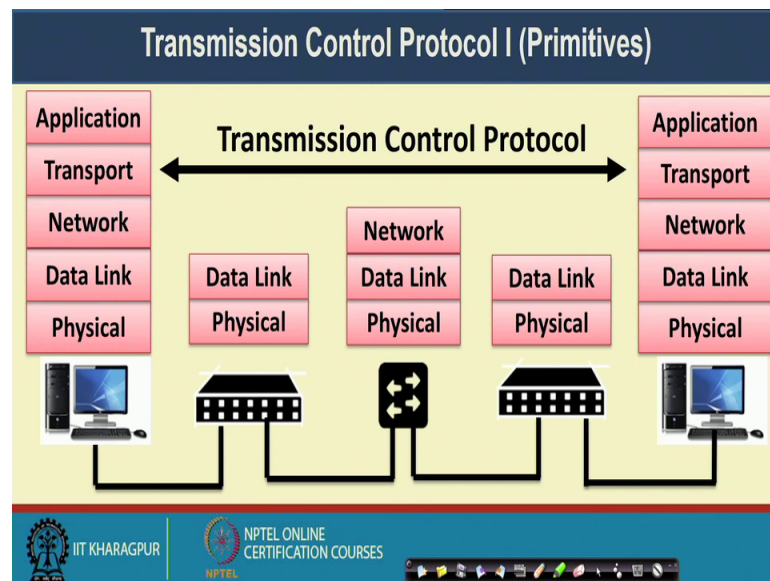


**Computer Networks and Internet Protocol**  
**Prof. Sandip Chakraborty**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 19**  
**Transmission Control Protocol – I (Primitives)**

Welcome back to the course on Computer Network and Internet Protocols.

(Refer Slide Time: 00:20)



So, till now we looked into the details of the transport layer, and what different services is being provided by the transport layer. Now we will take a specific example a transport layer protocol which is widely used in the networks. So, more than 80 percent of the traffic over the internet it uses this transmission control protocol to transfer end to end data. So, we will look in to the details of this Transmission Control Protocol or TCP in stock. And later on we will look into that how you can write an application with the help of such a programming to send or receive data over specific connections. So, let us start our journey to learn the TCP protocol in details.

(Refer Slide Time: 01:03)

**Transmission Control Protocol (TCP)**

- TCP was specifically designed to provide a reliable, end-to-end byte stream over an unreliable **internetwork**.
- **Internetwork** – different parts may have widely different topologies, bandwidths, delays, packet sizes and other parameters

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Well, so this TCP it was specifically design to provide a reliable end to end byte streaming over an unreliable inter network. So, what is meant by unreliable intern network? By the time I hope it is a clear to you that the network layer the IP based network layer that we are considering it is providing unreliable service because of this a buffer filled up are buffer over flow from inter media routers there is a possibility of packet drop. And whenever there is a packet drop from this intermediate routers the network layer does not take care of that.

So, the transport layer, if you want to provide the reliable service on top of the transport layer, that needs up needs to take care of that packet drop. So, TCP is the protocol which supports this reliability on top of this unreliable internetwork. And by internetwork we also look into that different parts of the network may have quietly different topology. So, it may happened that well one part of the network is using wireless technology, one part of the network is using wired technology, another part of the network is using say like that optical communication technology.

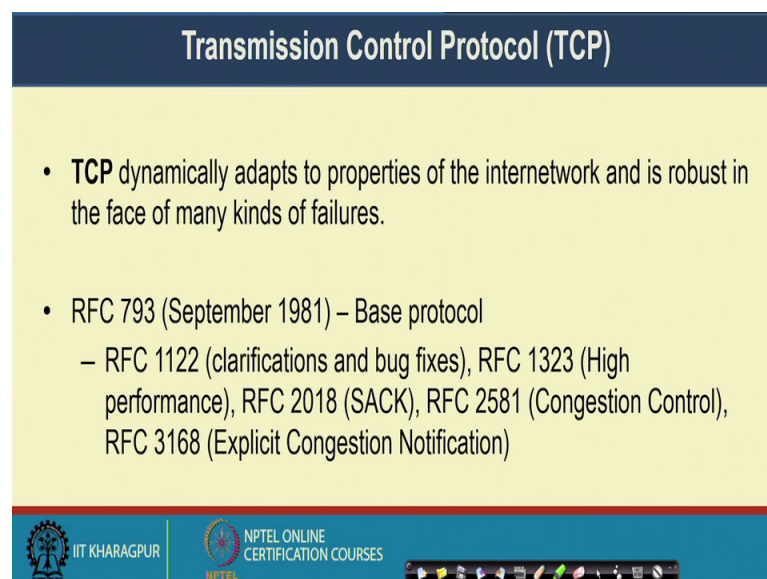
So, there that can be this kind of the various type of technologies which are there in the underlying network, and on top of that I need to transfer the data. So, for example; if you just think of an example when you are doing Facebook on top of your mobile; so, whenever you are doing Facebook on top of your mobile so, the first top is wireless, from mobile, from your mobile to mobile destination that part is wireless.

So, that uses a different set of protocol at the data link layer. Then from that destination to this mobile switching center that part is the wired network. And that uses high speed internet networks. So, there it uses another set of protocol. Now from there from this mobile switching center to the service get way, because say you are just accessing the Facebook server and the Facebook server is somewhere there in say USA.

