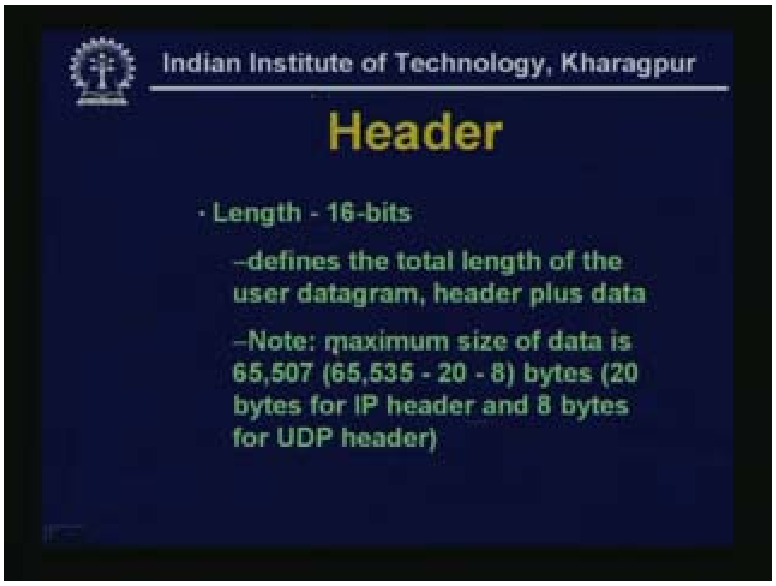

Indian Institute of Technology, Kharagpur

Header

- Destination port number
  - –16-bits
  - –port number used by the process running on the destination host
  - –in most cases, is a well-known port number

Similarly, there is a destination port number:
- It has got 16-bits.
- Port number used by the process running on the destination host.
- In most cases, it is a well known port number. We will what is meant by well known port number.

(Refer Slide Time: 13:15)


Indian Institute of Technology, Kharagpur

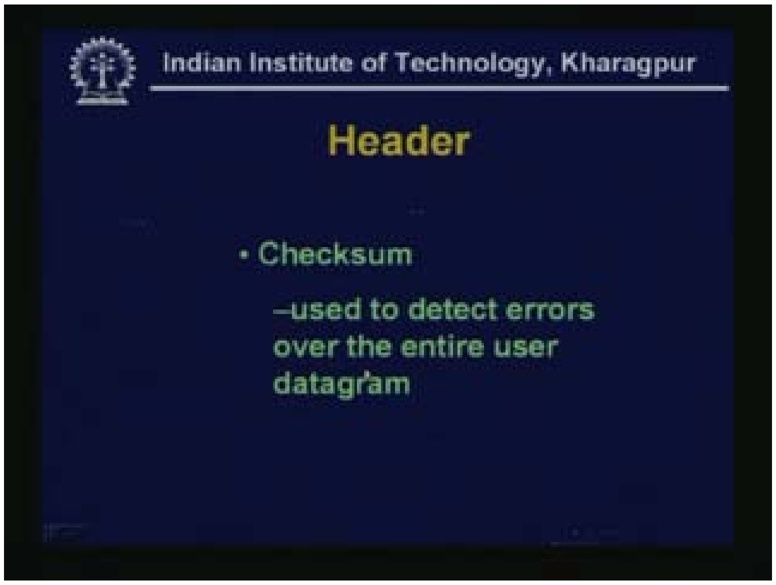
Header

- Length - 16-bits
  - –defines the total length of the user datagram, header plus data
  - –Note: maximum size of data is 65,507 (65,535 - 20 - 8) bytes (20 bytes for IP header and 8 bytes for UDP header)

- The length is 16-bits.
  - o It defines the total length of the user datagram, header plus data.

o Note: Maximum size of data is 65,507 after subtracting 20 bytes for IP header and 8 bytes for UDP header as this is the overhead.

(Refer Slide Time: 13:35)



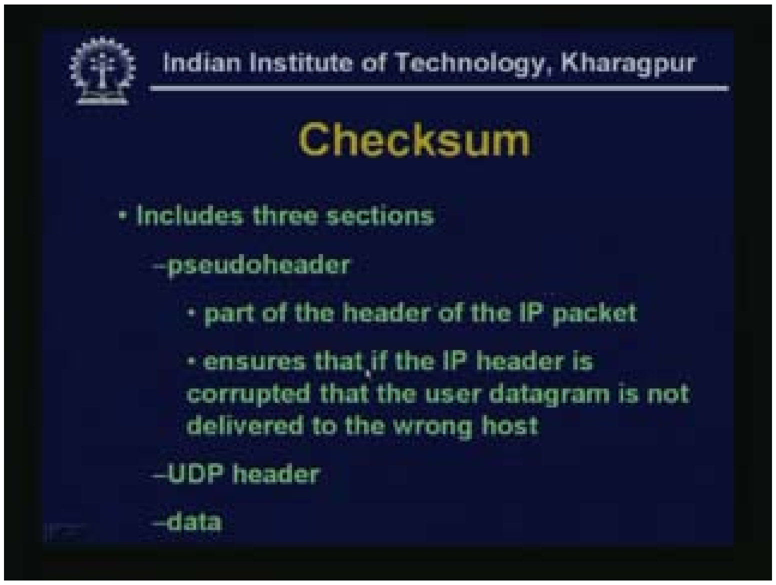
Indian Institute of Technology, Kharagpur

Header

- Checksum
  - –used to detect errors over the entire user datagram

- Checksum is 16 bits
  - o Used to detect errors over the entire user datagram.

(Refer Slide Time: 14:00)



Indian Institute of Technology, Kharagpur

Checksum

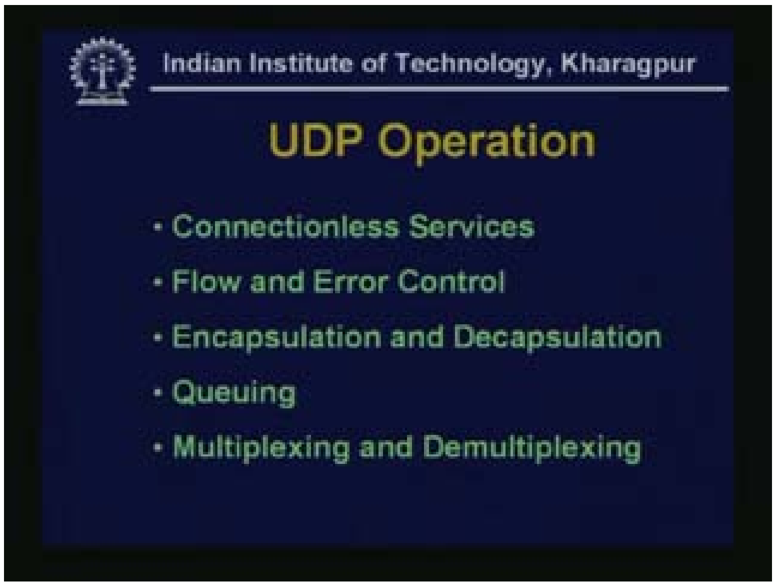
- Includes three sections
  - –pseudoheader
    - part of the header of the IP packet
    - ensures that if the IP header is corrupted that the user datagram is not delivered to the wrong host
  - –UDP header
  - –data

- Checksum includes three sections:
  - o Pseudoheader UDP header and the data.

- Part of the header of the IP packet.
- Ensures that if the IP header is corrupted the user datagram is not delivered to the wrong host. Since the pseudoheader contains the destination IP address, even if the IP header gets corrupted this does not get delivered to the wrong host. The pseudoheader, UDP header and the data taken together forms the payload for the next layer that is the IP layer and the checksum is computed over this entire body. So there is some amount of error checking and error detection done. This is done by UDP and this is the extent to which it will go for providing reliability. Beyond this if the entire packet is lost somewhere, UDP cannot do anything about it.

o UDP Header - There are the four fields in the header, source port number, destination port number, total length and the checksum.

(Refer Slide Time: 14:47)



Indian Institute of Technology, Kharagpur

UDP Operation

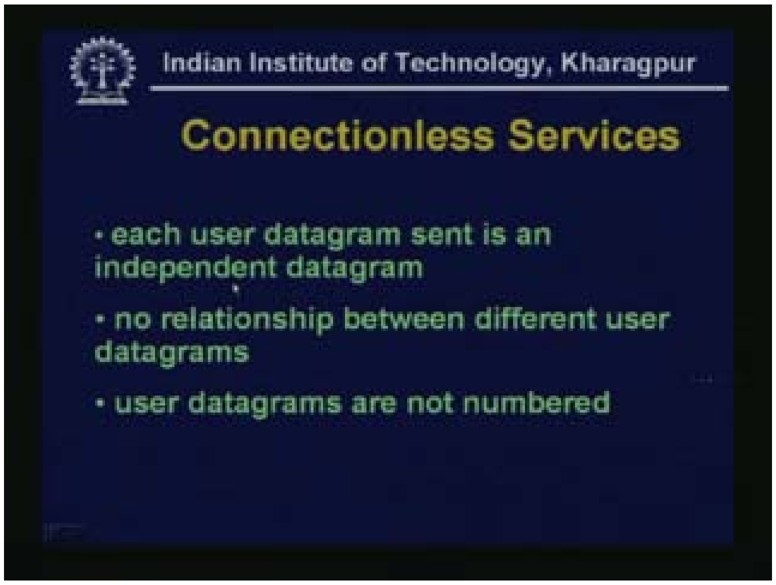
- Connectionless Services
- Flow and Error Control
- Encapsulation and Decapsulation
- Queuing
- Multiplexing and Demultiplexing

UDP operation:

- It is a connectionless service.
- Has very minimal flow and error control as given by the checksum.
- Performs encapsulation and decapsulation and forming of packets.
- Queuing.
- Multiplexing and De-multiplexing.

Let us look at these operations one by one.

(Refer Slide Time: 17:00)


Indian Institute of Technology, Kharagpur

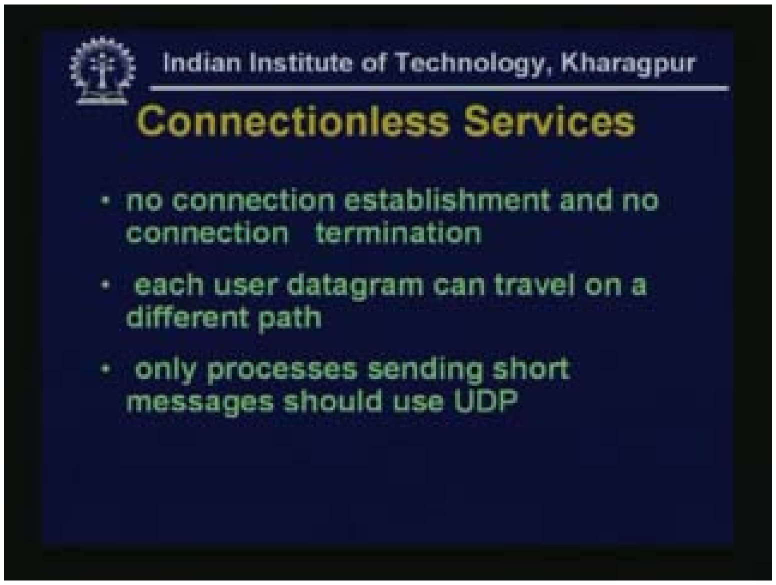
Connectionless Services

- each user datagram sent is an independent datagram
- no relationship between different user datagrams
- user datagrams are not numbered

Connectionless service:

- Each user datagram sent is an independent datagram. If an application sends one UDP and is going to send another then they are handled independently in the layers below this application. These two datagrams may be coming from the same source application process meant for the same destination application process, but they are going to be treated independently by the rest of the network layers. First of all, it may so happen that these two packets may go in two different directions, may be routed differently because there is no connection. This is completely a datagram oriented connectionless service. Therefore these two datagrams may travel in different paths. One of them may get lost or they may get out of order which means the datagram that was sent earlier may reach the destination quite later. UDP is not going to take any responsibility for all these mishaps and it is taken for granted that whatever application takes place using this UDP is resilient to such things.
- There is no relationship between different user datagrams.
- User datagrams are not numbered. Meaning that if the datagram which was sent later arrives much earlier and the vice versa then there is no way of knowing unless in the application layer itself we have taken some care to identify that.

