

**Computer Networks**  
**Prof. S. Ghosh**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture -31**  
**TCP**

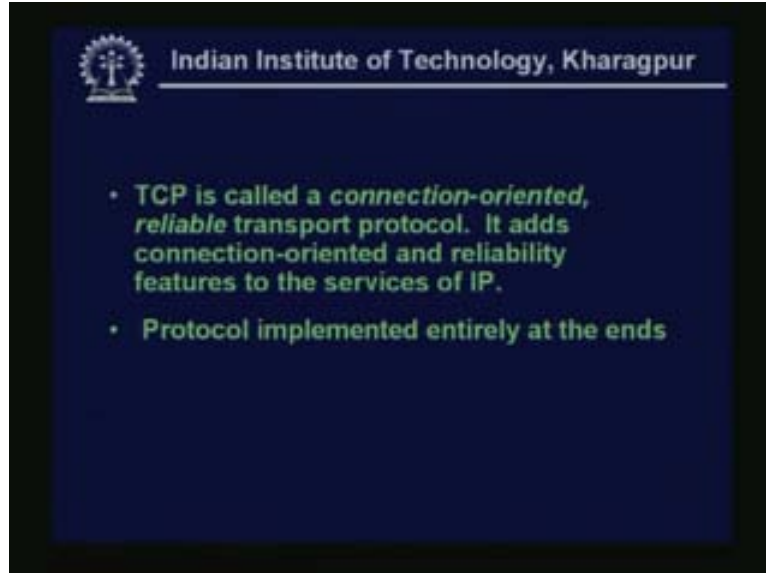
Good day, our topic today is TCP. TCP is the second most important transport protocol. TCP is very widely used by many applications. We have already discussed UDP. Actually this is a little more complex than UDP but it also has some advantages. We will see what they are. The Transport layer responsibilities are:

(Refer Slide Time 01:13 - 01:44)



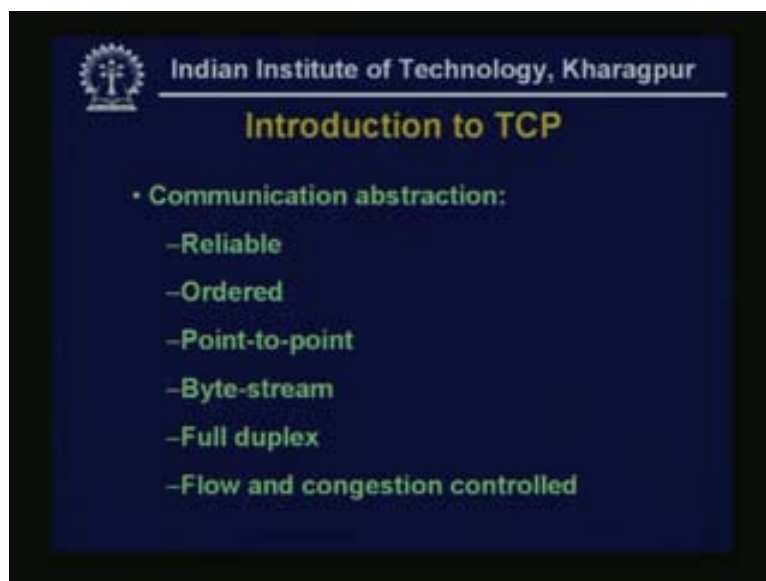
Transport layer creates packets from byte stream received from the application layer. And in order to multiplex and de-multiplex amongst various applications it uses port numbers to create process-to-process communication. It uses a sliding window protocol to achieve flow control and uses acknowledgement packets, time out and retransmission to achieve error control. So, unlike UDP which is unreliable this seeks to provide a reliable communication. That means it is error free, it is a connection oriented protocol and it also has some kind of congestion control mechanism. And it does the basic thing of connecting between processors amongst two distant nodes possible.

(Refer Slide Time: 02:54-03:02)



TCP is called connection-oriented reliable transport protocol. It adds connection oriented and reliability features to the services of IP. IP as such is best effort kind of service so it does not give you any reliability. Secondly IP is connectionless whereas TCP tries to give some kind of connection-oriented flavor to the connection. And proposal is implemented entirely at the two end nodes. That means the intermediate routers etc do not have any role to play.

(Refer Slide Time: 03:35-04:07)



The communication abstraction is that, it is reliable and ordered. Ordered means the packets are received in order. In IP it is not guaranteed that the packets will arrive in

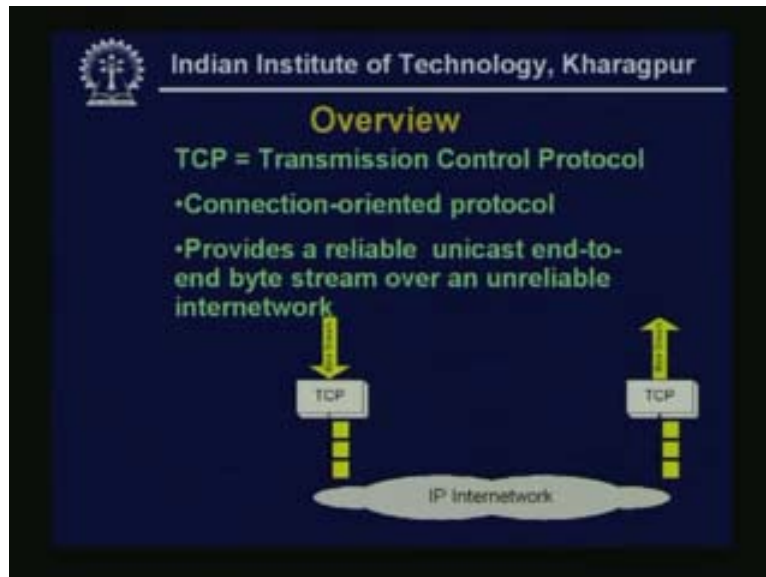
order because what might happen is that one packet may be routed in one direction and the other packet may be routed in other direction so the packet which was sent first may land up at the destination earlier and vice versa. But TCP makes them ordered. TCP brings an order amongst them.

(Refer Slide Time: 04:08 – 04:22)



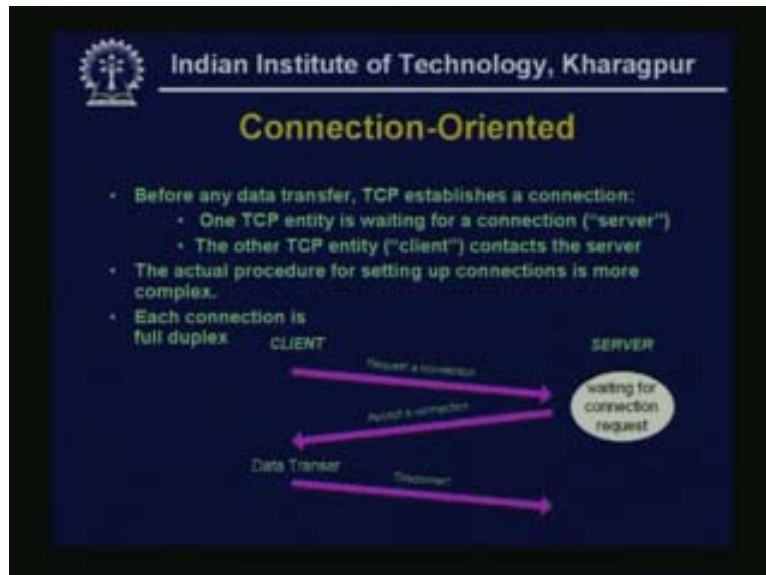
TCP is point-to-point and unicast. All these features namely it is ordered, point-to-point, reliable and unicast is what gives the connection-oriented flavor to TCP. It takes the byte-stream as an input and gives that as the output. It is a full loop-less connection. That means both the nodes A and B are connected by TCP then A can communicate to B and B can communicate to A and it has flow and congestion control.

(Refer Slide Time: 04:22-04:32)



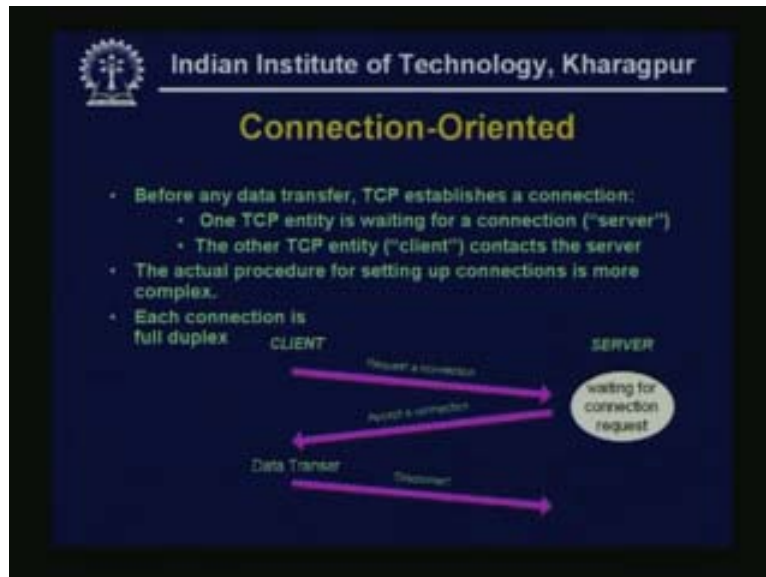
TCP (Transmission Control Protocol) is a connection-oriented protocol, reliable, unicast, end-to-end, byte-stream over an unreliable network.