So, you need to send the data to USA so, the gate way which is connecting this Asian network to the US network that uses optical fiber cable in between. So, you need to transfer the data on top of that top optical fiber cable. So, the underlying network is hugely different to which different properties, and you can have different type of packet loss, delay, re transmission on top of this unreliable network.

Now TCP is a protocol which is designed to handle all this different challenges. So, let us look that how we use TCP to handle this different heterogeneity this different challenges in the networks.

(Refer Slide Time: 03:53)



**Transmission Control Protocol (TCP)**

- **TCP** dynamically adapts to properties of the internetwork and is robust in the face of many kinds of failures.
- RFC 793 (September 1981) – Base protocol
  - RFC 1122 (clarifications and bug fixes), RFC 1323 (High performance), RFC 2018 (SACK), RFC 2581 (Congestion Control), RFC 3168 (Explicit Congestion Notification)

The slide footer contains the IIT Kharagpur logo, the NPTEL Online Certification Courses logo, and a navigation bar with various icons.

Well so, TCP dynamically adapts to the properties of the inter network, and this robust to face many kind of failures which may happen in the network.

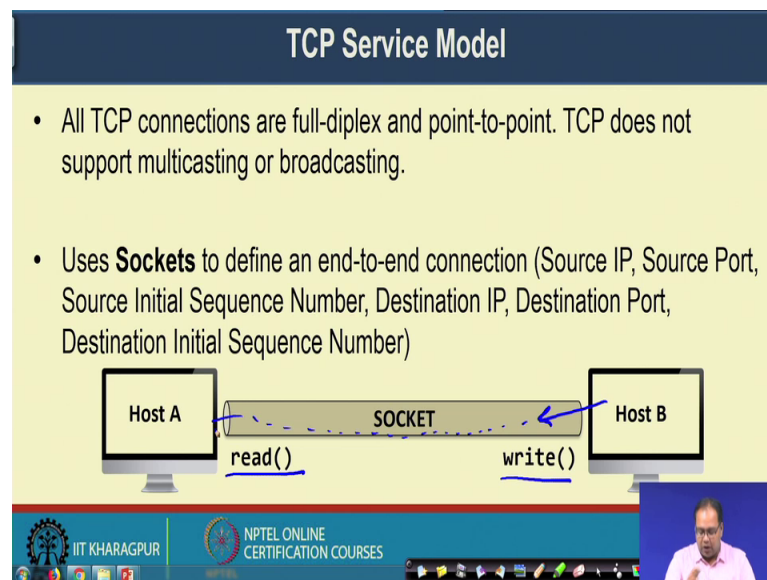
But this TCP protocol has a long history. So, the best TCP protocol it came as a part of this RFC 793. So, this RFCs are some the full form of the RFC is request for comments, and this RFCs are you can think of it as a standard document for a protocol specification,

which is published by IETF Internet Engineering Task Force, which is a global body to handle protocol standardization.

So, the first version of TCP came in September 1981 as a part of this RFC 793, and then it has seen many such changes. So, I have just listed the few changes, this RFC 1122 which does some clarification on the TCP protocol, the sum level of bug says then RFC 1323 which was designed and extension of TCP for high performance, then RFC 2018.

So, the standard TCP protocol, it uses this go back and it go back in ARQ for flow control algorithm. So, RFC 2018 it uses this selective acknowledgement. So, we call it version call TCPs act TCP selective acknowledgement protocol with which uses this selective repeat protocol to handle the flow control mechanism. In RFC 2581 it discusses about the TCP congestion control algorithm, RFC 3168, it uses one concept called explicit congestion notification. Even after that there are multiple amendment over the basic TCP protocol. So, this TCP protocol has changed a lot from what it was designed initially in September 1981.

(Refer Slide Time: 05:40)



So, a broad look on the TCP service model. So, all TCP connections they have full duplex and point to point. So, point to point means they are between 2 end host. And full duplex means both host A and host B whenever you are making a TCP connection between them, host take and send data to host B and at the same time host B will be able to send data to host A.

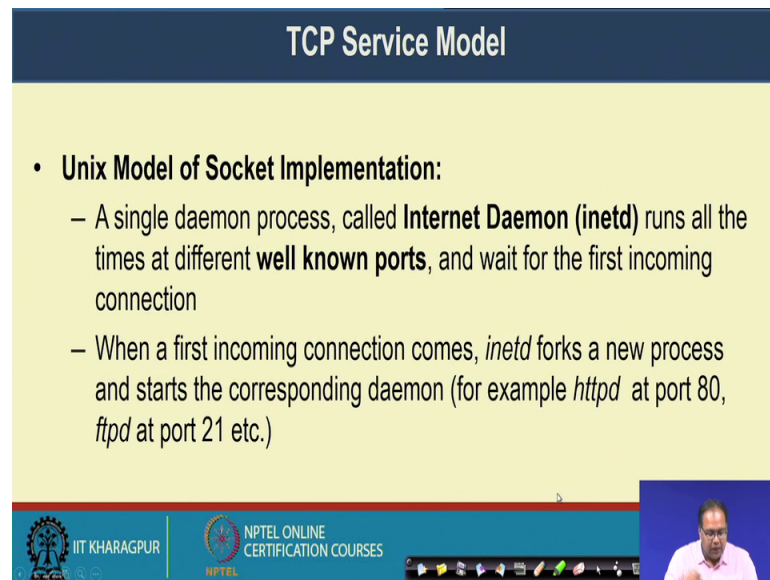
So, TCP it was designed for this point to point data transfer. Data transfer between 2 different machines. It was not designed to support multi casting or broadcasting, when you want to send data from one node to a group of nodes, or from one node to set of large set of nodes. So, for that TCP was not suitable. So, this TCP in a (Refer Time: 06:25) system it uses the concept of socket, which define an end to end connection so the concept of pipe that we are talking about during our discussion of generic service model of the transport layer. The same thing is a term as a socket in the context of the TCP. So, a socket as 6 couples 6 parameters to uniquely identify a socket, the source IP, the source port, the source sequence number, source initial sequence number, the destination IP destination port, and destination initial sequence number; the same thing that was designed to uniquely identify a pipe in a; logical pipe in a transport layer.