(Refer Slide Time: 17:27)



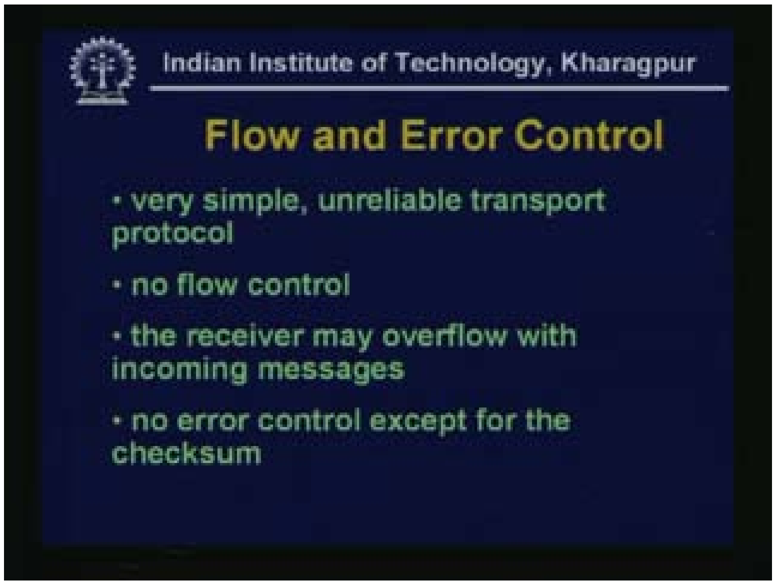
Indian Institute of Technology, Kharagpur

Connectionless Services

- no connection establishment and no connection termination
- each user datagram can travel on a different path
- only processes sending short messages should use UDP

- There is no connection establishment. Since this is completely a connectionless service there is no question of any connection establishment, and since there is no connection establishment there is no connection termination either.
- Each user datagram can travel on a different path.
- Only processes sending short messages should use UDP. Usually UDP is used in a case where you send one packet and that is the end of it. When you are trying to the send a stream of packets or stream of bytes etc usually you do not use UDP.

(Refer Slide Time: 18:20)



Indian Institute of Technology, Kharagpur

Flow and Error Control

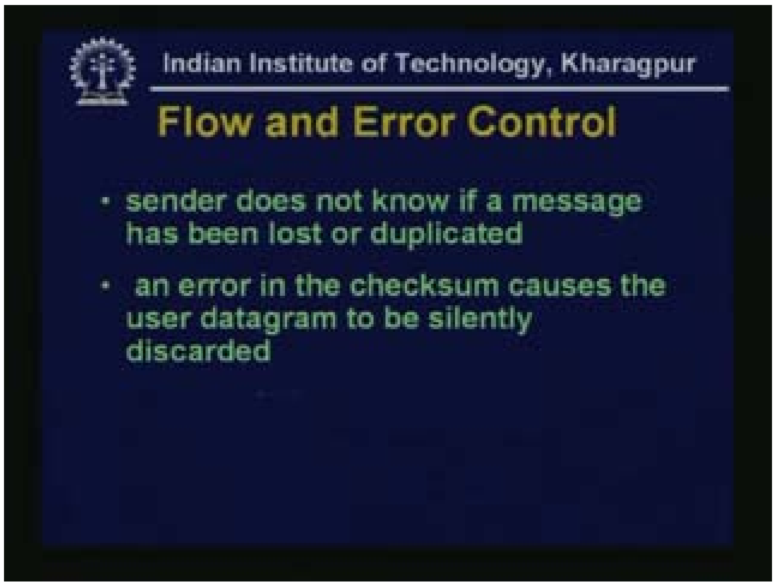
- very simple, unreliable transport protocol
- no flow control
- the receiver may overflow with incoming messages
- no error control except for the checksum

As far as flow and error control is concerned which may be another job of the transport layer the UDP does very little.

- It is very simple and an unreliable transport protocol.
- There is no flow control.
- The receiver may overflow with incoming messages.
- There is no error control except for the checksum.

What happens is that, suppose the receiver is getting a large number of UDP packets, and if it wants to just stop or slow down to the senders it cannot do that. If it overflows some of the packets will get lost. And then there is no error control except for the checksum.

(Refer Slide Time: 19:00)



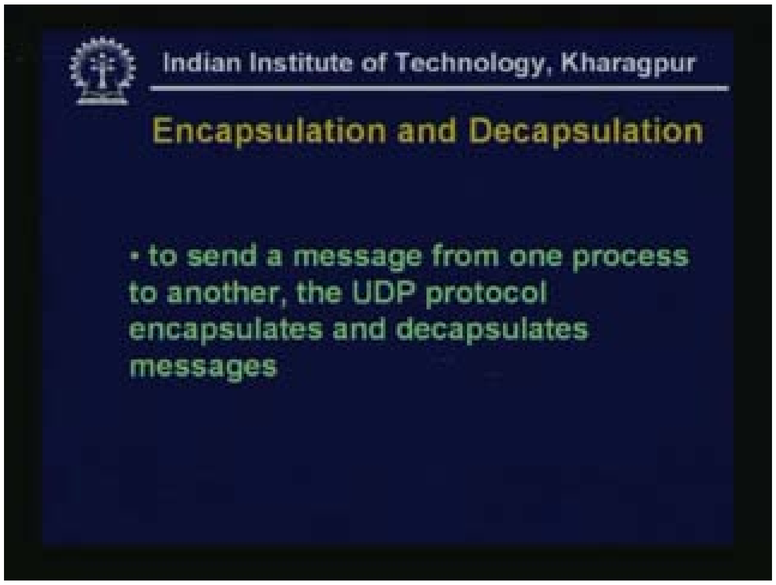
Indian Institute of Technology, Kharagpur

Flow and Error Control

- sender does not know if a message has been lost or duplicated
- an error in the checksum causes the user datagram to be silently discarded

- Sender does not know if a message has been lost or if it has been duplicated.
- An error in the checksum causes a user datagram to be silently discarded. This is a very simple protocol and this is very efficient and has a very low overhead.

(Refer Slide Time: 19:55)



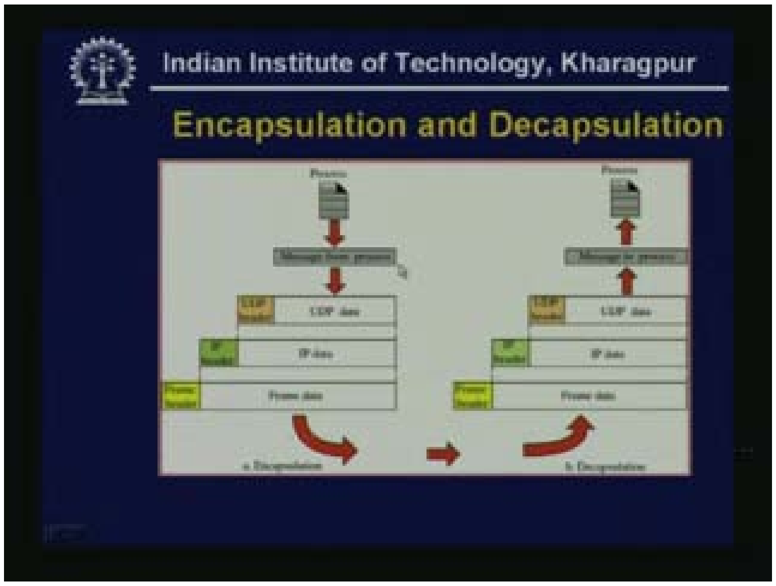
Indian Institute of Technology, Kharagpur

Encapsulation and Decapsulation

- to send a message from one process to another, the UDP protocol encapsulates and decapsulates messages

UDP does encapsulation and de-capsulation which is a fundamental job of any protocol in the transport layer. It has to form the packets, the packets form from the streams of data supplied by the application layer. The data is chopped into pieces, a header is added to each piece and then is passed on. To send a message from one process to another the UDP protocol encapsulates and decapsulates messages.

(Refer Slide Time: 19:59)

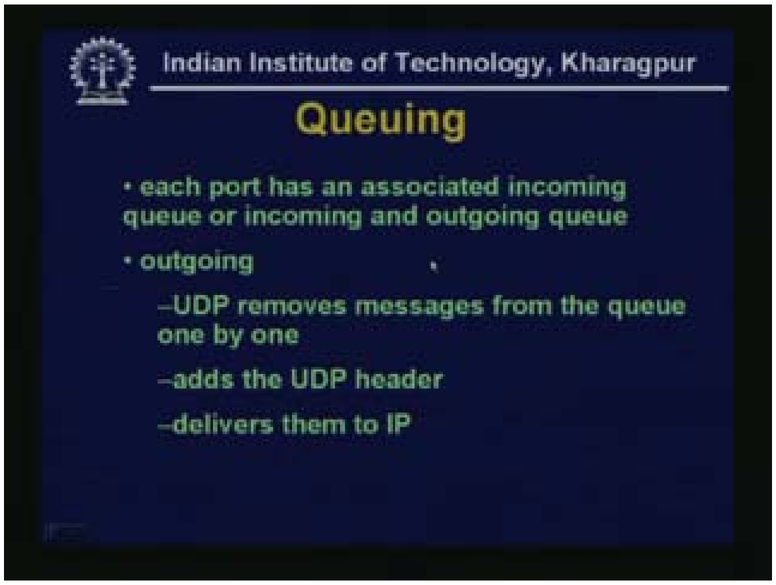


Indian Institute of Technology, Kharagpur

Encapsulation and Decapsulation

This is just a standard way as it goes down the stack, the message from the process, so this is the UDP data, this is the UDP header, then the IP header frame header etc and it is decapsulated in a similar fashion.

(Refer Slide Time: 20:40)

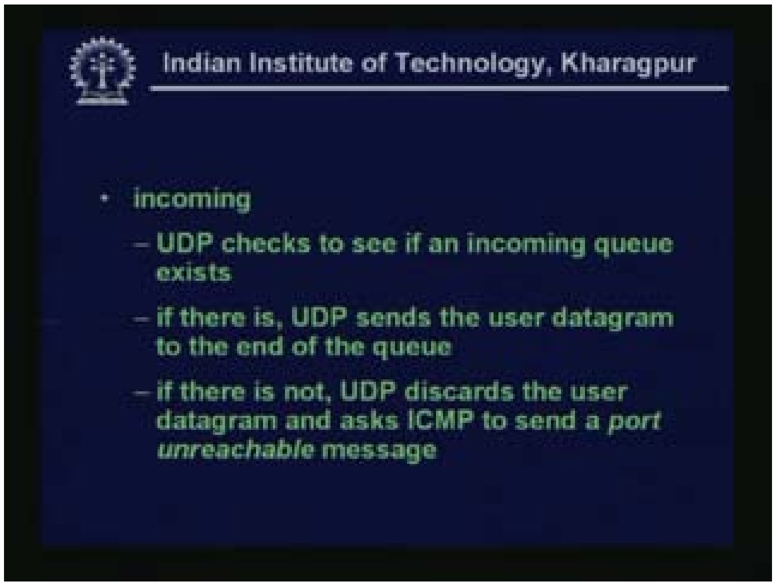

Indian Institute of Technology, Kharagpur

Queuing

- each port has an associated incoming queue or incoming and outgoing queue
- outgoing
  - UDP removes messages from the queue one by one
  - adds the UDP header
  - delivers them to IP

- Queuing:
- Each port has an associated incoming queue or incoming and outgoing queue depending on which way the communication is taking place. A port is not just a number. Along with the particular number, inside the OS there will be a queue of data being communicated in the machine. There will be an incoming queue and an outgoing queue.
- UDP removes message from the outgoing queue one by one.
- Adds the UDP header.
- Delivers them to the IP.