(Refer Slide Time: 04:58-05:41)



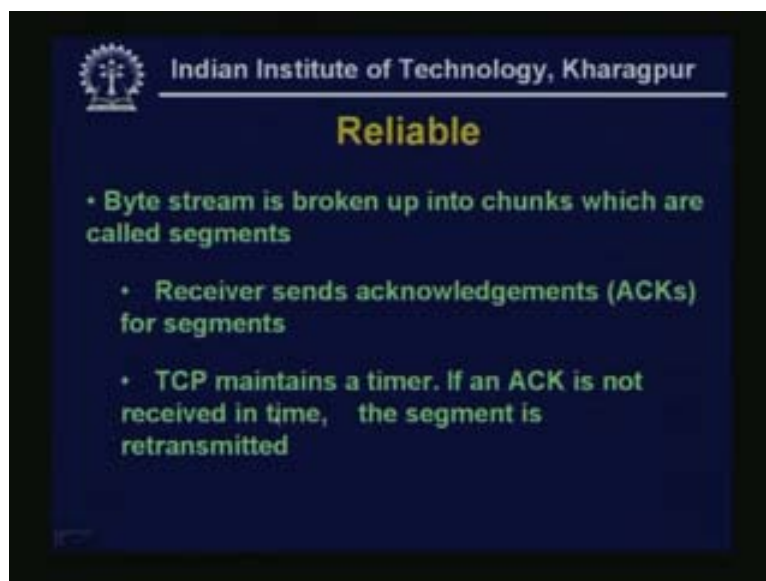
So, before any data transfer TCP establishes a connection. Just like in a connection-oriented network like classical telephones, first there is a connection which is set up by all kinds of control signals. Similarly, like in ATM, first a connection is set up and then the actual data transmission begins. Similarly, in TCP a connection is set up between the two end points before any transmission takes place, so it sets up a connection.

(Refer Slide Time: 05:42 – 06:09)



One TCP entity is waiting for a connection that is the server. The other TCP entity-client contacts the server. The actual procedure for setting up connection is more complex. The client first makes a request for a connection and then the server acknowledges the request and accepts it, then the data transfer begins. So there is a connection set up before actual data transfer can take place and at the end of the data transfer there is a disconnect phase also.

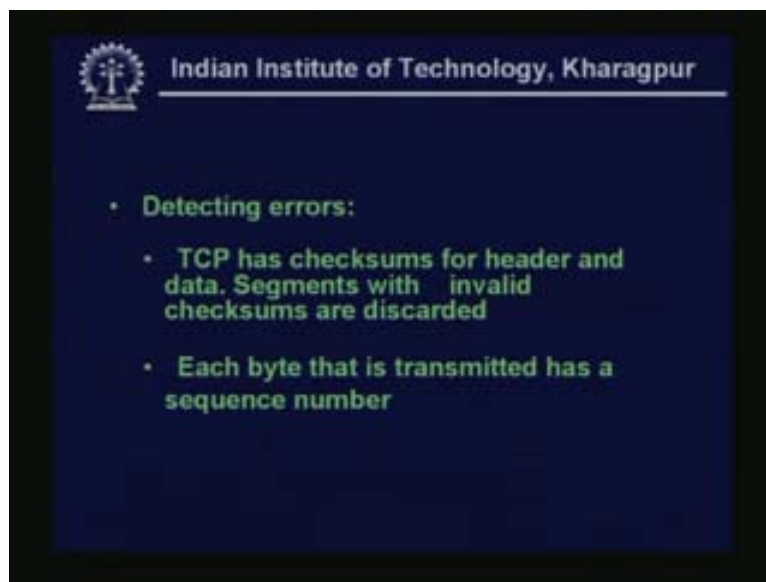
(Refer Slide Time: 07:08 – 07:34)



It is reliable. The byte-stream is broken up into chunks which are called segments. Therefore at the TCP level they are called segments. The receiver sends

acknowledgements for segments so this is the basic mechanism by which reliability is brought in that each and every segment is acknowledged. TCP maintains a timer. If an ACK is not received in time the segment is retransmitted. So this is the basic mechanism for reliability. The sender sends a packet and it waits for some acknowledgement. And if the acknowledgment does not come through from receiver then the sender assumes that the packet has been lost and sends the packet again hoping that this packet will eventually reach the receiver and the receiver will send the acknowledgement. This might also lead to duplicate packets at the destination. For example, the sender has sent the packet and the receiver has received it and it has sent an acknowledgement but then the acknowledgement got lost. So naturally the original sender did not receive any acknowledgement and after sometime it will send it again so a duplicate segment would be received by the receiver but then it knows that it is a duplicate and will eliminate that. So, by this way it achieves reliability.

(Refer Slide Time: 07:34 – 07:59)



TCP can detect errors. TCP has checksum for header and data like UDP. Segment with invalid checksums are discarded. Each byte that is transmitted also has a sequence number. So, if some intermediate sequence number is missing the receiver knows that something has been lost.

(Refer Slide Time: 07:59 – 08:32)

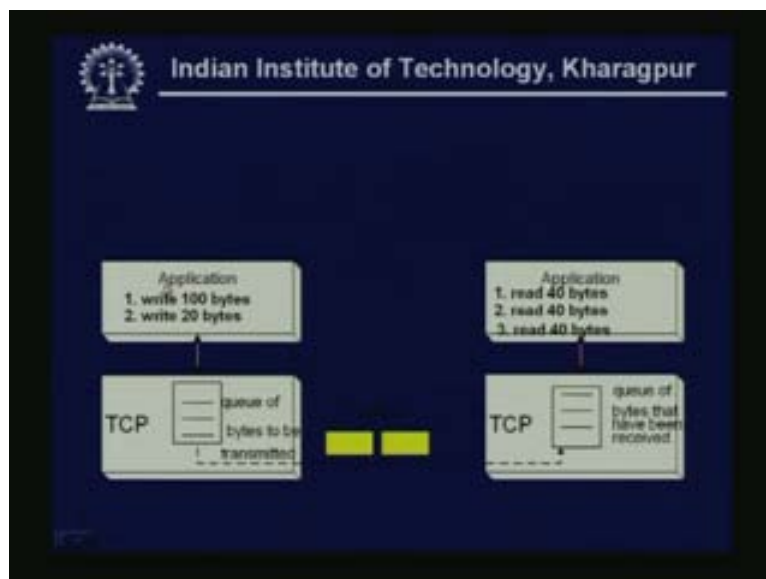
Indian Institute of Technology, Kharagpur

### Byte Stream Service

- To the lower layers, TCP handles data in blocks, the segments.
- To the higher layers TCP handles data as a sequence of bytes and does not identify boundaries between bytes
- So: Higher layers do not know about the beginning and end of segments

To the lower layers TCP handles data in blocks, the segments. This is where the packets actually originate. But to the higher layer TCP handles data as a sequence of bytes and does not identify boundaries between bytes. So, the higher layers do not know about the beginning and end of segments. Hence to the higher layer it is just a stream of bytes.

(Refer Slide Time: 08:33 – 09:02)




For example, the application writes 100 bytes then it writes 20 bytes so all these go into the queue. So the queues are bytes to be transmitted and then TCP transmits them. At the other end also there is a queue of bytes and at the other end also it reads 40 bytes at each



go. So, to this application this is the byte-stream and for this application also this is just a stream of bytes coming in such units.

(Refer Slide Time: 09:03 – 09:37)



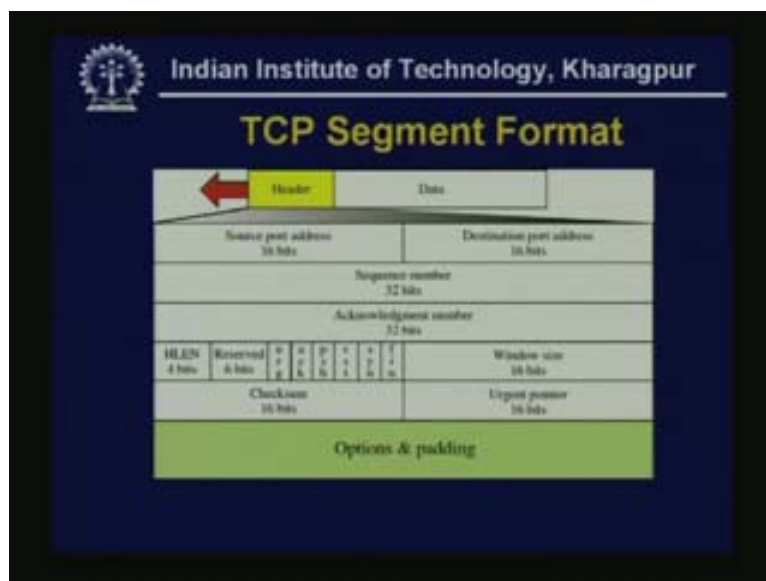
Indian Institute of Technology, Kharagpur

## TCP Segment

- The unit of data transfer between devices using TCP is a segment.
- 20 to 60 byte header, followed by data.
- 20 byte header if there are no options and up to 60 bytes if it contains some options.

The unit of data transfer between devices using TCP is a segment. It is 20 to 60 bytes header followed by data. It is a 20 byte header if there are no options and up to 60 bytes if it contains some options. So there can be up to 40 bytes of option. So the header naturally contains some field which allows the TCP protocol to run. Now let us look into the details of this header.

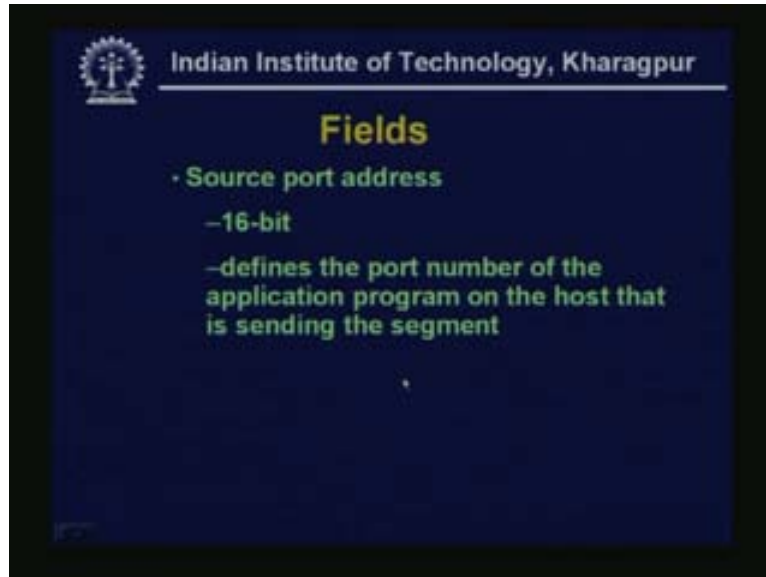
(Refer Slide Time: 09:37 – 09:02)





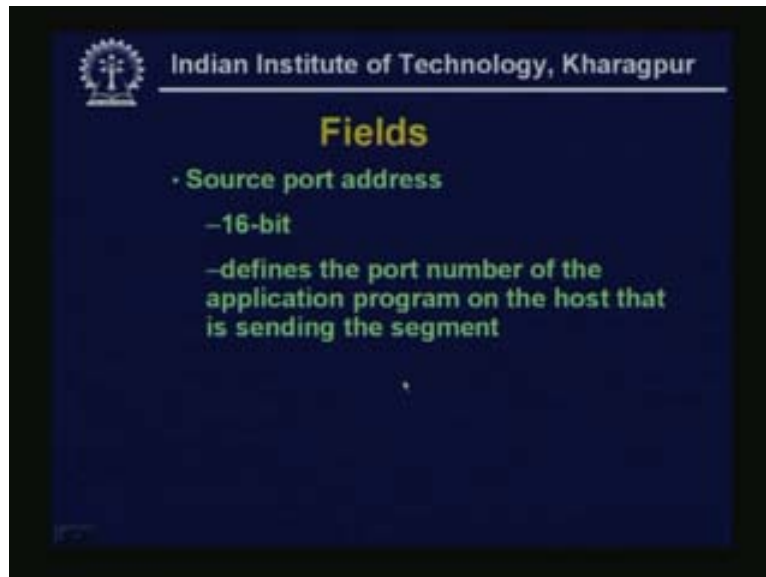
The Header has 16-bit source port address and 16-bit destination port address. It also has a sequence number, acknowledgement number, Header length (HLEN), reserved bits, some flags and then there are some options like window size, checksum, urgent pointer, options and padding.