Now once this host A and host B has set up a socket among them, then say host A want send some data to host B, sorry, the differs host B want to send some data to host A. So, host B can use this write system call to write the data in this socket. So, host B will be write in the socket, then this data will be delivered to this different layer of the protocol start. Receive at the transport layer of host A, then host A can execute the read call to read the data from the transport layer buffer.

And this delivery the reliable delivery that will be taking care of the by the transport layer and the delivery of the packet to host A based on it is IP address that will be taken care of by the network layer. And that way with all this layers of the protocol stack so this logical pipe or logical socket that defines the service model of a TCP protocol.

So, all the services of this TCP protocol is implemented to support reliable data through this pipe which is termed as the socket.

(Refer Slide Time: 08:17)



The slide is titled "TCP Service Model" in a dark blue header. The main content area is yellow and contains a bulleted list under the heading "Unix Model of Socket Implementation:". The list describes the role of the Internet Daemon (inetd) in listening for connections and forking child processes to handle specific services like httpd and ftpd. The footer of the slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video feed of a presenter.

### TCP Service Model

- **Unix Model of Socket Implementation:**
  - A single daemon process, called **Internet Daemon (inetd)** runs all the times at different **well known ports**, and wait for the first incoming connection
  - When a first incoming connection comes, *inetd* forks a new process and starts the corresponding daemon (for example *httpd* at port 80, *ftpd* at port 21 etc.)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

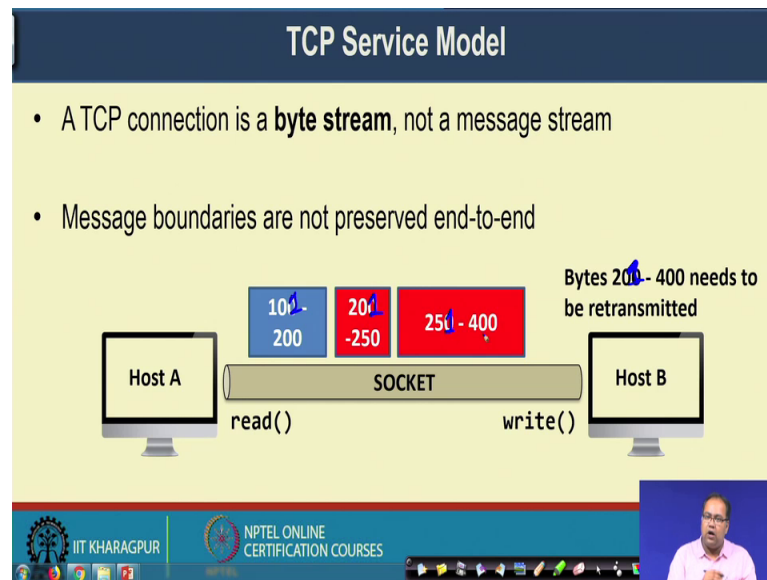
So, in a Unix model Unix base socket implementation we normally run a single daemon process, which is called as a internet daemon or inetd, this inetd it runs all the times at different well known ports. So, it is not like that the all the time you have to open you have to keep one socket open.

So, this inetd takes care of that, the inetd keeps on running on different well known ports, and wait for the fast incoming connection. So, when fast incoming connection comes this inetd it forks; that means, it creates a child process with a new process id and starts the corresponding daemon. So, for example, if you want to do a http file transfer.

So, http file transfer for that you have to run http daemon which at the http server which runs at port 80. So, initially this inetd keeps on listening on port 80 and whenever you try to initiate at connection and port 80, then httpd pops up because http step daemon process, which will use hypertext transfer protocol at port 80, that will have looked into the discussion of application layer protocol. So, it will start that daemon process and create the socket at port 80 at the client, port 80 at the server and some random port at that the client and start receiving the http packet.

Similarly, for ftpd type of the protocol the ftpd will start at port 21.

(Refer Slide Time: 09:44)



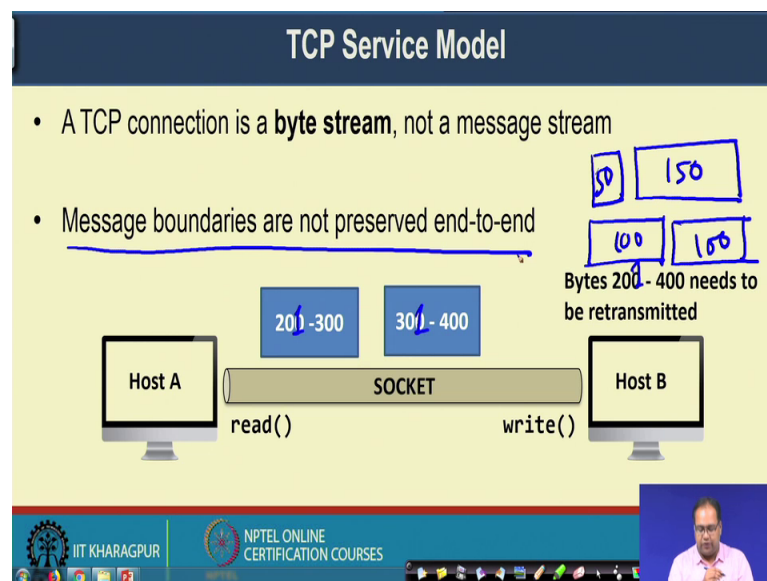
So, few details about TCP; so, first of all TCP connection is a byte stream not a message stream, so, every byte is identified by a unique sequence number, that we discussed during the generic service discussion of a transport layer protocol.