(Refer Slide Time: 22:15)

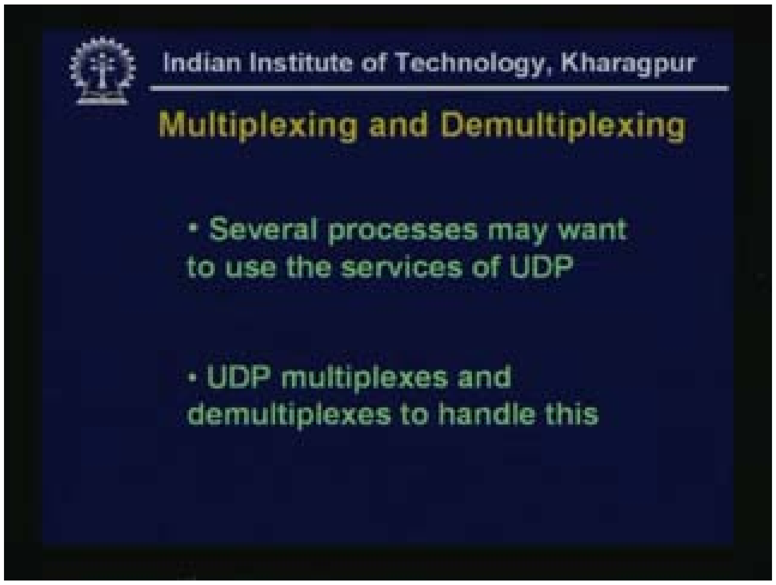

Indian Institute of Technology, Kharagpur

- incoming
  - UDP checks to see if an incoming queue exists
  - if there is, UDP sends the user datagram to the end of the queue
  - if there is not, UDP discards the user datagram and asks ICMP to send a *port unreachable* message

Incoming queue:

- UDP checks to see if an incoming queue exists. The first thing UDP does is to check whether the incoming queue exists.
- If there is, UDP sends the user datagram to the end of the queue. The particular application process is going to consume from this incoming queue. The process will consume from this incoming queue which is being fed from the UDP. For the outgoing queue, the application process is going to put things in the queue and UDP will take out the message to be communicated from the queue.
- If there is no such port then UDP discards the user datagram and asks ICMP to send a port unreachable message. This means that a particular packet has come meant for a particular port and that port does not exist in this machine so it will send them an error message saying that the port is unreachable. In order to generate this ICMP message sometimes a packet is sent to an improvable port number. So, when you get back you know that you have reached the destination but of course there is no application over there.

(Refer Slide Time: 23:25)



Indian Institute of Technology, Kharagpur

Multiplexing and Demultiplexing

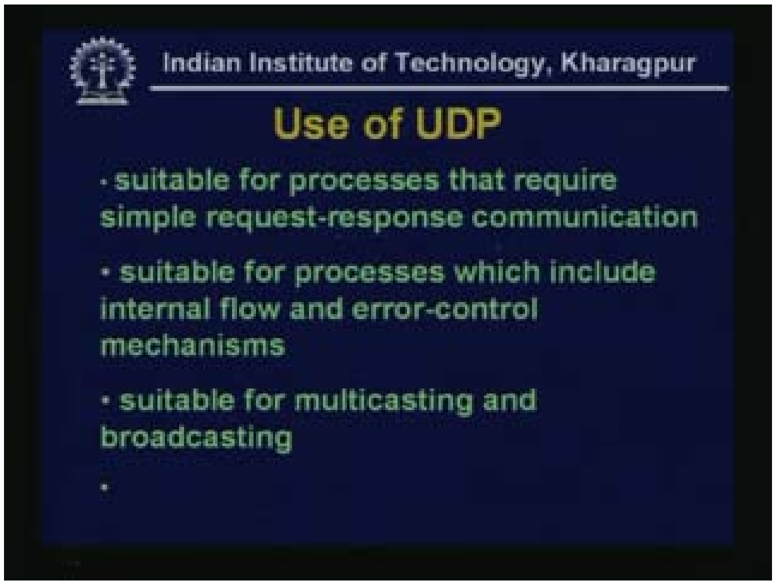
• Several processes may want to use the services of UDP

• UDP multiplexes and demultiplexes to handle this

It does multiplexing and demultiplexing.

- Several processes may want to use the services of UDP.
- UDP multiplexes and demultiplexes to handle this.

(Refer Slide Time: 23:40)


Indian Institute of Technology, Kharagpur

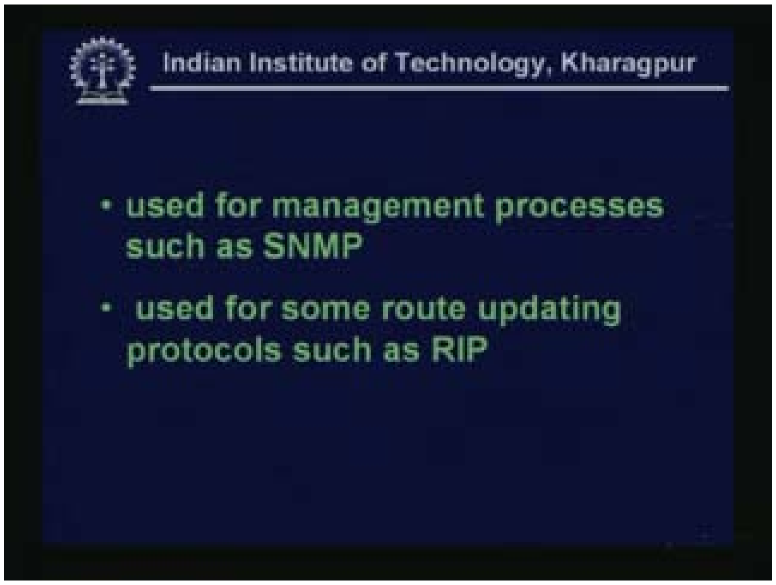
Use of UDP

- suitable for processes that require simple request-response communication
- suitable for processes which include internal flow and error-control mechanisms
- suitable for multicasting and broadcasting
-

Use of UDP:

- This is suitable for processes that require simple request-response communication. Suppose there is a simple request which may be sent by UDP and there is just one message as response to this request then that may also be sent back as an UDP. This also depends on the kind of network you have. Suppose you are communicating only inside the LAN, in that case you may not like to have an overhead since this is not going over the WAN. Therefore this is expected to be much more reliable and you may not like to incur any extra cost for providing reliability etc because you know that the underlying network is quite reliable. And secondly if your process is such that there is a simple message coming and there is a simple response, in that case a simple UDP protocol may be sufficient.
- This may also be suitable for processes which include internal flow and error control mechanisms. That means, if the application process itself is handling some flow and error control, that means if it is handled at an upper layer then you do not want to duplicate it in the transport layer, in which case you use of simple protocol like UDP which is more efficient because reliability is being handled by somebody else anyway. So that is another case where UDP is a very suitable protocol.
- UDP may also be suitable for multicasting and broadcasting.

(Refer Slide Time: 25:35)

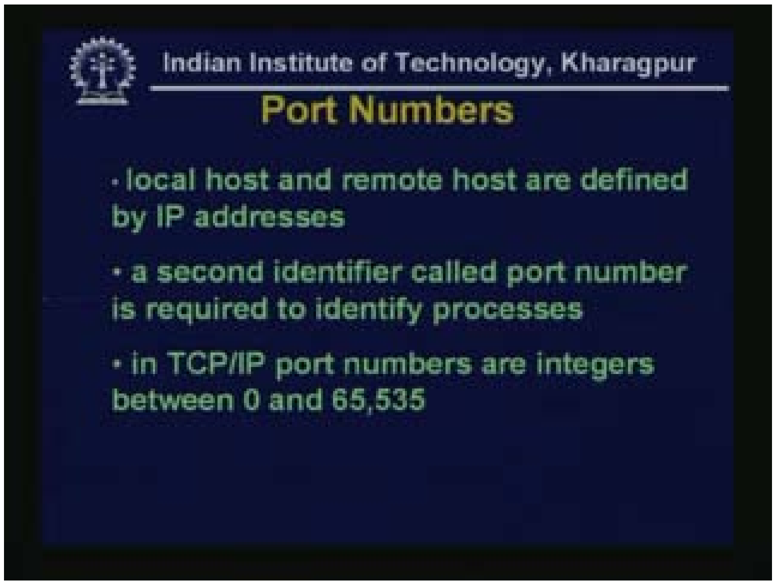

Indian Institute of Technology, Kharagpur

- used for management processes such as SNMP
- used for some route updating protocols such as RIP

Going to some specific examples:

- This is used for management processes such as SNMP. SNMP is a Simple Network Management Protocol used for managing network. For managing network we have this central network management software which from time to time may probe different network boxes to see if their health is alright, collect all kinds of statistics etc. So this is a simple message response kind of system. It asks for queries, manages a particular device in the network by sending a message and that device responds with some statistics or some alarm or whatever it is. So that is a case where UDP protocol may be quite suitable.
- This is used for some route updating protocols such as RIP. We have already discussed RIP. For updating the routes the routers have to communicate with each other and that uses UDP. These are just two examples. There are many other examples and user applications which may also use UDP.

(Refer Slide Time: 27:25)


Indian Institute of Technology, Kharagpur

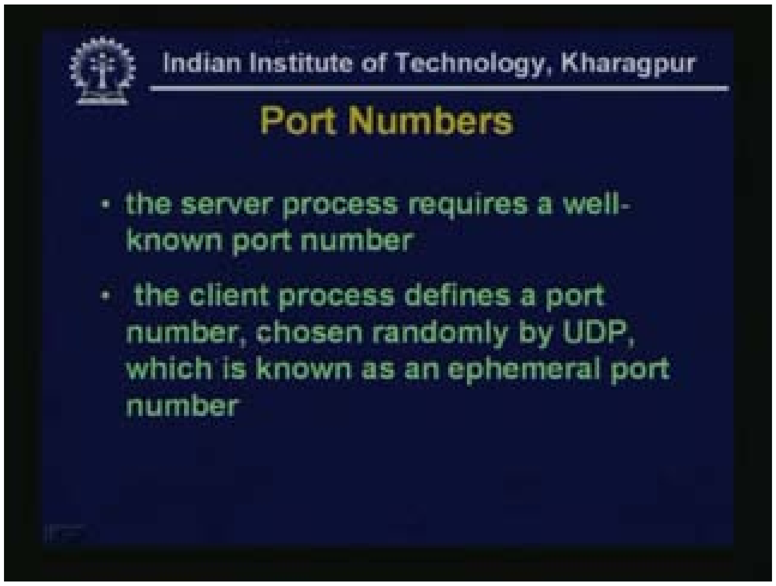
Port Numbers

- local host and remote host are defined by IP addresses
- a second identifier called port number is required to identify processes
- in TCP/IP port numbers are integers between 0 and 65,535

Now let us discuss about port numbers and then I will talk about client server. Port Numbers:

- Local host and remote host are defined by IP addresses. Here the host refers to the machine.
- A second identifier called port number is required to identify processes because many processes may be running on the same host. So just identifying the host is not enough but we have to identify the process in this host. Or more specifically you have to talk about this process and this host. So you have to have a port number as well as an IP address.
- In TCP/IP port numbers are integers between 0 and 65,535.