(Refer Slide Time: 10:21- 10:30)



So there is a source port address. In the client-server processes actually communicate through ports. And the port and the IP address together form the socket which uniquely identifies every session. When a TCP session is going on, on both the sides two port numbers are assigned. For standard applications the first communication will start on well-known port number then they will switch to ephemeral port numbers.

(Refer Slide Time: 10:31 – 10:51)



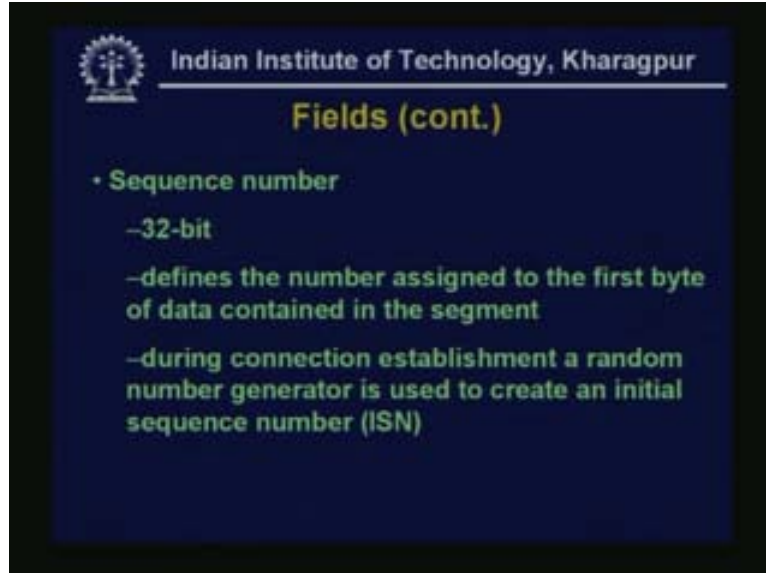
Then there is a source port address which is a 16-bit address that defines the port number of the application program on the host that is sending the segment.

(Refer Slide Time: 10:51 – 11:15)



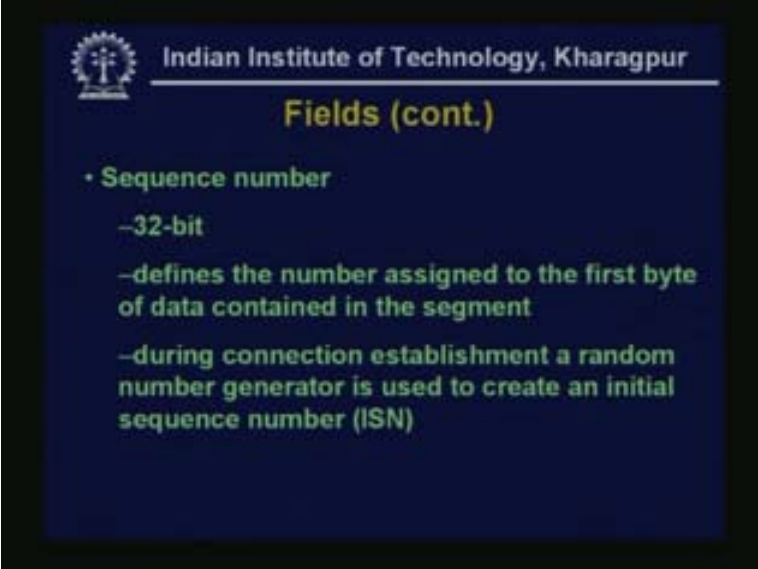
Similarly there is destination port address which is again 16-bits. The port number is from 1000 to 65000 and those are the ephemeral port numbers. Destination port address defines the port number of the application program on the host that is receiving the segment.

(Refer Slide Time: 12:50 -12:55)



There is the sequence number which is a 32 bit number. It defines the number assigned to the first byte of data contained in the segment. During connection establishment a random number generator is used to create an Initial Sequence Number (ISN). The field for the sequence number has been kept quite big, it is a 32 bit and is kept with a reason. If the number of bits for the sequence number was small then what could happen is that, when a particular session starts it sends the number and then it would go back to the beginning. For any finite length sequence number after some point it is going to go back. Now, if it goes back and starts these numbers once again for a very small segment size what might happen is that first of all the two packets may get the same segment number which are on the network at the same time and similarly other kinds of confusion might arise. So what is done is that, actually a large sequence number is generated so that even if it comes back to a low number we can know that which one came earlier and which one came later. If after receiving very high numbers if you start getting some low sequence numbers you know that it has looped back. Actually it may not be a strict loop.

(Refer Slide Time: 13:27 -14:28)



Indian Institute of Technology, Kharagpur

### Fields (cont.)

- Sequence number
  - 32-bit
  - defines the number assigned to the first byte of data contained in the segment
  - during connection establishment a random number generator is used to create an initial sequence number (ISN)

Therefore this is the range of sequence numbers. So  $2^{32}$  is about 4 billion which is a large number and random number is used to generate the Initial sequence number. This initial sequence is expected in a large range so it is not going to clash and this ISN is exchanged between the two nodes.

(Refer Slide Time: 14:29 – 14:44)



Indian Institute of Technology, Kharagpur

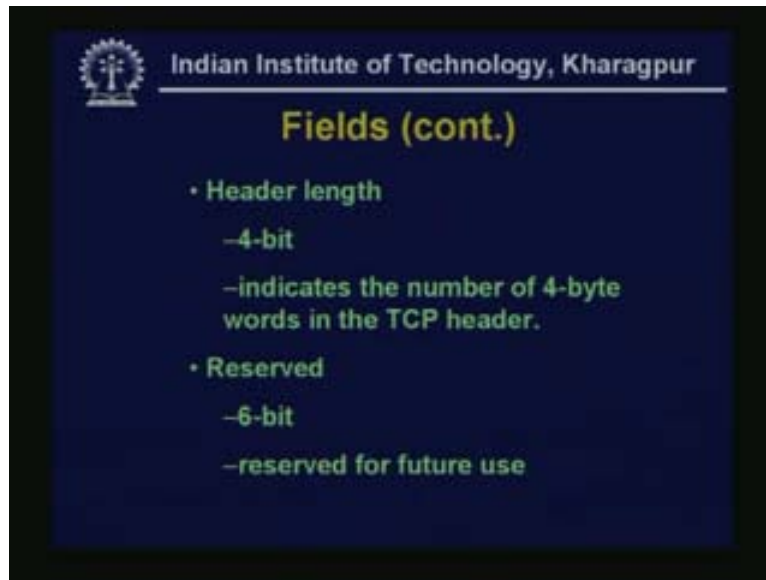
### Fields

- Acknowledgment number
  - 32-bit
  - defines the byte number that the receiver of the segment is expecting to receive from the other party.

There is an acknowledgment number which is 32 bit. Just as the segments which have been sent they have a sequence number. Similarly, these segments as they arrive on the other side will get an acknowledgment. So it is one acknowledgment after the other so a

stream of acknowledgements will come. Actually in the best of circumstances there will be as many acknowledged numbers as there are segments which are sent. So this acknowledged number will start somewhere and that would be communicated by the receiver to the sender in the connection set up phase which is also a 32 bit number that is large. It defines the byte that the receiver of the segment is expecting to receive from the other party. This is standard norm for the sliding window protocol.

(Refer Slide Time: 14:53-14:57)



The next field in the header is the header length which is of 4 bit. This indicated the number of 4 byte word in the TCP header. This is required because the header is of variable length. The header could be anything from 20 bytes to 60 bytes depending on the options. So the Header length is to be specified and then there are some reserved bits for future use followed by some flags. It defines 6 controls flags.