And this message boundary they are not provided preserved end to end; that means, all the messages are in TCP terms we call it as a segment that we have looked in to; the segments may not be of the same size though difference segments may vary. So, here from host B to host A, it may happen that the first segment is starting from sequence number 100, and has a length 100 so, it goes from 100 to 200. The second sequence it goes from this should be 201 it goes from 201 to 250, the third segment it goes from 251 to 400. So, that way this segment contains some 100 bytes of data, this segment contents so, this is say from 101 to 200.

This context content contains 100 bytes of data. This contains 50 bytes of data and this contains 200 bytes of data. So, they have different segments may have different size and the size of the segment will be determined by the flow control algorithm that we will see later on. Now in a hypothetical example, if it happens that well this segment 1 is received correctly by host A and say segment 2 and segment 3 are dropped a lost. So, host B will try to re transmit bytes 201 to bytes 400. So, this should be 201. So, it will try to re transmit from bytes 201 to bytes 400.

So, in this TCP philosophy it is not trying to retransmit 2 segments, rather it will understand that byte 201 to byte 400 has lost. And it needs to retransmit bytes 201 to bytes 400, not the 2 segment. So, these segments may not preserve, because in TCP everything is in the form of a byte stream, and everything is identified by how many bytes I have sent or how many bytes I have received or how many bytes are in transition in the network. So, whenever it is doing the retransmission, may be because of that rate control algorithm which we will discuss in details.

(Refer Slide Time: 12:16)



It may happen that this entire thing is divided into 2 different bytes 2 different segments. So, the first segment contains bytes 201 to 300.

And the second segment contains bytes 301 to 400. So now, you can see that the earlier division that we had from bytes to segment that was not being preserved here. So, earlier I have a small segment of 50 bytes and another large segment of 150 bytes. But now whenever I am doing the retransmission, I found out or better to say that TCP at host B finds out that well.

Now, I do not need to send a small segment of 50 byte, rather I can retransmit this entire byte with 2 segments of 100 bytes each. So, that is why we use this term that the message boundaries are not preserved end to end in the context of a TCP protocol. So, everything is byte stream.

(Refer Slide Time: 13:24)

### TCP Service Model

**Example:**

- The sending process does four 512 byte writes to a TCP stream – for `write()` call to the TCP socket
- These data may be delivered as – four 512 byte chunks, two 1024 byte chunks, one 2048 byte chunk or some other way
- There is no way for the receiver to detect the unit(s) in which the data were written by the sending process.

The diagram shows a buffer divided into segments. An application (APP) writes 512 bytes to the buffer. The buffer is then divided into segments. The application then reads 1024 bytes from the buffer. The buffer is labeled 'segment'.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

One example that the sending process it does four 512 byte writes to a TCP stream using the write call to the TCP socket. So, the application is sending four 512 bytes of blocks to the transport layer, and if you remember that the transport layer architecture, you have this application whenever the application is making a write call, that data is going to a buffer.

So, the data is going to a buffer, and the transport layer NPT is reading the data from this buffer and creating the segments. Now when it is creating the segments, if the sending process writes four 512 bytes block to this buffer, now this data may be delivered as four 512 bytes chunks that mean four 512 bytes segment 2048 bytes segments or one for 2400 byte segments or in some other way it is not necessary that all the segments need to be a of same size.

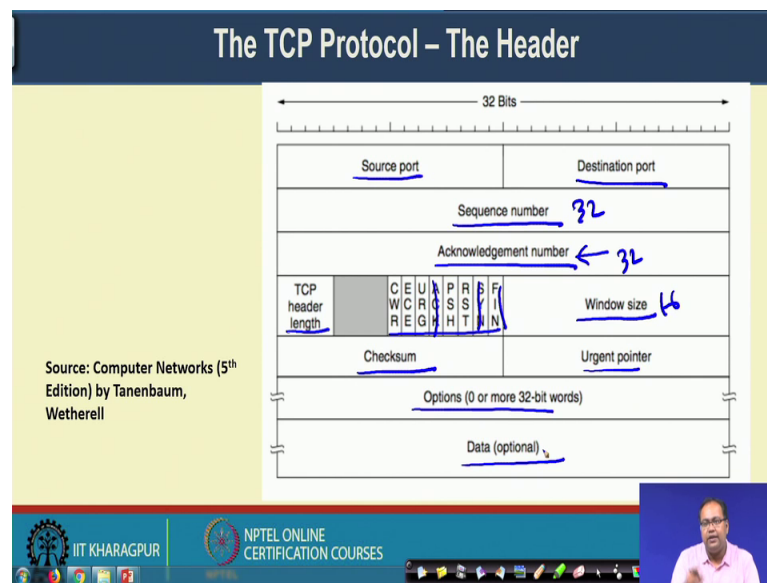
So, there is no way for the receiver when the receiver will receive that data, to detect the units in which the data were written by the sending process. So, at the sending process as written data in 512 byte chunks, but whenever the receiver process will receive the data, that is the opposite team, you get the data put it in a buffer receiver buffer, then the receiver application will make a read call to read the data.