(Refer Slide Time: 27:55)


Indian Institute of Technology, Kharagpur

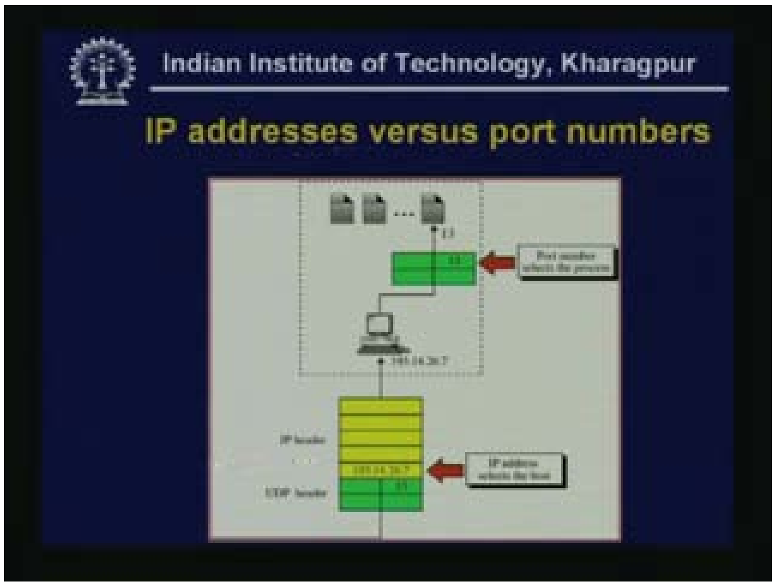
Port Numbers

- the server process requires a well-known port number
- the client process defines a port number, chosen randomly by UDP, which is known as an ephemeral port number

- The server process requires a well known port number.
- The client process defines a port number chosen randomly by UDP which is known as an ephemeral port number.
-

We will discuss this in more detail when we discuss more about client server. This is one of the ways you write applications in a network environment by using the client server paradigm. There will be different servers giving different services like you may have a web server giving you web services or mail server etc. Different clients could be using these services. So they use different sets of port numbers. This will be clear when we talk about the client server paradigm in more detail.

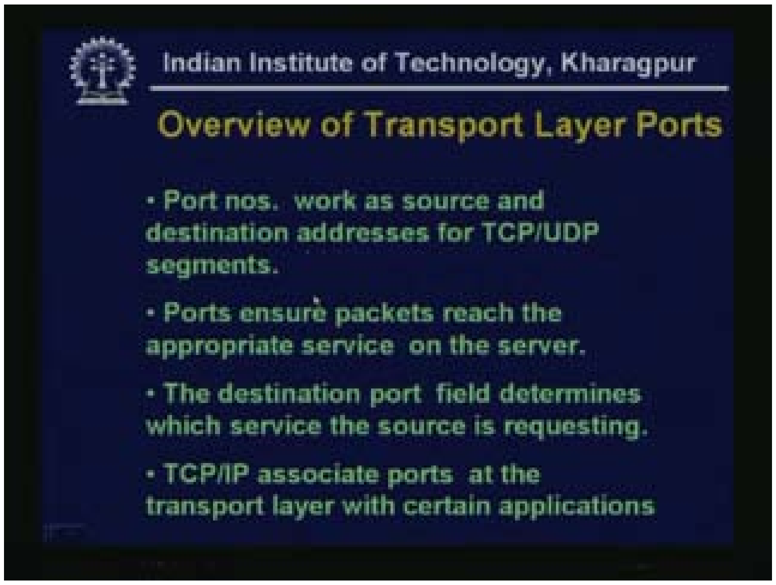
(Refer Slide Time: 29:00)


Indian Institute of Technology, Kharagpur

IP addresses versus port numbers

Once again, if you think about the IP address, this selects the host and then the port number selects the process. When you give the destination IP address etc it reaches a particular host and in that host it looks at the port number and then selects that process using that particular port number.

(Refer Slide Time: 29:23)

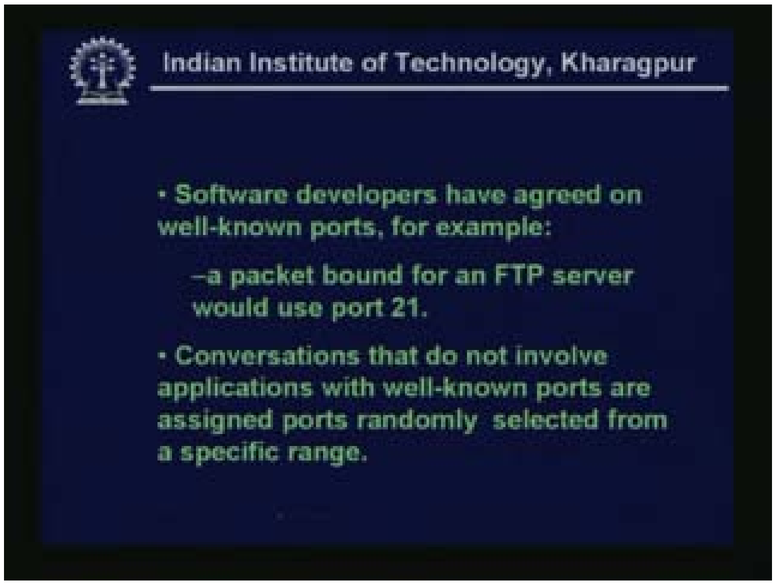

Indian Institute of Technology, Kharagpur

Overview of Transport Layer Ports

- Port nos. work as source and destination addresses for TCP/UDP segments.
- Ports ensure packets reach the appropriate service on the server.
- The destination port field determines which service the source is requesting.
- TCP/IP associate ports at the transport layer with certain applications

- Port numbers work as source and destination addresses for TCP/UDP segments.
- Ports ensure packets reach appropriate service on the server.
- The destination port field determines which service the source is requesting.
- TCP/IP associate ports at the transport layer with certain applications.

(Refer Slide Time: 29:40)


Indian Institute of Technology, Kharagpur

- Software developers have agreed on well-known ports, for example:
  - –a packet bound for an FTP server would use port 21.
- Conversations that do not involve applications with well-known ports are assigned ports randomly selected from a specific range.

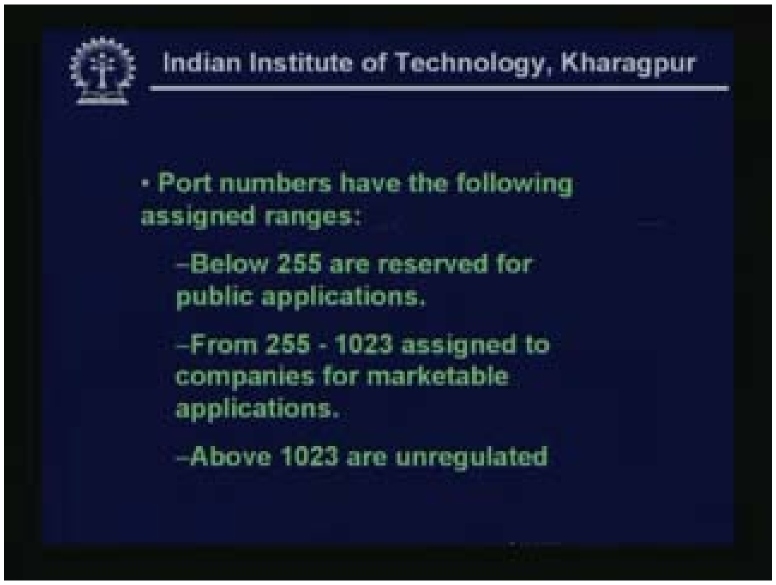
- Software developers have agreed on well known ports, for example:

o A packet bound for an FTP server would use port 21. Just think of some network service, a web service. When you are surfing the net you will use a browser and you may click on a particular URL or network address and you immediately get the opening page of that particular site displayed by your browser. If you think about how did this happen? A particular site is hosted in some remote machine somewhere. You may know only its IP address. Now, by knowing its IP address you have to make a request to that IP address, like requesting it to show the opening page. Now the IP address will help you through this mace of routers who may be running some routing protocol like RIP, OSPF so that your request reaches the destination machine. But in the destination machine how does it know the port number? You do not know about the destination machine, number of processes running etc. Suppose if it is a HTTP request that means it is a request for a web page to a web server then this request has to reach the web server.

Now there may be ten or fifty other processes running on that same machine, it should not go to any of these other servers. So, for very standard applications like FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol), HTTP (Hyper Text Transfer Protocol) and so on are all applications. So, for all these protocols the port numbers are fixed. So, if I make an FTP request, as it says FTP server use a well known port 21. That means if I make an FTP request to any machine anywhere in the world I am going to use a port number 21 knowing that this is a well known port number and if it is reaches the destination machine port number 21 would mean the FTP server.

- Conversations that do not involve applications with well known ports are assigned ports randomly selected from a specific range.

(Refer Slide Time: 33:05)



Indian Institute of Technology, Kharagpur

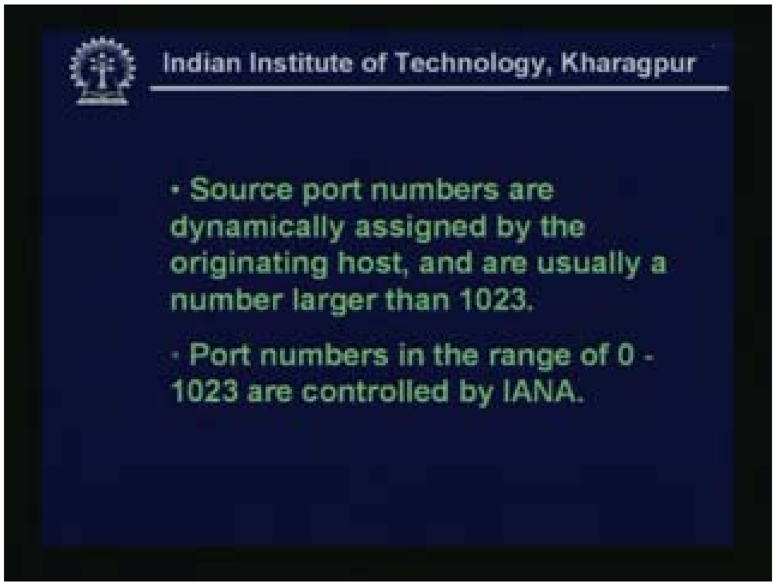
- Port numbers have the following assigned ranges:
  - Below 255 are reserved for public applications.
  - From 255 - 1023 assigned to companies for marketable applications.
  - Above 1023 are unregulated

There is an organization IANA which handles these port numbers. So port numbers have the following assigned ranges:

- Below 255 (0 to 255) are reserved for public applications like FTP, SMTP, HTTP etc.
- From 255 to 1023 these are assigned to companies for marketable applications.
- Above 1023 these are unregulated which means that up to 1024 these are reserved. This is again divided into two parts, one part for the public applications and the other for some vendor specific applications. Apart from well known port numbers you also need a whole lot of other

port numbers. Take the previous example where we made an HTTP request to a web server. Now the web server will send you back something, may be the contents of the first page of its website. And that is going to be sent to the requester. For this another port number is temporarily assigned and this is assigned from a number range from 1024 to 65,535 and the number is randomly chosen. This is an ephemeral port, it is not a fixed port and it will be held constant for the duration of this communication and then it will be released for use by some other process.