(Refer Slide Time: 14:58 -15:51)

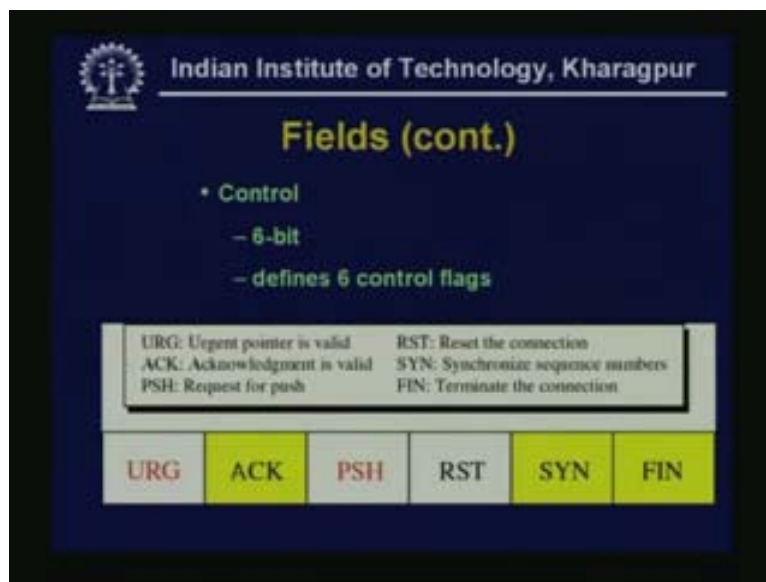


Indian Institute of Technology, Kharagpur

### Fields (cont.)

- Header length
  - 4-bit
  - indicates the number of 4-byte words in the TCP header.
- Reserved
  - 6-bit
  - reserved for future use

(Refer Slide Time: 15:51 – 16:17)



Indian Institute of Technology, Kharagpur

### Fields (cont.)

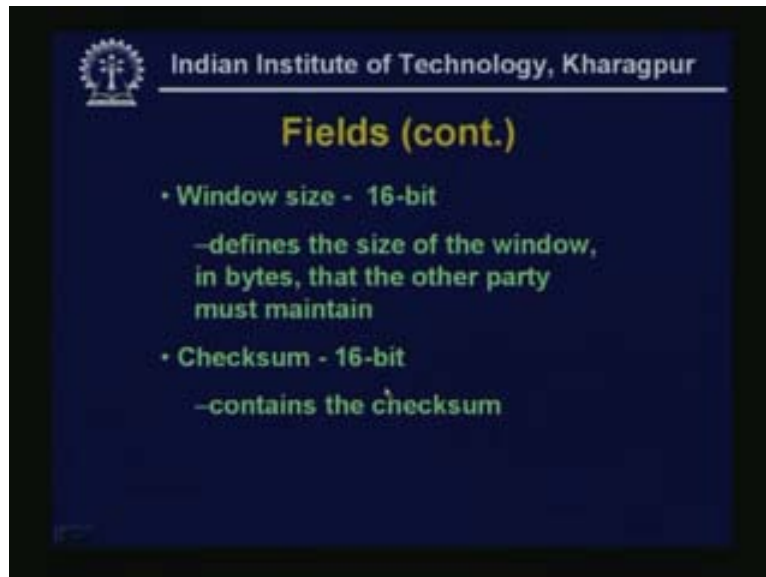
- Control
  - 6-bit
  - defines 6 control flags

URG: Urgent pointer is valid	RST: Reset the connection
ACK: Acknowledgment is valid	SYN: Synchronize sequence numbers
PSH: Request for push	FIN: Terminate the connection

URG ACK PSH RST SYN FIN

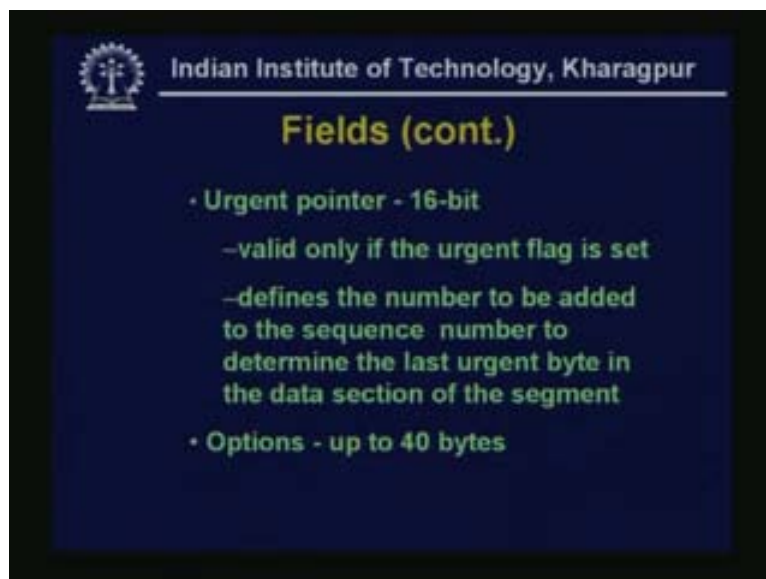
URG defines the urgent pointer which determines whether the urgent pointer is valid or not. If the urgent pointer is not valid the urgent pointer field itself may contain some garbage. ACK tells whether acknowledgement is valid or not. PSH is for request for push. RST is for resetting the connection. SYN is to synchronize sequence number and FIN is to terminate these connections. These flags are used for setting up and termination of connections.

(Refer Slide Time: 16:17 -16:35)



There is a window size of 16-bit which defines the size of the window in bytes that the other party must maintain. It is sort of controlled by a receiver. The receiver gives the window size which is of the sliding window protocol. And then there is a Checksum which is of 16-bits. The checksum is like UDP.

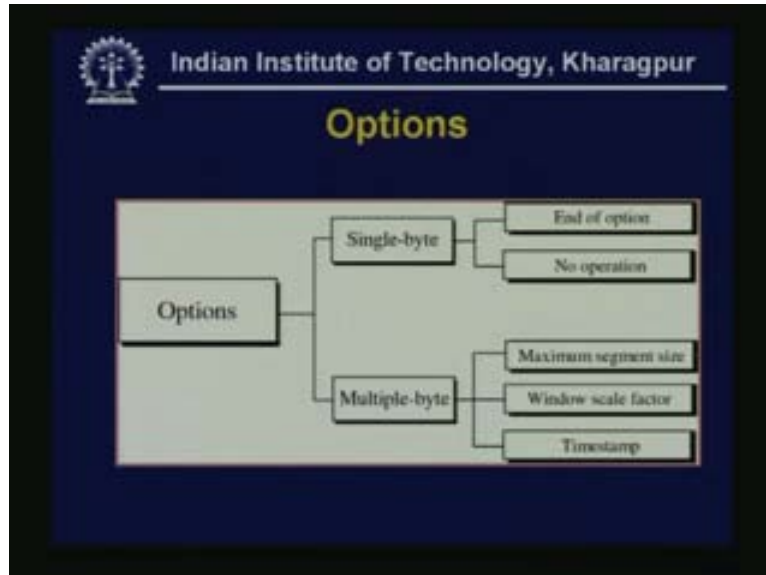
(Refer Slide Time: 16:36 – 16:58)



The urgent pointer is 16-bits and is valid only if the urgent flag is set. It defines the number to be added to the sequence number to determine the last urgent byte in the data section of the segment.

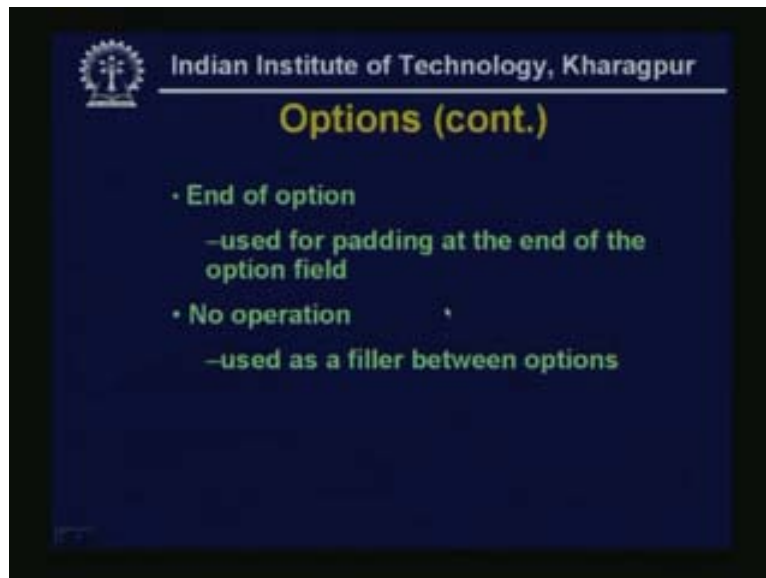


(Refer Slide Time: 16:55- 17:02)



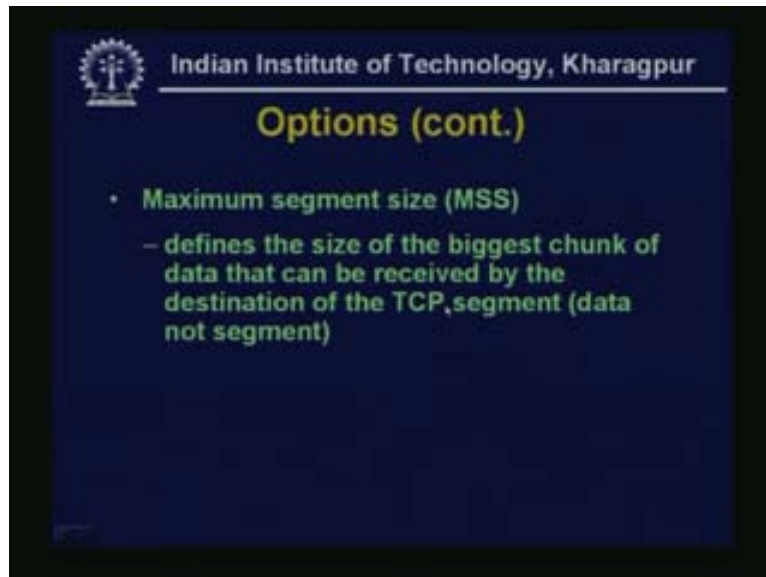
Options are up to 40 bytes. Options could be a single-byte or multiple-byte. Multiple-byte in options may contain maximum segment size, windows scale factor and timestamp etc. In single-byte there are end of options and no operation for padding purpose.

(Refer Slide Time: 17:03 -17:24)



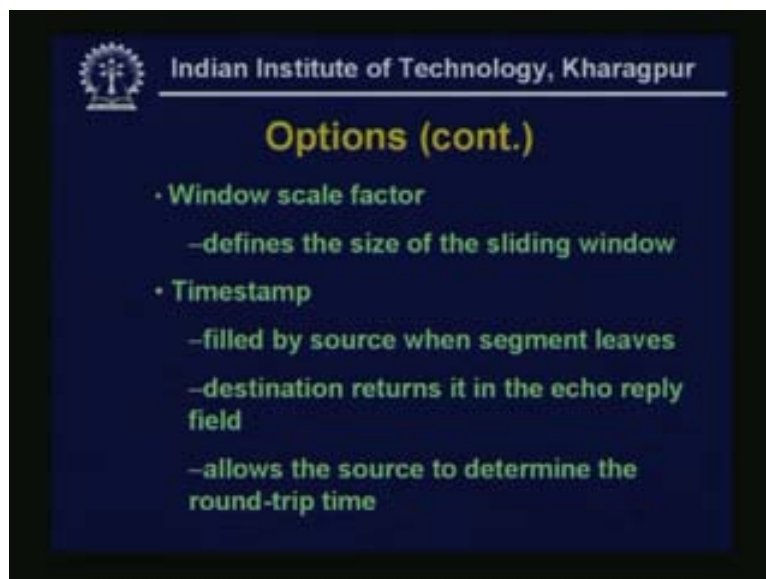
End of option is used for padding at the end of the option field and no operation is used as filler between the options.