From the buffer and when the receiver application will made the read call to read the data from the buffer it will again get started number of bytes. And during that time it may

happened that the read, read, read this application is making a read call at 1024 bytes chunks.

So, the sender has read any that 512 bytes chunks and the receiver is receiving in that 1024 bytes chunks. So, that may widely differ and even the receiver does not know that at which or a what was the chunk size when the sender has written that to the transmit process to the transmission control protocol to the TCP process.

(Refer Slide Time: 15:46)



So, this is the header structure of the TCP protocol. So, the well known fields are already there, that you have looked into the source port and the destination port, to uniquely identify the application through which you are making a communication.

You have this sequence number to uniquely identify each packet you have an acknowledgement number to acknowledging the bytes that you have received. I made a mistake, I will taking I have told that sequence number for packet rather than that sequence number for the byte because you are using byte sequence number. So, you should use the correct ampler.

Then the header length the length of header certain flags so, these are the flag bits. So, we will look into the flag bits in a detail. Just for the few flag bits like this field flag FIN bit like this FIN bit is used to close connection to finish a connection. So, if this FIN bit is set; that means, it is a connection closer message if this SYN bit is set SYN bit is for

connection initialization. So, if this SYN bit is set; that means, it is a SYN message for connection initiation. If this ACK bit is set; that means, it is an acknowledgement message which is sending this acknowledgement number about the up to which bytes have been acknowledged by the receiver.

Then you have this window size. This window size is the receiver advertised window size for sliding window protocol for dynamic buffer management. So, with this window size the receiver is announcing that what is the available buffer space in the receiver side; this is 16-bit window size.

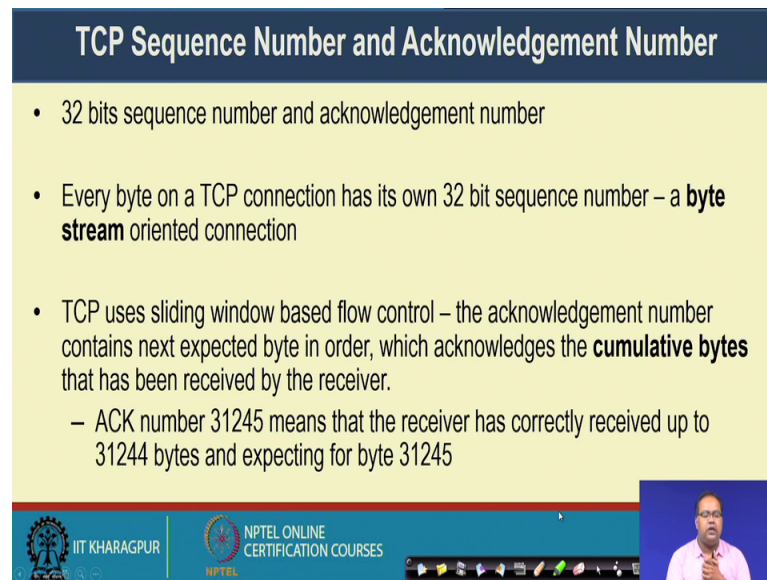
So, we have 32-bit sequence number 32-bit acknowledgement number 16 bit windows size. Certain checksum to check that the correctness of the received data, urgent pointer we will discuss about this urgent pointer later on in brief like if you want to send some message urgently by bypassing the queue, because if you look into the transmit queue, trans or that transport layer queue.

That transport layer queue is FIFO thing First In First Out queue. So, whatever byte has been came first, it will send that byte first. So, if you said the urgent pointer, the urgent pointer says that well if you are sending some data from the application by setting the urgent pointer; that means, whatever data you are sending from the application layer by setting the urgent pointer, you can do that with the help of such a programming we will look into that.

If you do that then it will first create that segment and segment then send it out with the urgent bit set to one it, indicates that this particular data is urgent that should be should not wait inside the queue for this first come first out or first in first out behavior.

Then you have some optional fields. And finally, the data which is coming from the upper layer; that means, the pay load for this packet.

(Refer Slide Time: 18:37)



**TCP Sequence Number and Acknowledgement Number**

- 32 bits sequence number and acknowledgement number
- Every byte on a TCP connection has its own 32 bit sequence number – a **byte stream** oriented connection
- TCP uses sliding window based flow control – the acknowledgement number contains next expected byte in order, which acknowledges the **cumulative bytes** that has been received by the receiver.
  - ACK number 31245 means that the receiver has correctly received up to 31244 bytes and expecting for byte 31245

The slide footer includes the IIT KHARAGPUR logo, NPTEL ONLINE CERTIFICATION COURSES text, and a small video inset of a speaker.