(Refer Slide Time: 34:35)

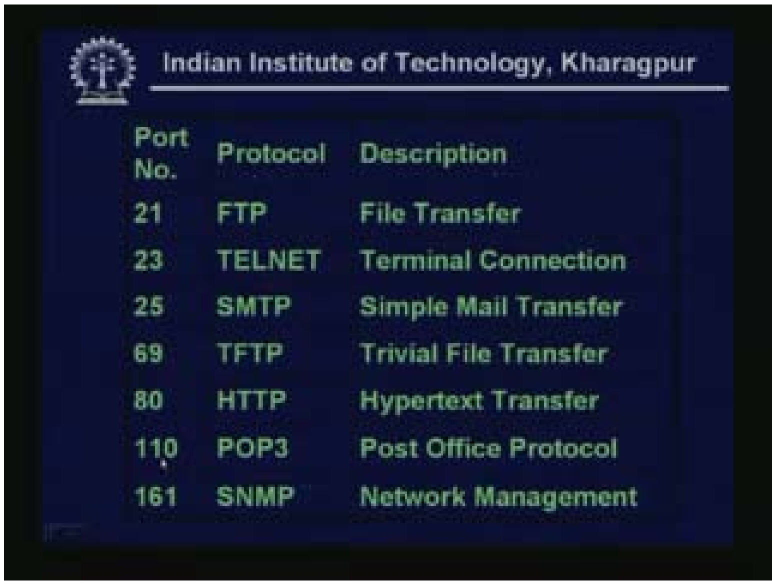


Indian Institute of Technology, Kharagpur

- Source port numbers are dynamically assigned by the originating host, and are usually a number larger than 1023.
- Port numbers in the range of 0 - 1023 are controlled by IANA.

- Source port numbers are dynamically assigned by the originating host and are usually a number larger than 1023.
- Port numbers in the range of 0 to 1023 are controlled by IANA.

(Refer Slide Time: 34:55)


Indian Institute of Technology, Kharagpur

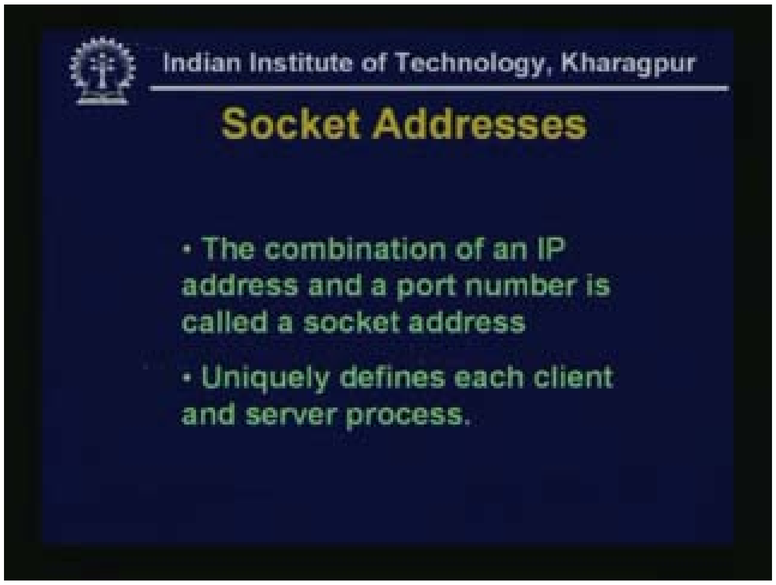
Port No.	Protocol	Description
21	FTP	File Transfer
23	TELNET	Terminal Connection
25	SMTP	Simple Mail Transfer
69	TFTP	Trivial File Transfer
80	HTTP	Hypertext Transfer
110	POP3	Post Office Protocol
161	SNMP	Network Management

These are some examples of well-known port numbers. There are quite a good number of them but I have just mentioned some important protocols.

- FTP is a file transfer protocol uses port number 21.
- TELNET, a terminal connection uses port number 23. I will be talking more about some of these application layer protocols in a later lecture. There are hundreds of applications which have come up and we cannot talk about all of them but we will talk about few of them towards the last part of our course.
- SMTP is a Simple Mail Transfer Protocol that uses port number 25.
- TFTP is a Trivial File Transfer, when you just have to send a short message it uses port number 69.
- HTTP is the Hyper Text Transfer Protocol which is the one used for web services that uses well-known port number 80.
- POP3 is a Post Office Protocol that uses the port number 110. What POP3 does is that, suppose you got mail in your mail box in the local mail server then on your desktop you can download all the mails from the local server to your machine through the Post Office Protocol.
- SNMP is used for network management that uses port 161.

There are a whole lot of others but I have mentioned only a few which are important.

(Refer Slide Time: 36:46)



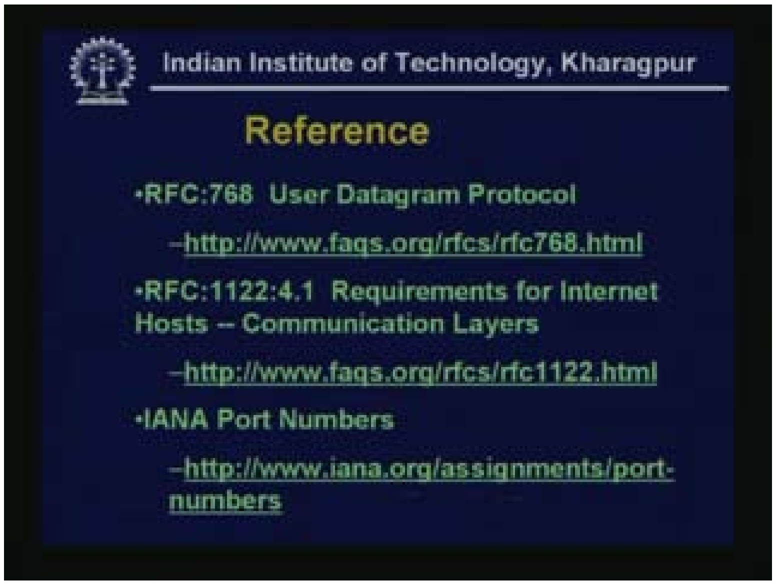
Indian Institute of Technology, Kharagpur

Socket Addresses

- The combination of an IP address and a port number is called a socket address
- Uniquely defines each client and server process.

Now moving on to the full description of client server programming it uses what is known as a socket address.
Socket Address:
- Socket address is a combination of an IP address and a port number.
- This uniquely defines each client and server process. So, this process is in this machine. Therefore this process is given by the port number and this machine is given by the IP number.

(Refer Slide Time: 37:25)



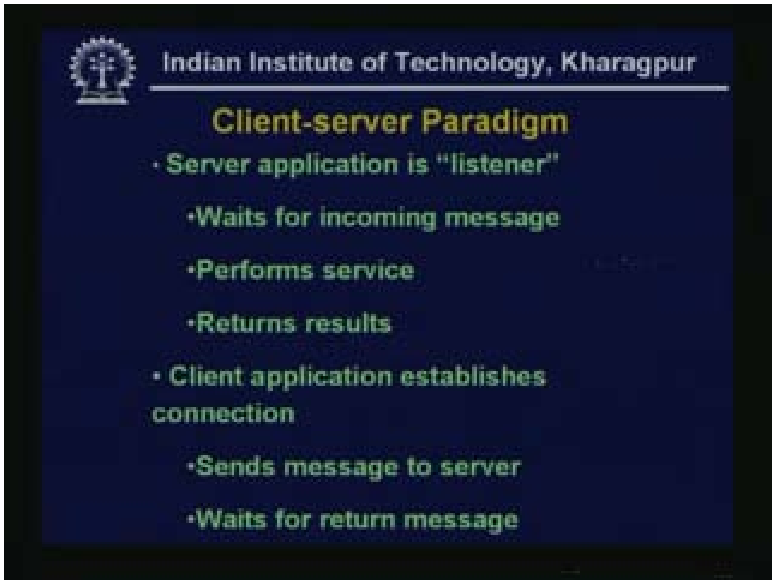
Indian Institute of Technology, Kharagpur

Reference

- RFC:768 User Datagram Protocol
  - http://www.faqs.org/rfcs/rfc768.html
- RFC:1122:4.1 Requirements for Internet Hosts -- Communication Layers
  - http://www.faqs.org/rfcs/rfc1122.html
- IANA Port Numbers
  - http://www.iana.org/assignments/port-numbers

These are some of the references for the different RFCS and the UDP is described in RFC 768. Then there is RFC 1122 and IANA port numbers can be seen in the websites mentioned here.

(Refer Slide Time: 37:43)


Indian Institute of Technology, Kharagpur

Client-server Paradigm

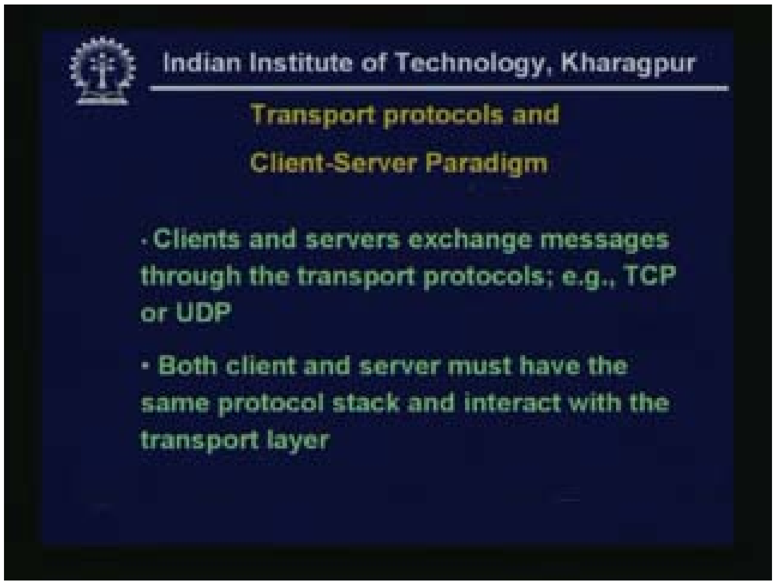
- Server application is "listener"
  - Waits for incoming message
  - Performs service
  - Returns results
- Client application establishes connection
  - Sends message to server
  - Waits for return message

Let us see some more about client-server paradigm which is sort of universally used for network based application.

- Server application is listener
  - o Waits for incoming message
  - o Performs service
  - o Returns results
- Client application establishes connection:
  - o It sends the message to the server and then it waits for a return message.

In this client-server paradigm usually what happens is that, the client would initiate the request. It initiates the request for some service. It has to know the address of the service and if it is a well-known service then it will use the well-known port number. So in that particular machine in that particular port number it will make the request. And the server will accede to the request, in the sense that it will give the service and then return the result to the client and then it goes back to listening. So, the server is always in the listening mode that means it is waiting for a particular request to come in.