(Refer Slide Time: 17:24- 17:51)



If there are options on the other hand then one option is the maximum segment size which defines the size of the biggest chunk of data that can be received by the destination of the TCP segment. This is not the segment but the size of the data which is taking the header field out.

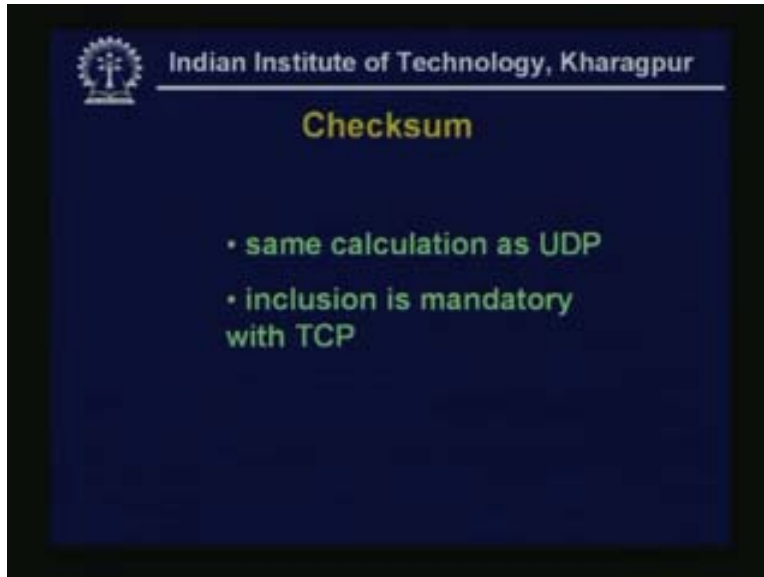
(Refer Slide Time: 18:53 -19:00)



The window scale factor defines the size of the slicing window and how it is changed. Timestamp is filled by source when segment leaves and destination returns it in the echo reply field. This allows the source to determine the round trip time. Some estimation about the round trip time is very essential because suppose a particular segment has been

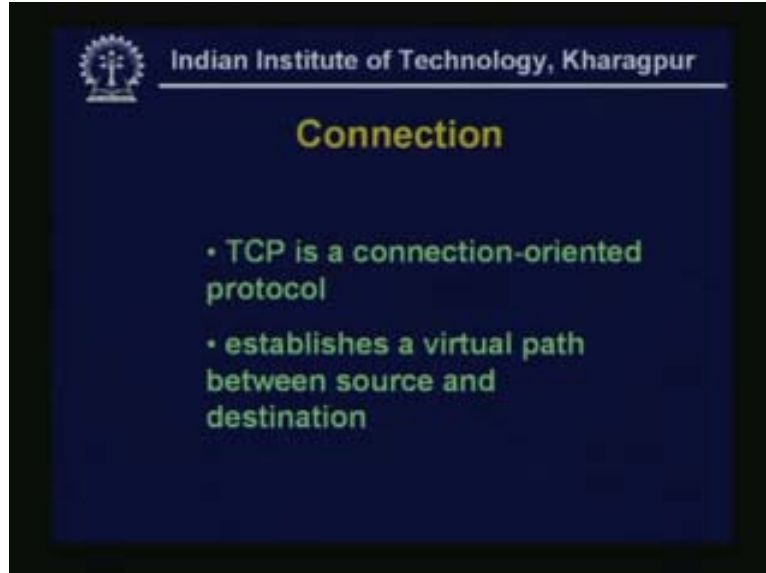
sent then the sender is expecting an acknowledgement. If the acknowledgement does not arrive on time, then how long would the sender wait for the acknowledgement? So you have to make an estimate and that estimate is based on the round trip delay, the maximum segment life. So it has to wait that much or may be some more and only then the sender would sort of come to the conclusion that may be the original packet is lost or may be the acknowledgement is lost. But in either case the packet or the segment that was send it has to be retransmitted.

(Refer Slide Time: 19:01 – 19:24)



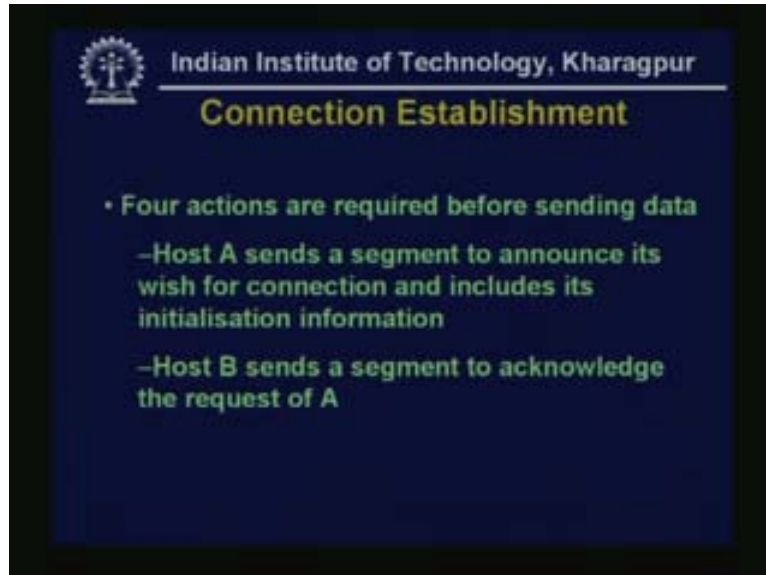
There is a Checksum, it has the same calculation as UDP and inclusion is mandatory with TCP.

(Refer Slide Time: 20:21 – 21:10)



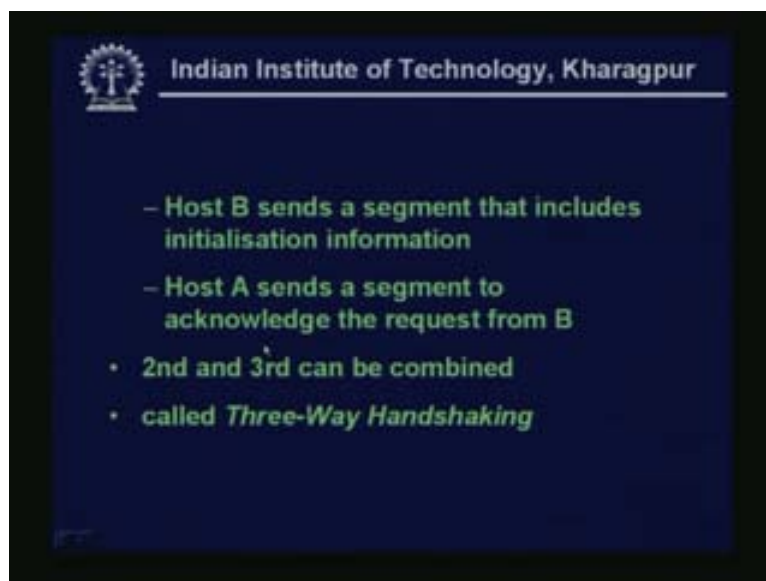
Setting up connections and resetting of connections: TCP is a connection-oriented protocol. So, a virtual path between the source and the destination has to be established. This is only a virtual path because remember that TCP is working on IP that means an IP is a connectionless service. So the underlying actual service is actually connectionless. But TCP gives a feeling to the upper layer, this application layer that, as if it is a connection-oriented thing. That means all good things about connection orientation like segments, arriving in order and reliability etc is present. But TCP has to work on a connectionless IP network. So this path established from source to the destination is only a virtual path unlike traditional strict connection-oriented service where the connection may be physical.

(Refer Slide Time: 21:10 -21:21)



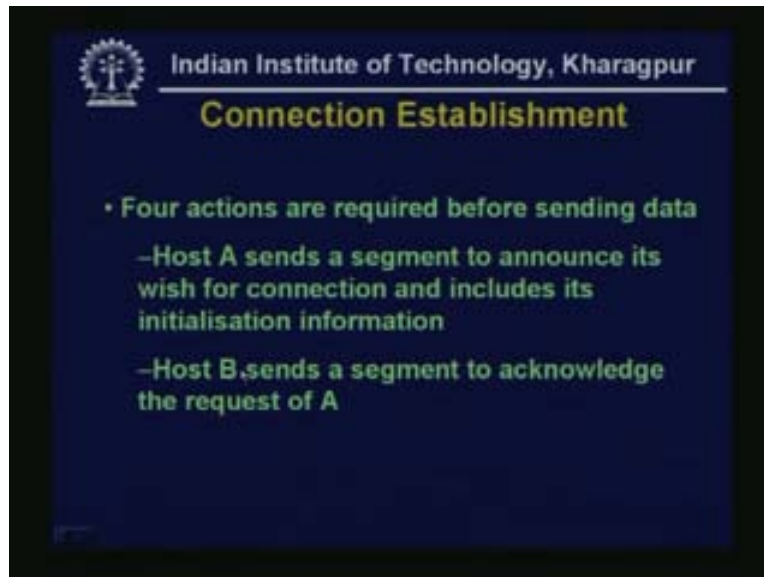
For connection establishment four actions are required before sending data. And here two of them may be combined so it can even be three actions. First host A sends a segment, if host A is the client and it wants to establish a connection to the server may be a host B, so host A sends a segment to announce its wish for connection and includes its initialization information. And then host B sends a segment to acknowledge the request of A. Here host B sends back an acknowledgement.