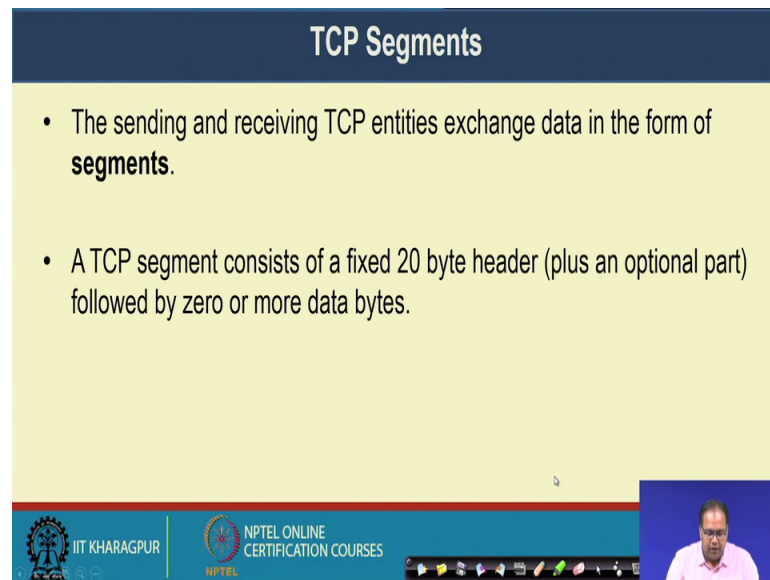
Well we have looked into that is TCP sequence number and acknowledgement number it uses 32 bits sequence number and 32 bit acknowledgement number.

So, every byte on a TCP connection has its own 32 bit sequence number. So, because it is a byte stream oriented protocol that you have seen. So, TCP it uses sliding window based flow control. So, the acknowledgement number, it contains the next expected byte in order which acknowledges the cumulative bytes that have been received by the receiver.

So, the example is like that. If you receive an acknowledgement number 31245; that means, that the receiver has correctly received all the bytes up to 31244 and it is expecting the byte 31245. So, that way it is the cumulative acknowledgement number. So, once you are getting an acknowledgement number, it means that all the bytes before that number immediately before that number, that have been received correctly by the receiver and it is expecting that particular byte.

So, it is expecting byte 31245 and it has received everything correctly up to byte 31244.

(Refer Slide Time: 19:42)



### TCP Segments

- The sending and receiving TCP entities exchange data in the form of **segments**.
- A TCP segment consists of a fixed 20 byte header (plus an optional part) followed by zero or more data bytes.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, we looked in to this earlier. That in TCP this message boundary we call it has a segment. So, the sending and the receiving TCP entities they exchange the data in the form of segment. In general, a segment consists of a fixed 20-byte header plus an optional part. So, the header format that we have looked earlier the TCP is segment header followed by 0 or more data bytes. So, if it is a connection message or connection closer message like the SYN message or the FIN message.

You do not have any data, but if it is a data message you may have additional data which is along with that segment.

(Refer Slide Time: 20:22)

The slide is titled "TCP Segments" in a dark blue header. The main content area is yellow and contains two bullet points:

- TCP can accumulate data from several `write()` calls into one segment, or split data from one `write()` into multiple segments
- A segment size is restricted by two parameters
  - IP Payload (65515 bytes)
  - Maximum Transmission Unit (MTU) of the link

To the right of the text is a hand-drawn diagram in blue ink showing a network path. It starts with a laptop icon, goes through a node labeled "WiFi", then a node labeled "Ethernet", and ends at a server icon. There are some additional scribbles and a small "0F" at the end of the path.

The footer of the slide is dark blue and contains the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a man speaking.

Now, how let us see how this TCP segments are being formed and as you have seen earlier, that it is not necessary that all the segments will be of equal size. Here it will be little clear to you that why all the segments are not of equal size in TCP. So, TCP it can accumulate data from several write calls in to one segment. Or split data from one write call into multiple segments.

So, this write call with this write system call your sending a chunk of data from the application to the transport layer. Now whenever the TCP is running TCP say even if you have send some just think of you have sent 1024 byte data you are sending 1024 byte data as a single chunk from application layer to the transport layer with the help of this write system call.

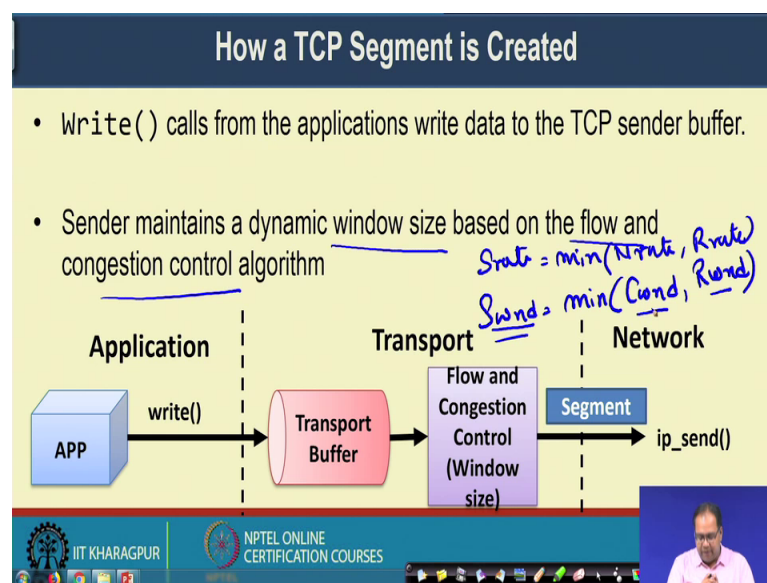
It may happened that, the TCP may break that 1024 chunk in to 2 512 byte chunk, and send 2 segments based on the need the need, I will discuss a couple of minute later. Or it can combine thousand to 1024 byte chunk together, and create a single segment of 2048 byte and send it to one group.