(Refer Slide Time: 38:57)



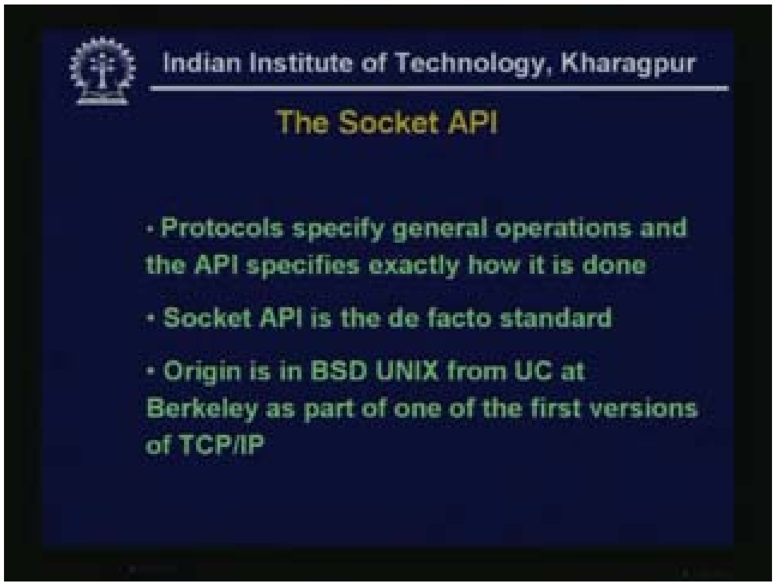
Indian Institute of Technology, Kharagpur

Transport protocols and Client-Server Paradigm

- Clients and servers exchange messages through the transport protocols; e.g., TCP or UDP
- Both client and server must have the same protocol stack and interact with the transport layer

- Clients and servers exchange messages through the transport protocols e.g. TCP or UDP.
- Both client and server must have the same protocol stack and interact with the transport layer. So you may have a client server using both TCP and UDP.

(Refer Slide Time: 39:10)



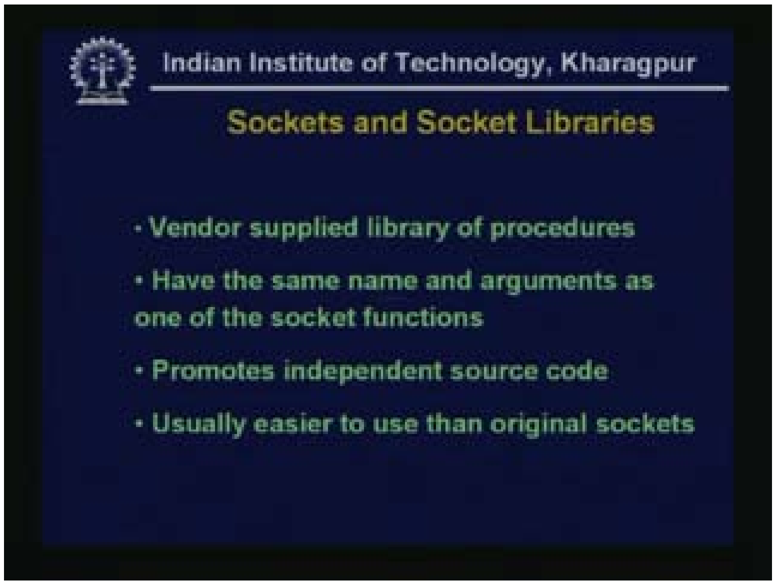
Indian Institute of Technology, Kharagpur

The Socket API

- Protocols specify general operations and the API specifies exactly how it is done
- Socket API is the de facto standard
- Origin is in BSD UNIX from UC at Berkeley as part of one of the first versions of TCP/IP

- Protocols specify general operations and the API specifies exactly how it is done. For using the socket there is a socket API. API stands for Application Program Interface.
- So the socket API is the de facto standard.
- Origin is in the BSD UNIX from University of California at Berkeley as part of one of the first versions of TCP/IP.

(Refer Slide Time: 39:47)



Indian Institute of Technology, Kharagpur

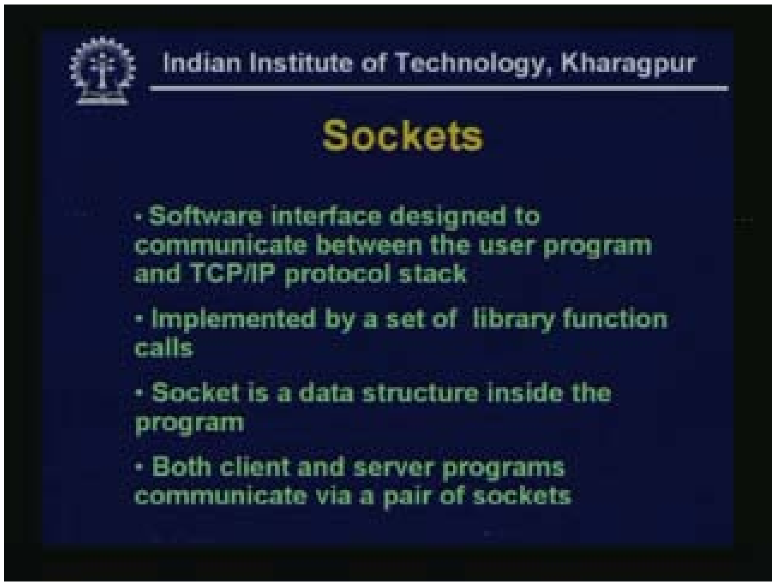
Sockets and Socket Libraries

- Vendor supplied library of procedures
- Have the same name and arguments as one of the socket functions
- Promotes independent source code
- Usually easier to use than original sockets

Let us see a little bit about the sockets and socket libraries:
- This is a vendor supplied library of procedures.
- They have the same name and arguments as one of the socket functions.
- It promotes independent source code and it is usually easier to use than original sockets.

The vendors also provide socket libraries with some bells and whistles.

(Refer Slide Time: 40:05)



Indian Institute of Technology, Kharagpur

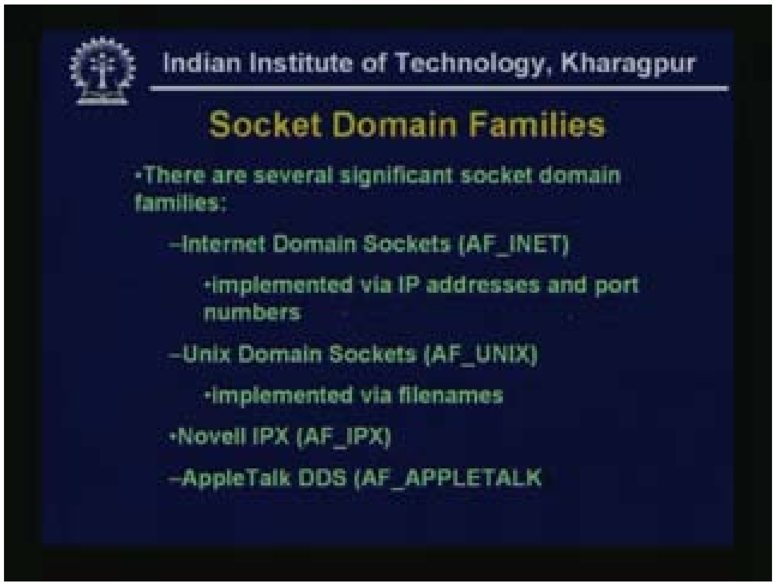
Sockets

- Software interface designed to communicate between the user program and TCP/IP protocol stack
- Implemented by a set of library function calls
- Socket is a data structure inside the program
- Both client and server programs communicate via a pair of sockets

- Socket software interface is designed to communicate between the user program and TCP/IP protocol stack. Internally that is the socket. All TCP/IP stacks that we talked about right from the bottom one to right up to the transport layer is already in the system. Now you want to develop an application which should be run over the network. In such a case you have to write your application program for any service you would like to give. Now this user program has to communicate with this TCP/IP protocol stack which is already in the machine and the intervening layer is the API for sockets so you go to this TCP/IP layer stack through the sockets.

- Therefore socket is a data structure inside the program.
- Both client and server programs communicate via pair of sockets.

(Refer Slide Time: 41:20)



Indian Institute of Technology, Kharagpur

Socket Domain Families

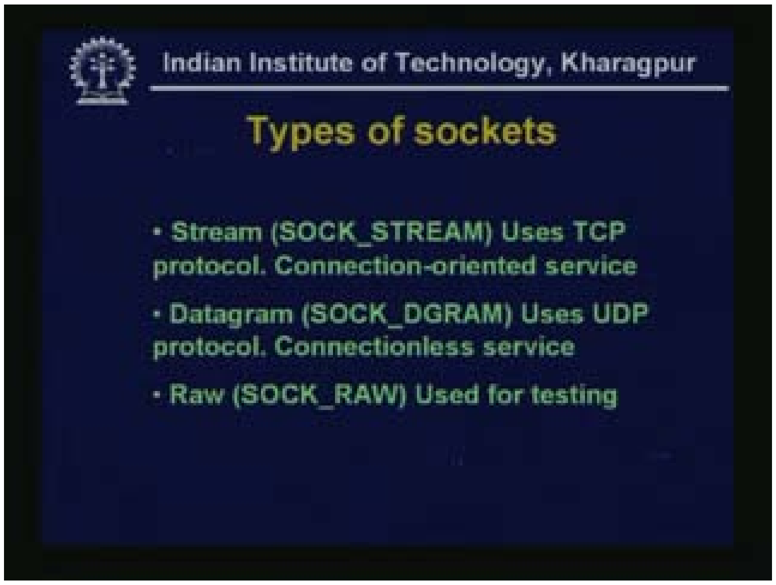
- There are several significant socket domain families:
  - Internet Domain Sockets (AF_INET)
    - implemented via IP addresses and port numbers
  - Unix Domain Sockets (AF_UNIX)
    - implemented via filenames
  - Novell IPX (AF_IPX)
  - AppleTalk DDS (AF_APPLETALK

There are several significant socket domain families:

- Internet domain sockets implemented via IP addresses and port numbers.
- UNIX domain sockets implemented via filenames.
- Novell IPX Apple Talk etc.

All these other vendor specific socket domains are also there. But the first two are the most important.

(Refer Slide Time: 41:45)


Indian Institute of Technology, Kharagpur

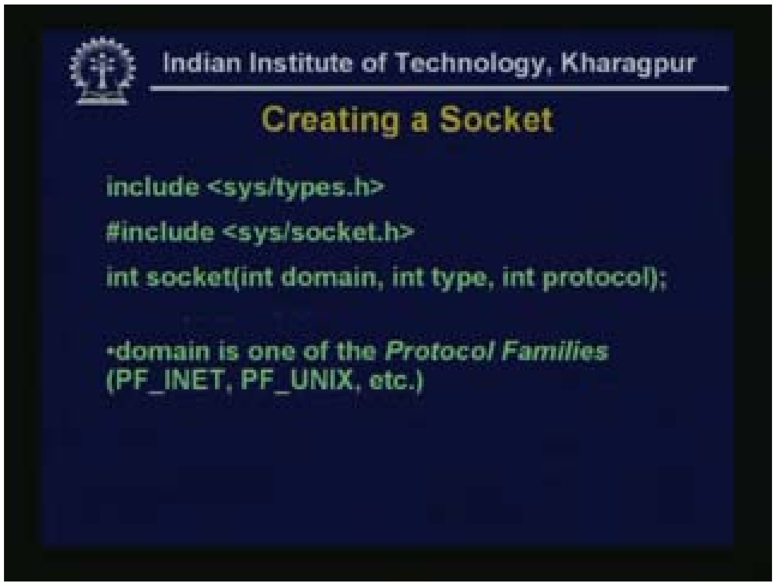
Types of sockets

- Stream (SOCK_STREAM) Uses TCP protocol. Connection-oriented service
- Datagram (SOCK_DGRAM) Uses UDP protocol. Connectionless service
- Raw (SOCK_RAW) Used for testing

There are three types of sockets:

- Stream: It uses a TCP protocol. Stream socket is a connection oriented service. So, naturally you cannot use stream socket with UDP. Hence the transport layer protocol to use with this stream socket is TCP.
- Datagram: In the datagram socket there is sock datagram that uses UDP protocol. The type of socket determines the type of protocol it is going to use TCP or UDP. But this uses the UDP protocol.
- There are the raw sockets which may be used for testing anything internally.