(Refer Slide Time: 21:21-21:24)



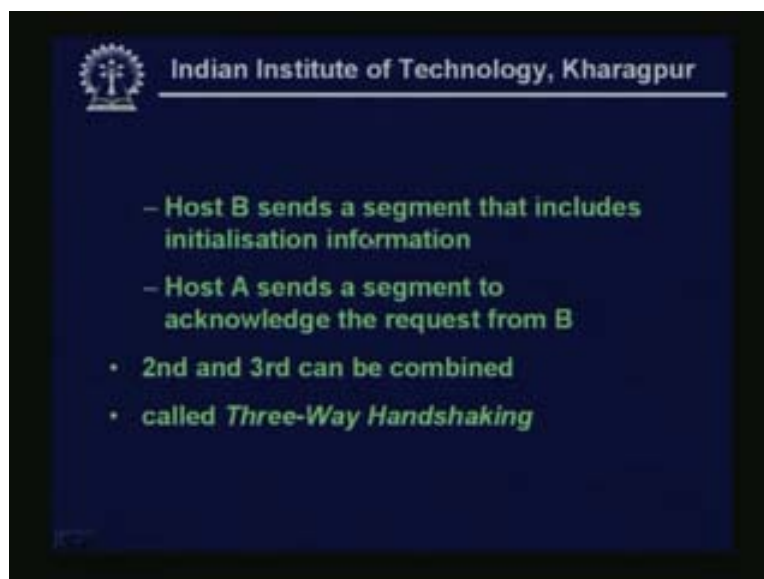
Then host B sends a segment that includes the initialization information. So here the second and third steps can be combined.

(Refer Slide Time: 21:24 - 21:47)



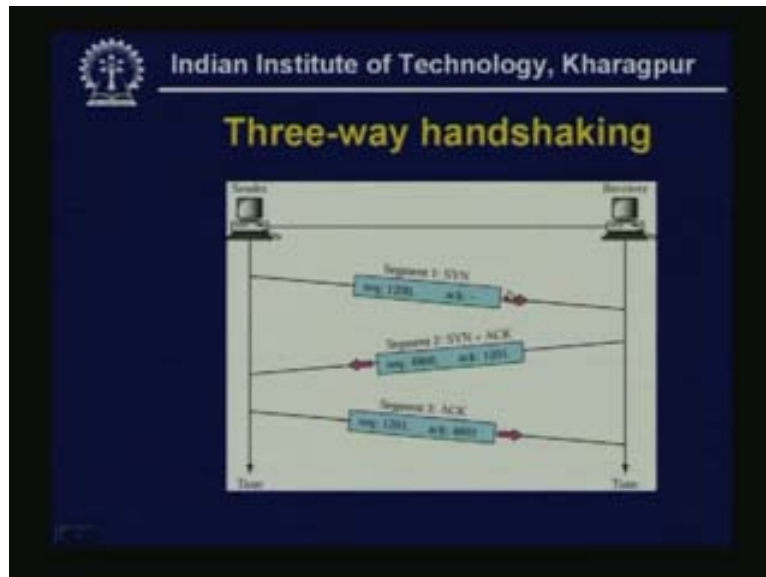
That means the host B can acknowledge the request of host A as well as in the same acknowledgement it might send the initialization information meaning the sequence number etc have to be exchanged.

(Refer Slide Time: 21:47- 24:09)



Then host A sends a segment to acknowledge the request from B. Second and third can be combined which is called as the Three-Way Handshaking.

(Refer Slide Time: 24:09-24:30)



For the first SYN, that is, for synchronization the sender sends a synchronization which is a request to set up a connection to the host. And it also gives initial sequence number. The receiver then sends a SYN, its own SYN the sequence number which is something like 4800 and an acknowledgement of 1200. This segment which is the segment 1 already consumes the first sequence number 1200. So, while acknowledging it sends the value 1201 meaning that 1201 is the next segment or next segment number that the receiver is expecting. What might happen is that, this packet may get lost because once again the TCP is sitting on a best effort kind of IP service so it may get lost.

Therefore, what would happen to the sender is that, the sender will not get any acknowledgement so after some time it will send the SYN pack again if it is persistent and then finally it will get the SYN and the acknowledgement. Similarly, if this is lost once again this is sent after sometime and finally this is achieved and then the sender sends the sequence number 1201 and acknowledgement 4801. So this sequence number is always the next number that is expected from the other side. When it sends sequence number 1200 it replies back saying that next it is expecting 1201 and its starting number is 4800. So he replies in his acknowledgement that the next acknowledgment he is expecting is 4801. So this is the Three-Way Handshake where the SYN and the ACK has been combined and then the data transmission can start.



(Refer Slide Time: 24:53-24:58)



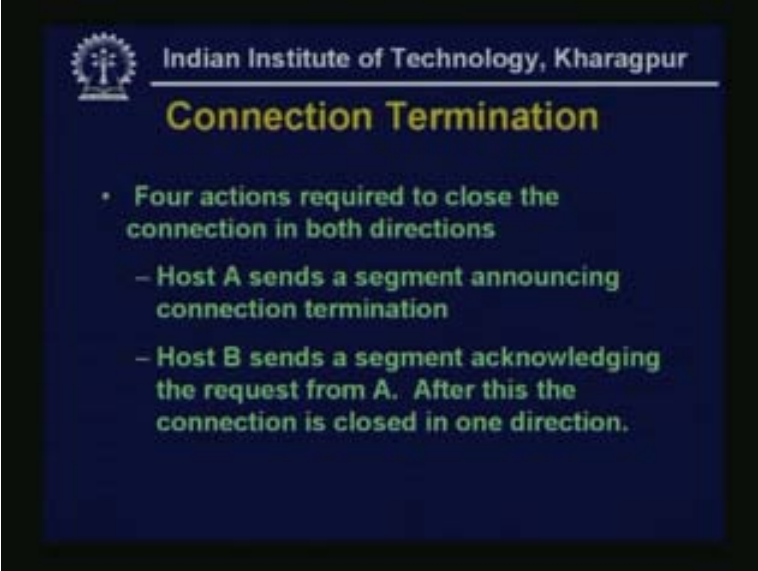
On the other side we have to think about the connection termination. To terminate the connection either party can close the connection. If connection is terminated in one direction data can continue to be sent in the other direction. Remember that this is the full loop-less communication which means A is communicating to B and B is communicating to A at the same time. Even if only one side is sending data to the other side the acknowledgement is coming from the other side anyway. Now, the connection can be terminated from the both sides but then somebody has to initialize the termination.

(Refer Slide Time: 24:58 –25:22)



So, if connection is terminated in one direction data can be continued to be sent in the other direction.

(Refer Slide Time: 25:27 – 25:34)



Indian Institute of Technology, Kharagpur

## Connection Termination

- Four actions required to close the connection in both directions
  - Host A sends a segment announcing connection termination
  - Host B sends a segment acknowledging the request from A. After this the connection is closed in one direction.

Four actions are required to close the connection in both directions. First, host A sends a segment announcing connection termination. This means it sends the segments containing FIN.

(Refer Slide Time: 25:34-25:44)



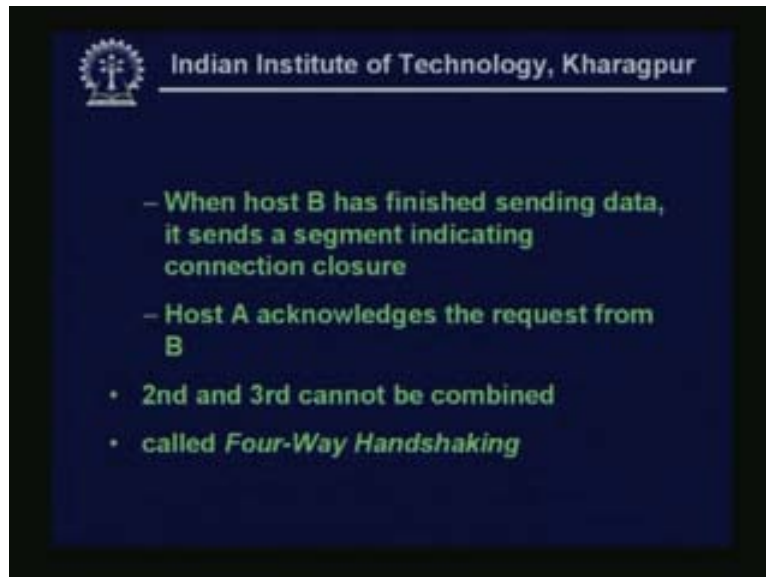
Indian Institute of Technology, Kharagpur

## Connection Termination

- Four actions required to close the connection in both directions
  - Host A sends a segment announcing connection termination
  - Host B sends a segment acknowledging the request from A. After this the connection is closed in one direction.

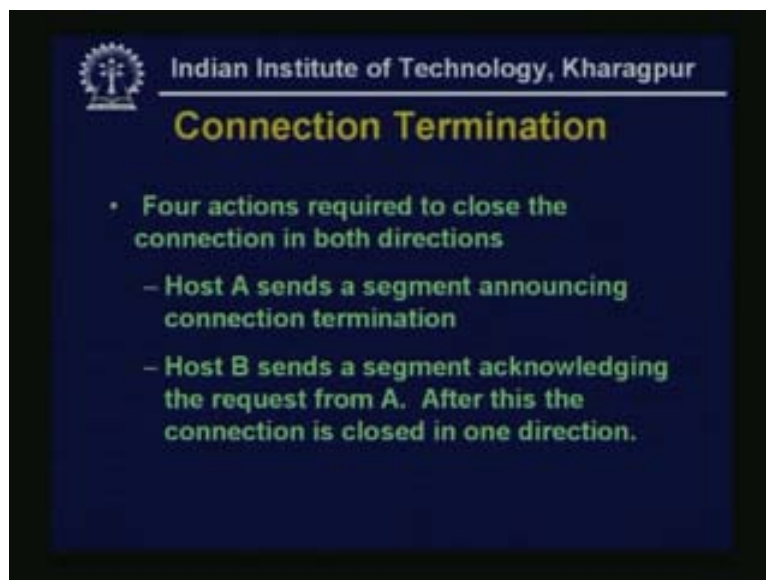
Host B sends a segment acknowledging the request from A and after this the connection is closed in one direction.