Now, how this segments are been created? That the segment size it is restricted by 2 parameters. The first parameter is the IP payload; the amount of data that you can put inside the IP fragment, whenever it is going to the network layer. That is restricted to 65515 bytes. So, your segment size cannot be more than that. The second parameter is the maximum transmission unit of the link.

So, what is maximum transmission unit? That means, whenever you are considering and net multiple network link from say source to destination, this links have a maximum transmission unit. So, that comes from the concept of data link layer, how this maximum transmission unit from data link layer comes in to practice. We will discuss that in details when we discuss about the data link layer.

But for the time link just take it as an example, or take it as an given postulate that for different technology the maximum transmission unit is different. So, for example, if this link is the Wi-Fi link, you have one in queue, if this is an Ethernet link or a WAN link you will have another MTU. If this is an optical fiber link, you will have another MTU Maximum Transmission Unit. So, the maximum transmission unit is basically that at a single go what is the amount of data or what is bit should be the amount of the data that you can put inside the packet.

(Refer Slide Time: 23:10)



So, what TCP does? TCP get uses this write calls from the application to write the data in the TCP sender buffer. So, here in this example that the application makes a right call to write data in the transport buffer, and the sender it maintenance a dynamic window based on the flow control and the congestion control algorithm.

So, ideally your sending rate was minimum of network rate and receiver advertise rate. So, whenever we convert this window form, your sender window size will be minimum of one window size which will be given by the congestion control protocol. So, earlier

we are talking about that in case of congestion control, you increase the network rate from low rate from very high rate using this additive in this principle.

So, to increase that rate, it is just like your increasing the window. So, size if you are increasing the window size; that means, at the same instance of time you will be able send more data. So, you will be able to increase the rate. So, this congestion window keeps a indication that well if your window size is 1 so, you can send 1 byte of data if your window size is 2, you can send 2 bytes of data simultaneously, if your window size is 4, you can send 4 bytes of data simultaneously.

So, you have this congestion window and the receiver advertised window size minimum of that. So, you have this sender window size. So, this sender window is dynamically triggered, dynamically updated based on the receiver advertise window size and the congestion window size, that you are gradually increasing in the additive in this space, and whenever a congestion is deducted you are dropping it again to up small value or a minimum value.

So, this flow and congestion control algorithm, it will use this window size and based on that your segment will be created.

(Refer Slide Time: 25:07)

**How a TCP Segment is Created**

- Modern implementations of TCP uses path MTU discovery to determine the MTU of the end-to-end path (uses ICMP protocol), and sets up the Maximum Segment Size (MSS) during connection establishment
  - May depend on other parameters (buffer implementation).
- Check the sender window after receiving an ACK. If the window size is less than MSS, construct a single segment; otherwise construct multiple segments, each equals to the MSS

*Handwritten notes:* Round = 2048 B, MSS = 1024 B

*Diagram:* A sequence of segments: 512B, 1KB, 1KB, 2KB, followed by a destination 'D'.

**Footer:** IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES

So, this is the algorithm for creating a segment. So, today's implementation of TCP, it uses this path MTU discovery a protocol which is there in the internet control message

protocol path, as a path of the internet control message protocol which is implemented at the network layer.

So, what it does? Is it tries to estimates that what is the MTU of all the links in the path. So, by getting the information about all the MTUs of the path, of the link in the path so, as an example if it happens that well, so, these are the links. So, this is your source and this is your destination. So, this link support 512 byte this supports 1KB, this supports 1KB and this supports say 256 byte. So, if that is the case; that means, ideally you should not send data more than 256 byte in this entire end to end path.

So, that is the task which is done by this path MTU discovery mechanism inside the ICMP protocol Internet Control Messaging Protocol. And it sets up it is maximum segment size during the connection establishment. So, during the connection establishment by exchanging this message at the network layer by getting this feedback from the network layer it is sets up the maximum segment size.

This maximum segment size sometime depends on a other parameters, like the buffer implementation of what is the amount of the data that your buffer can hold at one go. Now the sender it checks the window after receiving an act because whenever you are receiving an act with that you have this currently receiver advertised window size, which will tell you that what how much of the data the receiver can hold.

So, if the window size is less than MSS; that means, the receiver can whatever receiver can hold it is less than your maximum segment size. So, you construct a single segment, otherwise if that is the case a if your receiver window size is.

More than your MSS, say your receiver window size is 2048 byte and that is your receiver window size and your MSS is 1024 byte; that means, you can if you have 2048 byte of data in your inside your sender buffer, then you can create 2 difference segments with 1024 bytes.

So, that way you create it 2024 bytes segment and transfer it over the network. So, that way dynamically this segment size are get adapted based on this particular mechanism by looking into that what is the receiver advertised window, what is your maximum sender size, as well as what is the amount of data which is there in the sender buffer. It may happened that the sender buffer have only 10 byte of data, if the sender buffer has

10 byte of data, then if you just keep on waiting for whenever you will get 1024 byte, because it is equal to your maximum segment size and you will transmit that, you may unnecessary delay certain message.

So, that is why we will look them in mechanism in details later on, but broadly even if you have a small amount of data in the sender buffer, you do not wait for the data for the maximum segment size, whatever data is there in the segment size you transfer that data.

So, that is why many of the times it may happen that if the application is not generating sufficient data, then you can push the data in the network which is less than the where the segment size is less than your maximum segment size.