(Refer Slide Time: 42:30)


Indian Institute of Technology, Kharagpur

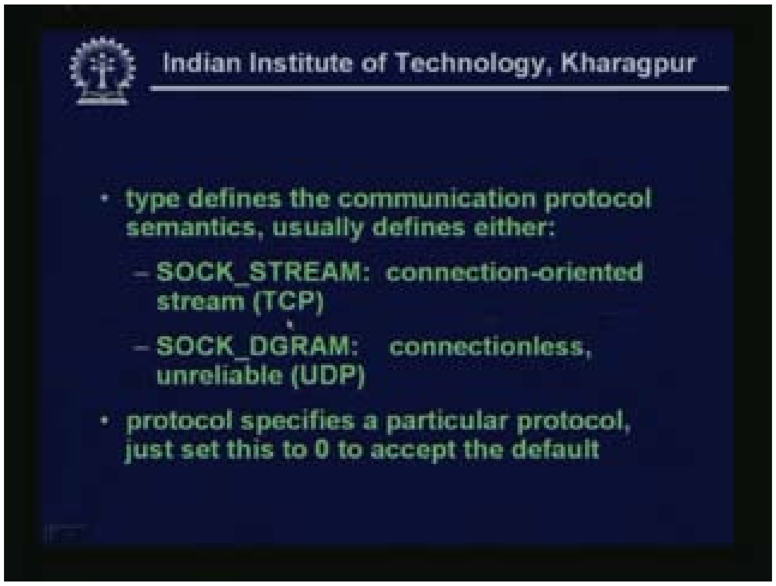
Creating a Socket

```
include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

•domain is one of the *Protocol Families* (PF_INET, PF_UNIX, etc.)

For creating a socket you have to have a library and for accessing the functions in the library which is specifically that socket you have to include that <sys/types.h> and <sys/socket.h> assuming that you are writing in c language. So you may have a function called something like this:
int socket (int domain, int type, int protocol).
When you give this socket call, you have to mention its domain, its type and its protocol.
- Domain is one of the protocol families like PF_INET, PF_UNIX etc so these are the socket domain families we talked about.

(Refer Slide Time: 43:10)

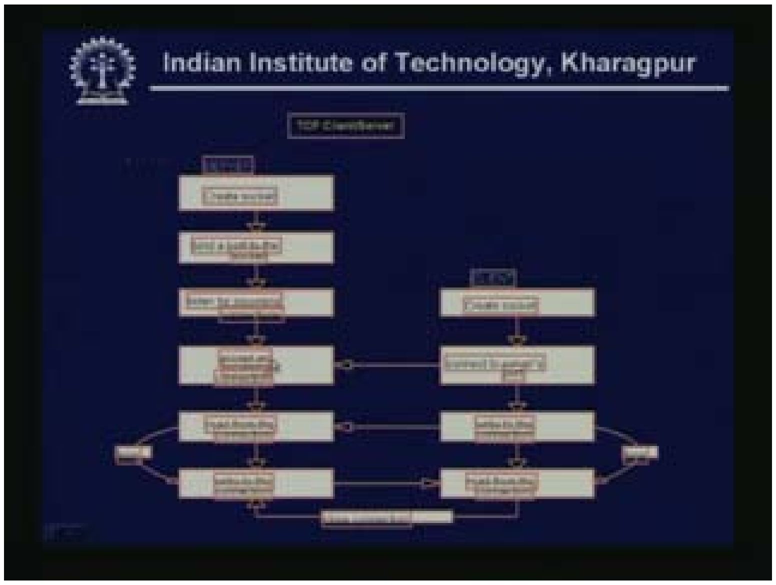


Indian Institute of Technology, Kharagpur

- type defines the communication protocol semantics, usually defines either:
  - SOCK_STREAM: connection-oriented stream (TCP)
  - SOCK_DGRAM: connectionless, unreliable (UDP)
- protocol specifies a particular protocol, just set this to 0 to accept the default

- Type defines the communication protocol semantics. usually it defines either:
  - SOCK_STREAM that means connection oriented stream like TCP or
  - SOCK_DGRAM which is a connectionless unreliable UDP.
- Protocol specifies a particular protocol, just set this to 0 to accept the default.

(Refer Slide Time: 43:53)



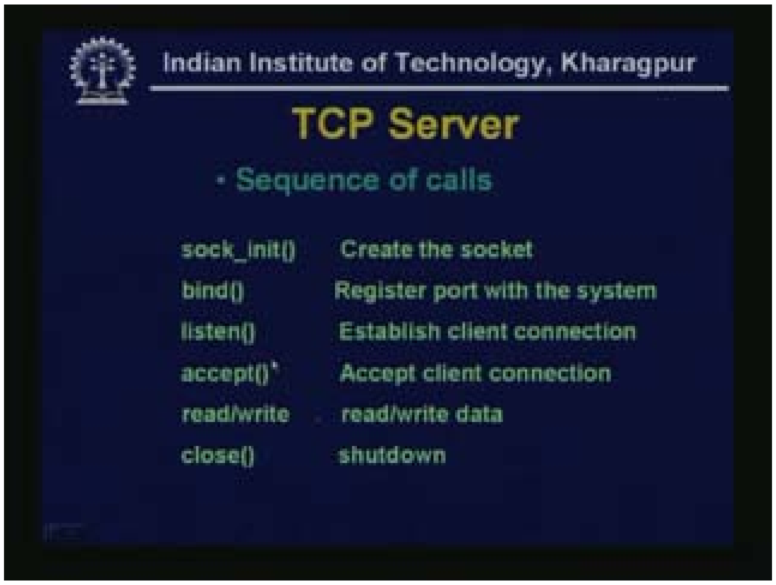
What happens in the client-server is, when the client is initiating a request for a standard kind of a service it knows the destination IP address and the well-known port number And this message has reached the destination machine. And in the destination machine the server program always listens to that particular well-known port number. So the server program is actually in a loop listening whether any request is coming in through that port number. In the case of a UDP server this may be simple. It gets some request, it immediately gives a very short response and goes back to listening.

But in such a case when the interaction with the user may be prolonged, for example, the user makes an FTP request and then downloading the file is going to take quite a bit of time. Similarly, the user may have made an HTTP request which means it may have asked for a particular webpage. Now, sending that webpage is also going to take some time. Of course it is always with respect to the kind of speed people are accustomed to.

If the communication is always through that well-known port then during the currency of this particular session between this particular client and the server that well-known port number is going to be blocked and others will not be able to get the service which is not acceptable. So usually in the client-server there is a handshake. So the request comes to the well-known port number and of course there is a port address which has been sent by the client. From the server side the response goes back with another ephemeral port number which is not a standard port number.

Now, on both sides you have two ephemeral port numbers that means the port number more than 1024 and they can then communicate through these two port numbers with source and destination respectively depending on from which side it is being sent. So these two different port numbers are used. This is happening on one side and on the other side the original server goes back to listening to that same well-known port number ready to service the next incoming request.

(Refer Slide Time: 47:20)


Indian Institute of Technology, Kharagpur

TCP Server

- Sequence of calls

sock_init()	Create the socket
bind()	Register port with the system
listen()	Establish client connection
accept()	Accept client connection
read/write	read/write data
close()	shutdown

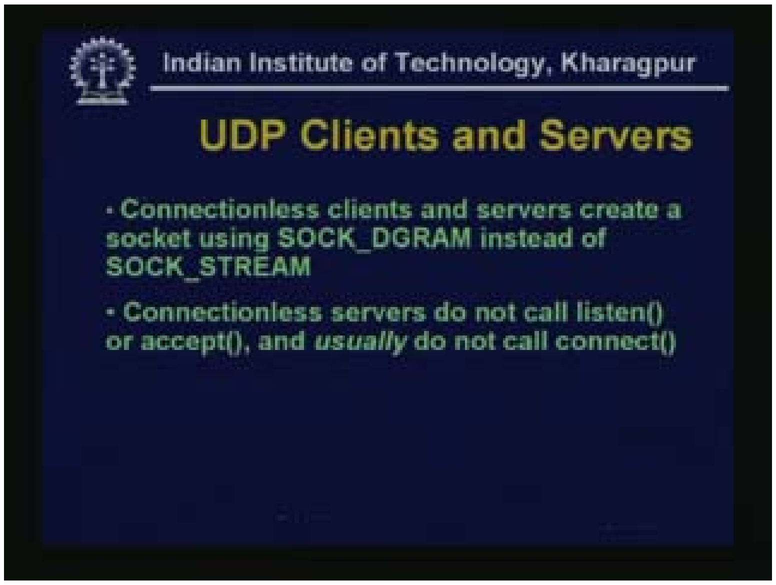
Although we have not discussed TCP as yet but so far as this client-server part is concerned both of them are very similar.

TCP Server:

- Sock_init() - creates the socket
- Register port with the system
- Establish the client connection
- Accept the client connection
- Read/write data
- Close() is for shutting down

When you establish the client connection and when you accept the client connection in this part all these two ephemeral port numbers are exchanged. This is on the server side. On the client side it creates the socket and tries to setup a connection. Usually in a client-server, the initiation of the connection is always from the client side. Then the write/read goes on and then there is a shut down.

(Refer Slide Time: 48:20)



Indian Institute of Technology, Kharagpur

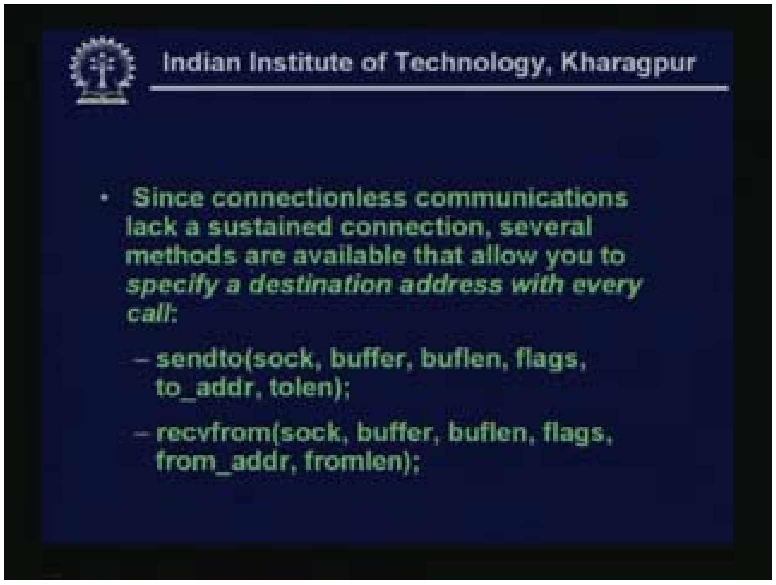
UDP Clients and Servers

- Connectionless clients and servers create a socket using SOCK_DGRAM instead of SOCK_STREAM
- Connectionless servers do not call listen() or accept(), and *usually* do not call connect()

UDP clients and servers are similar except that it uses SOCK_DGRAM instead of SOCK_STREAM.
- Connectionless clients and servers create sockets using SOCK_DGRAM.
- Connectionless servers do not call listen() or accept() and usually do not call connect().

This means you may not require any specific handshake and you may not require a special ephemeral port for prolonged communication between the client and the server because in this case usually the service is to just send only one message and ending the service.

(Refer Slide Time: 49:14)

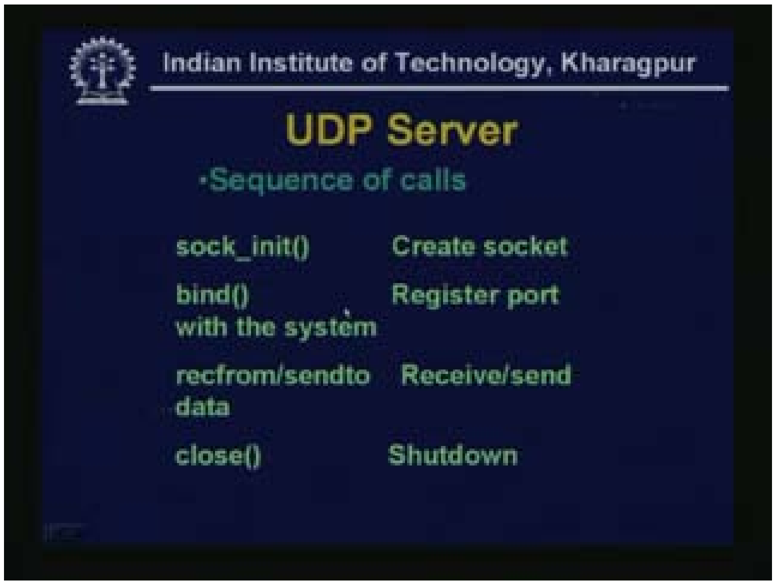


Indian Institute of Technology, Kharagpur

- Since connectionless communications lack a sustained connection, several methods are available that allow you to *specify a destination address with every call*:
  - sendto(sock, buffer, buflen, flags, to_addr, tolen);