(Refer Slide Time: 25:44-25:50)



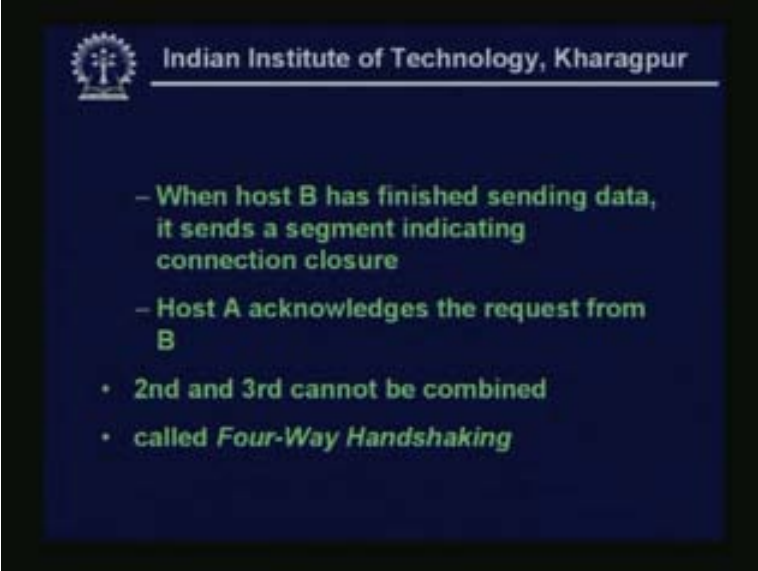
When host B has finished sending data it sends a segment indicating connection closure.

(Refer Slide Time: 25:51-25:55)



Here, the second step and the third step cannot be combined together.

(Refer Slide Time: 25:55 – 26:10)



Indian Institute of Technology, Kharagpur

- When host B has finished sending data, it sends a segment indicating connection closure
- Host A acknowledges the request from B
- 2nd and 3rd cannot be combined
- called *Four-Way Handshaking*

(Refer Slide Time: 26:11-26:24)



Indian Institute of Technology, Kharagpur

### Connection Termination

- Four actions required to close the connection in both directions
  - Host A sends a segment announcing connection termination
  - Host B sends a segment acknowledging the request from A. After this the connection is closed in one direction.

Although we are sort of allowing the termination of connection for one side but on the other side it may have acknowledgements or other things to send to this side so it will not terminate the connection. Therefore these two cannot be combined together.

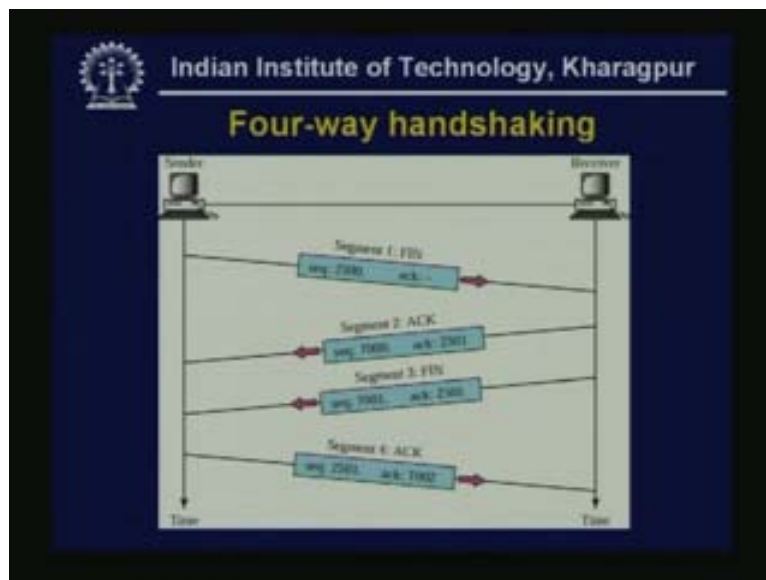
(Refer Slide Time: 26:24- 29:32)

Indian Institute of Technology, Kharagpur

- When host B has finished sending data, it sends a segment indicating connection closure
- Host A acknowledges the request from B
- 2nd and 3rd cannot be combined
- called *Four-Way Handshaking*

The third step can be taken only when host B has finished sending data from its side and it sends a segment indicating a connection closure. Host A acknowledges the request from B. So this is called a Four-Way Handshaking.

(Refer Slide Time: 29:32-29:36)

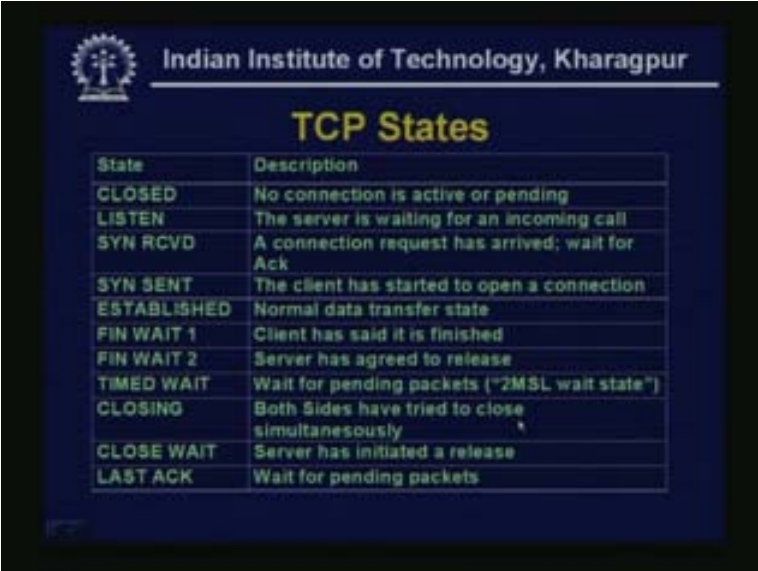


This is the diagram showing Four-Way Handshaking. For example, assume that the sender has sent a FIN in a segment 2500 and when it receives the FIN if everything is all right he will send back an ACK and he say he will say that the next one he is expecting from the other side is sequence 2501 and sequence 7000 which is the acknowledgement for this FIN. After sometime when B has finished sending all its acknowledgements and

other things it might want to send to the sender then it is sends a FIN from this side. So, this is for closing the connection from the other side. The acknowledgement that is expected once again is 2501 and this is still 2501 because nothing else has arrived from the other side and then it takes the next segment number 7001 from this side. And he acknowledges this second FIN when the acknowledgement reaches here so everything is closed gracefully.

Of course things may not run so well because one or more of these packets may get lost on the way because we know that the underlying network is unreliable. If the first FIN is missed he does not get any acknowledgement and after sometime he will send the FIN again. Similarly, if this ACK is not received then he will send this FIN once again anyway. So this FIN may be lost. Since he does not get any acknowledgement he will send the FIN after sometime. The trouble is over here because at this point of time after sending this acknowledgement the sender will assume that everything is fine so he will close. But this last acknowledgement may get lost. As all the FINs have been sent and received and the acknowledgement has been received and sent he will say that it is the end of the story but for him it is not end of the story because this ACK is now lost but if this fellow has already completed he will not send any ACK any longer. So what he will have to do is that after sometime he will have to close the connection.

(Refer Slide Time: 29:36-29:42)

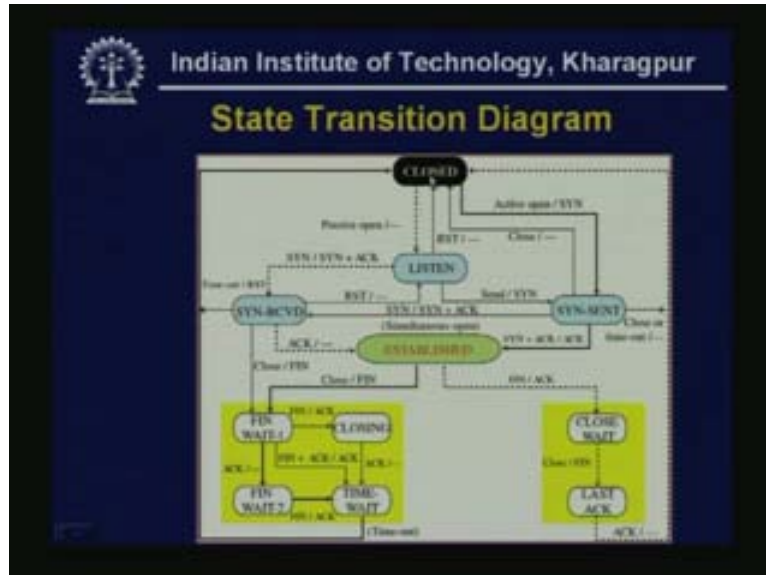


State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for Ack
SYN SENT	The client has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Client has said it is finished
FIN WAIT 2	Server has agreed to release
TIMED WAIT	Wait for pending packets ("2MSL wait state")
CLOSING	Both Sides have tried to close simultaneously
CLOSE WAIT	Server has initiated a release
LAST ACK	Wait for pending packets

Actually the TCP goes through a state diagram.




(Refer Slide Time: 29:42-31:11)



This is the state diagram and these are the different states.

(Refer Slide Time: 31:11-32:21)

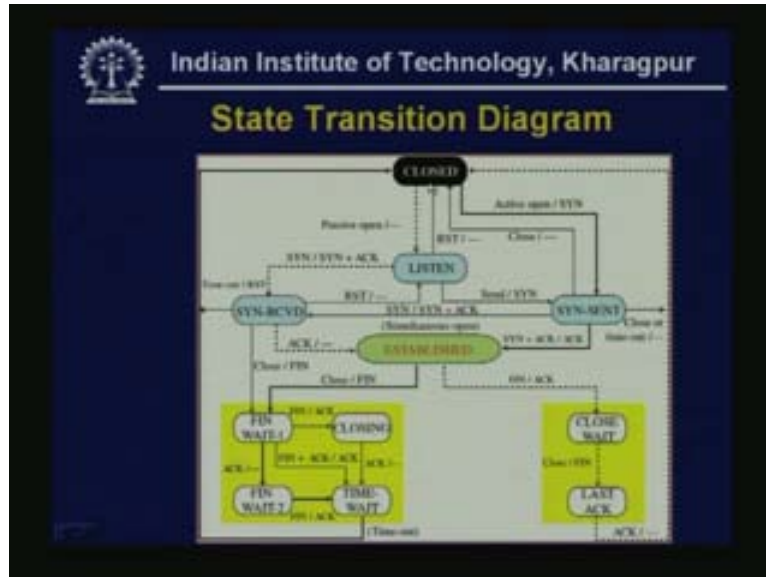
 <b>Indian Institute of Technology, Kharagpur</b>	
<h2 style="text-align: center;">TCP States</h2>	
State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for Ack
SYN SENT	The client has started to open a connection
ESTABLISHED	Normal data transfer state
FIN WAIT 1	Client has said it is finished
FIN WAIT 2	Server has agreed to release
TIMED WAIT	Wait for pending packets ("2MSL wait state")
CLOSING	Both Sides have tried to close simultaneously
CLOSE WAIT	Server has initiated a release
LAST ACK	Wait for pending packets