(Refer Slide Time: 28:56)

### Challenges in TCP Design

- Segments are constructed dynamically, so retransmissions do not guarantee the retransmission of the same data segment – a retransmission may contain additional data or less data
- Segments may arrive out-of-order. TCP receiver should handle out-of-order segments in a proper way, so that data wastage is minimized.

The diagram illustrates the challenges in TCP design. It shows a sender (S) and a receiver (R). The sender has a buffer with segments of 50, 100, and 150 bytes. A retransmission of 100 bytes is shown. The receiver has a buffer with segments of 100, 150, and 50 bytes. A retransmission of 100 bytes is shown. The diagram highlights that retransmissions may contain additional data or less data, and segments may arrive out-of-order.

So, the challenges which is there in the TCP design, and from here we will look in to the design details design in details. First of all this segments are constructed dynamically. So, re transmission that do not guarantee that the re transmission of the same segment.

So, that we have seen earlier, that the earlier you had one segment of 50 bytes another segment of 150 bytes and whenever you are making re transmission you are making to re transmission of 100 bytes each. So, re transmission may content additional data or less data or re arrangement of the segments. And sometime this segments may be out of order. Because TCP it does not determine the path, the network layer is determining the path. And the network layer it may happen that for one segment, one packet which is

coming from the application the network layer packet, it decides one path for another packet it decides another path. Because of this load balancing or many other mechanism in the routing protocol. And because of that this segments, you may receive the segments out of order a TCP. So, this TCP receiver it should handle the out of order the segment in a proper way so, that data wastage is minimized.

So, if you are applying this go back in ARQ and if you just think of that well, if I am receiving something out of order I will not put it in to the buffer because anyway the sender will re transmit the entire thing all together; that is not a wise idea. Why? Because small example, because it may happened that at the this is a receiver this is a sender, the receiver side say you have received this much of data and then you have received this much data.

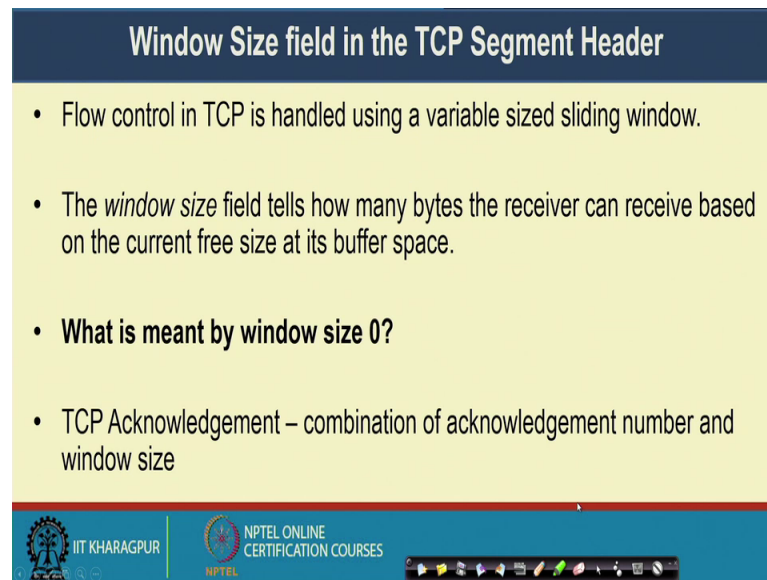
Ok, so, you have received from say 100 to 120, then 121 to 150, you have not received then from 151 to some 500 byte you have received. Now at this stage whenever the sender gets a time out, the sender will try to re transmit this byte along with all the other bytes which are there in the sender buffer. So, this is the sender buffer we will send that re transmit that data.

Whenever it is re transmitting that data, assume that in the first segment it has re transmitted from 121 to 160 in a single segment so, the moment you are getting this the receiver will get this, receiver will can put receiver will just chop out this byte from 120 to 149 and put the data here in between; so, this entire data now from 120 to 500 that has been received.

So, the receiver can send an acknowledgement, and that is the cumulative acknowledgement saying that it has received up to byte 500. And the moment sender gets it is the sender as stops may stop sending this re transmission of the additional bytes. And it can update it is window parameter accordingly and start sending the data from 501 bytes to the remover.

So, that way you can do the optimization to have better utilization of the network resources.

(Refer Slide Time: 32:10)



**Window Size field in the TCP Segment Header**

- Flow control in TCP is handled using a variable sized sliding window.
- The *window size* field tells how many bytes the receiver can receive based on the current free size at its buffer space.
- **What is meant by window size 0?**
- TCP Acknowledgement – combination of acknowledgement number and window size

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, we have this window size field in the TCP segment header. So, this window size field is used for flow control algorithm in TCP. It uses variable size sliding window protocol the dynamic buffering that we have looked in to earlier. So, this window size field it tells that how many bytes the receiver can receive, based on the current free size at it is buffer space. And as we have seen earlier that a window size 0 means, the receiver does not have a sufficient buffer space.

So, the sender should stall transmitting further data until it gets a good amount or sufficient or sufficient amount of window size advertisement. Now TCP acknowledgement; so, the final TCP acknowledgement it is a combination of the acknowledgement number and the advertised window size based on that the sender will tune it is parameter. So, that is the basic things about the TCP protocol.

So, in the next class, we will go to the details of this sliding window based flow control algorithm which is adopted as the part of the TCP.

Thank you all for attending this class.