  - recvfrom(sock, buffer, buflen, flags, from_addr, fromlen);

- Since connectionless communications lack a sustained connection several methods are available that allow you to specify a destination address with every call:
  - sendto (sock, buffer, buflen, flags, to_addr, tolen);
  - recvfrom (sock, buffer, buflen, flags, from_addr, fromlen);
- There is a way to specify a destination address with every call.

(Refer Slide Time: 49:30)



Indian Institute of Technology, Kharagpur

UDP Server

- Sequence of calls

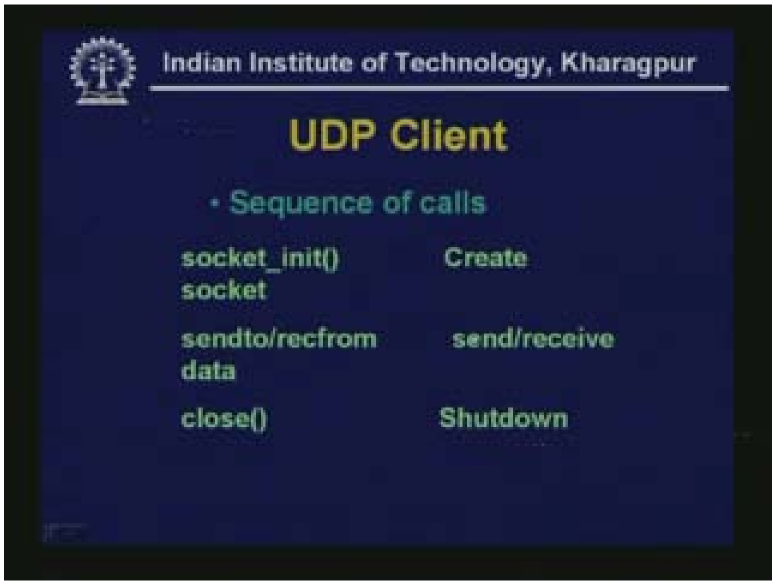
sock_init()	Create socket
bind() with the system	Register port
recfrom/sendto data	Receive/send
close()	Shutdown

For UDP server the sequence of calls is the following:

- Create a socket.
- Register the port using the bind command.
- Receive or send data.
- Shutdown.

This is somewhat simpler than the TCP server. And then there is the UDP client that you create, send or receive and then you shutdown.

(Refer Slide Time: 49:50)



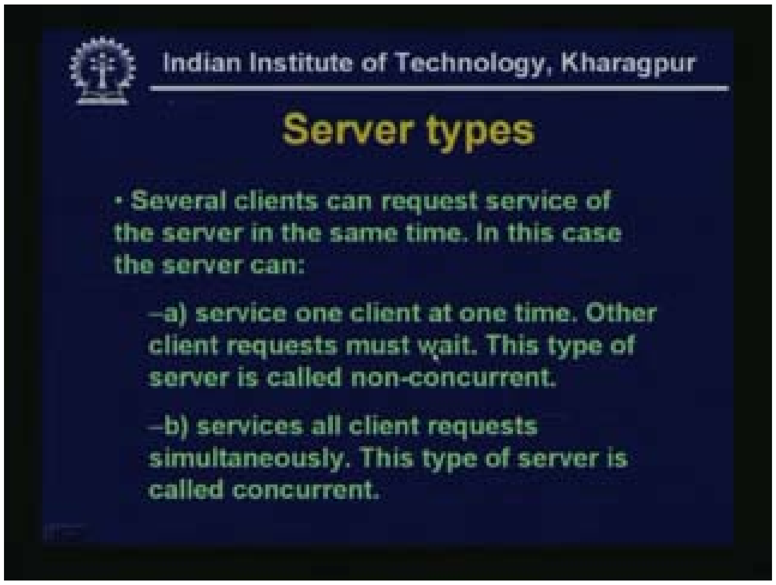
Indian Institute of Technology, Kharagpur

UDP Client

- Sequence of calls

socket_init() socket	Create
sendto/recfrom data	send/receive
close()	Shutdown

(Refer Slide Time: 50:12)



Indian Institute of Technology, Kharagpur

Server types

- Several clients can request service of the server in the same time. In this case the server can:
  - a) service one client at one time. Other client requests must wait. This type of server is called non-concurrent.
  - b) services all client requests simultaneously. This type of server is called concurrent.

There are two types of servers. Several clients can request service of the server in the same time. In this case a server can:

- Service one client at one time. Other client requests must wait. This type of server is called non-concurrent. The second type is,
- Services all client requests are handled simultaneously. This type of server is called concurrent.

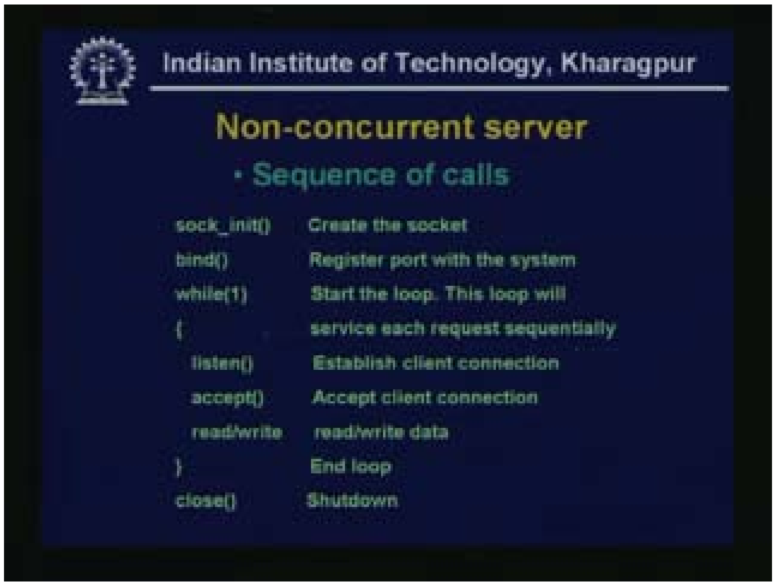
This is what happens, the client's request for a connection has come to the server. By the term server, we mean that software process which is running there and not the hardware box. The hardware is also called a server but in a different sense. In our context by server we mean the process which is giving the service and which is running in a particular machine. If this server is non-concurrent all the user requests come and are sort of put in a queue. Then the server process will take one from the queue, process the service and then send the result and take the next one out of the queue. So there is a queue where all the client requests are waiting and the server process takes one request at a time out of the queue. This is called a non-concurrent server.

When you are using UDP and when using SOCK_DGRAM that means you get a request, send a message and may be that is the end of the service. In that case this non-concurrent server also called as iterative server is more efficient. But it may also happen that, in a particular service the client server communication is for an extended period of time. In this case, one particular request may block all other requests for an unnecessarily long time. In that case concurrent server may be preferred.

In concurrent server, as soon as the server gets the request at the well-known port it immediately spawns a new process. When you execute a fork, in say unique, what you get is an exactly similar piece of code to which you make some changes. What you do is that you give them a new port number and let this new process communicate with this particular client's request and the original process goes back to listening to the well-known port.

Another request may come from some other client somewhere else so it will again spawn another process and the original process will go back listening to the well-known port. All these child server processes use different ephemeral port numbers to communicate with different clients. In the strictest sense if you have only one particular processor in the server machine then only one program can run, things cannot be concurrent but have to be sequential. But then, that sequentiality is imposed by the way processors are scheduled by the scheduler inside the …. So it will give some milliseconds for one process and then it gives few more milliseconds for some other process etc so it will look as if all the clients are getting the service simultaneously. So that is what is meant when we say client requests are serviced simultaneously.

(Refer Slide Time: 54:30)



Indian Institute of Technology, Kharagpur

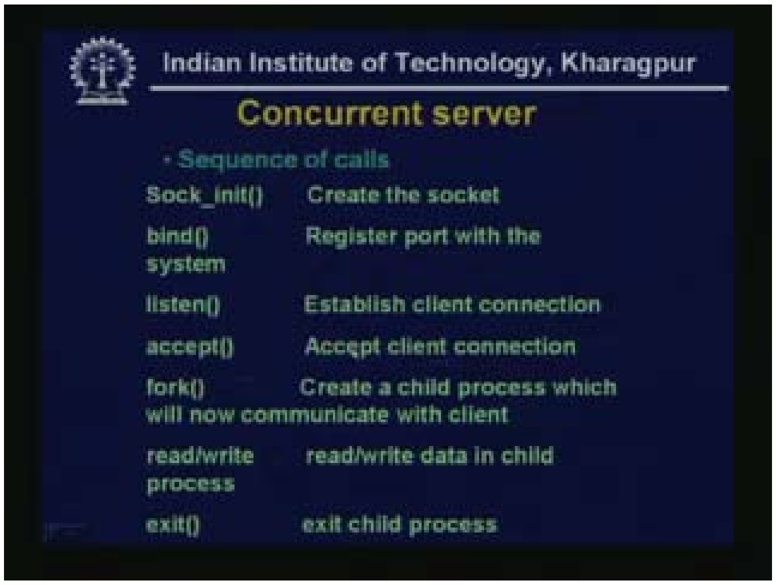
Non-concurrent server

• Sequence of calls

```
sock_init()     Create the socket
bind()          Register port with the system
while(1)        Start the loop. This loop will
{               service each request sequentially
  listen()      Establish client connection
  accept()      Accept client connection
  read/write    read/write data
}               End loop
close()         Shutdown
```

These are the non-concurrent server sequence of calls. The only thing to note is that, there is a while loop and this loop will service each request sequentially.

(Refer Slide Time: 54:45)



Indian Institute of Technology, Kharagpur

Concurrent server

- Sequence of calls

Sock_init()	Create the socket
bind()	Register port with the system
listen()	Establish client connection
accept()	Accept client connection
fork()	Create a child process which will now communicate with client
read/write process	read/write data in child
exit()	exit child process

The concurrent server uses the fork and in the fork it creates a child process which will now communicate with the client. With this we come to the end of this lecture. In the next lecture we will discuss the TCP protocol.