One is of course closed that means no connection is active or pending. The other is listening. May be the server is waiting for an incoming call. SYN RCVD means somebody has made a request, a connection request has arrived and is waiting for acknowledgement. SYN SENT: The client has started to open a connection. ESTABLISHED means a connection has been established and that is the normal data transfer state. FIN WAIT 1 means client said it is finished. FIN WAIT 2 means server has agreed to release. TIMED WAIT means wait for pending packets. This is the 2 MSL wait



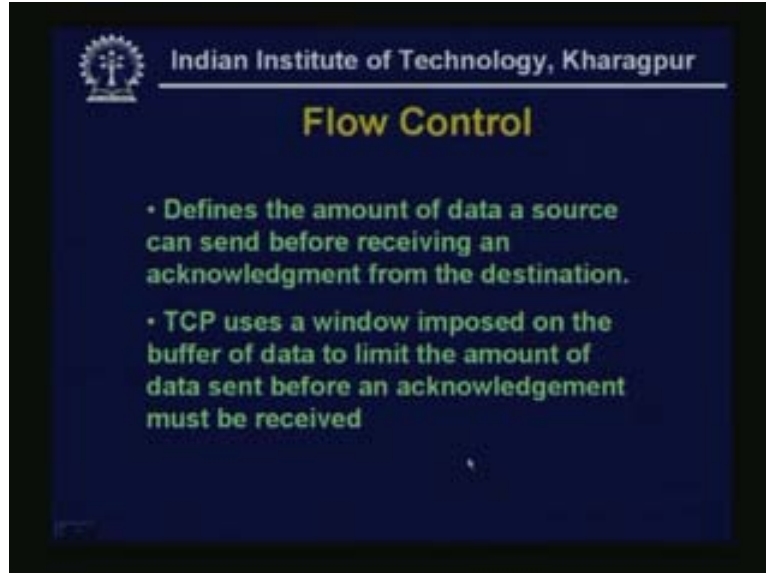
state so for the last pending packet to come in you have to wait this much. This is the maximum segment and this is sort of estimated from the round trip time. CLOSING means both sides have tried to close simultaneously. CLOSE WAIT means server has initiated a release and a LAST ACK is waiting for the pending packets.

(Refer Slide Time: 32:21-33:05)



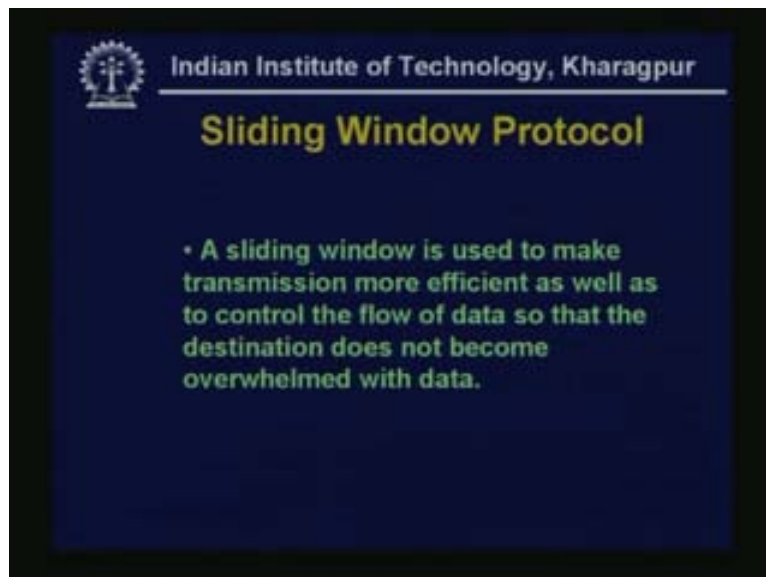
When it is closed nothing is happening. Now some SYN or SYN plus ACK it receives and once it receives that then it goes to the SYN RCVD state. As explained earlier this is the connection set up phase and once the connection is set up then it reaches the Established phase and then the two way communication is going on. Then, after this, it goes through a closure either through the closing and then FIN WAIT 1 and FIN WAIT 2, TIME WAIT etc and here it is the close width and the last ACK.

(Refer Slide Time: 33:05-33:17)



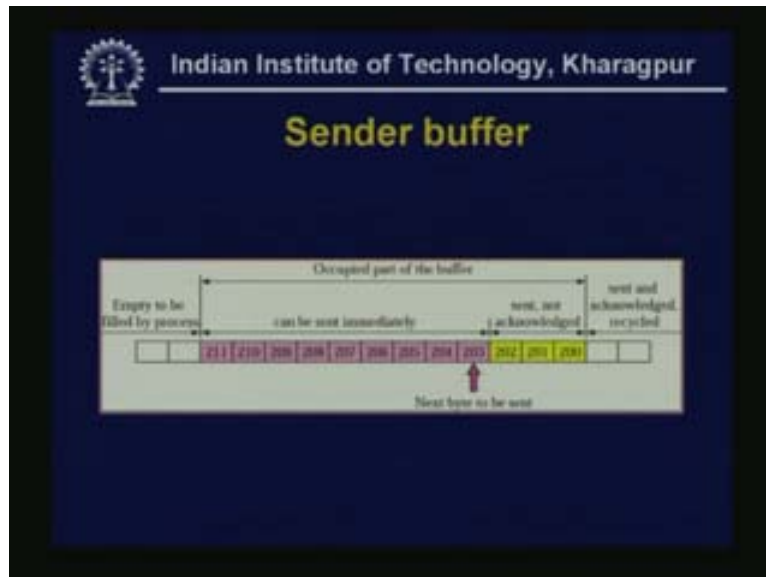
The flow control is implemented by TCP. This defines the amount of data a source can send before receiving an acknowledgment from the destination. TCP uses a window imposed on the buffer of data to limit the amount of data sent before an acknowledgement must be received. This is the traditional sliding window protocol.

(Refer Slide Time: 33:18 -34:21)



In the Sliding Window Protocol a sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data.

(Refer Slide Time: 34:23-35:01)



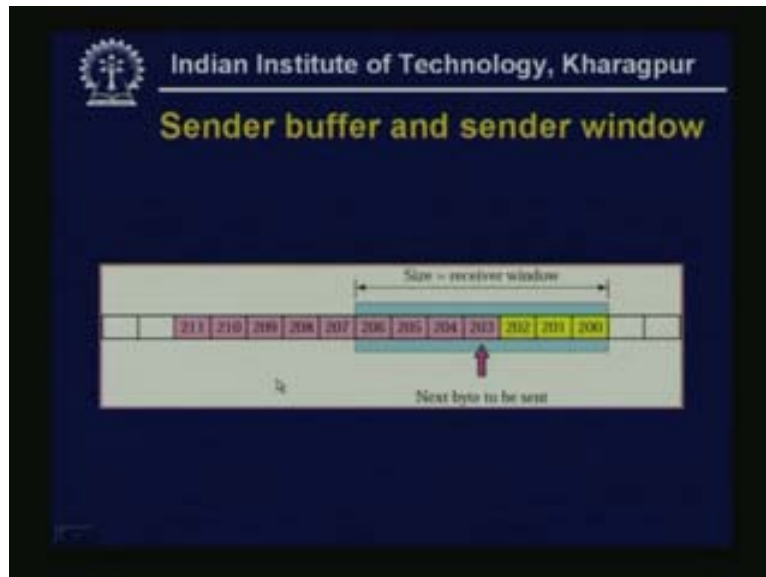
In sender buffer there is the occupied part of the buffer and out of this some number or packets or segments have been sent but they have not been acknowledged and these are the next byte to be sent which is available in the buffer. Now, depending on the windows size it can go on sending these bytes. Or if the window size is already over then it has to wait here. Let us assume that the size of the window is three units, in that case the sender has to wait here till some acknowledgment comes. And if the acknowledgment comes suppose 200 is acknowledged then the window will automatically slide and this now can be sent.

(Refer Slide Time: 35:02-35:22)



At the receiver side it has received 194 to 199. This is the occupied part of the buffer and from this buffer the destination process will keep on consuming the data from these segments as a stream of bytes going out of this segment. Here, this is the empty part of the buffer where new segments can come in.

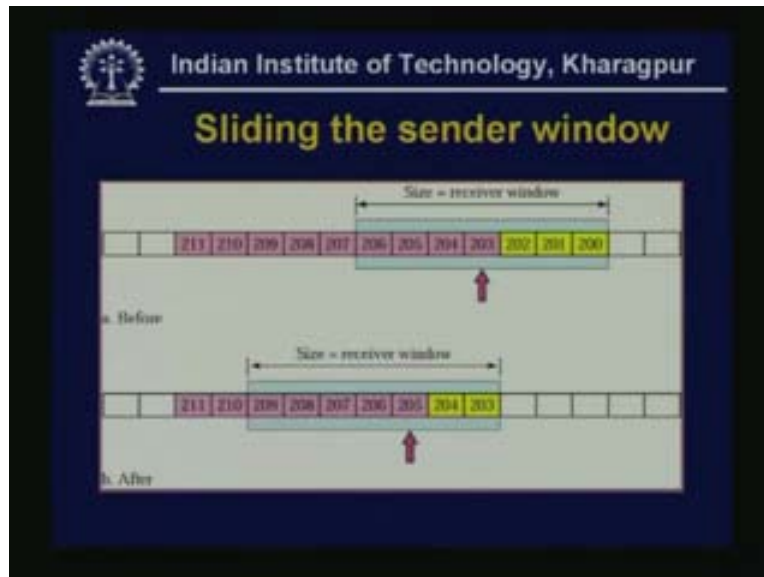
(Refer Slide Time: 37:48-39:55)



Suppose the size of the receiver window is 7 then after having sent this the sender can go on sending up to the receiver window size. The point is that, if a window size is not fixed what the sender might do is that, the sender may send in a lot of packets. Or may be the channel is absolutely bad, in that case none of the packets have gone to the other side so you will have to send all these packets once again, and that is one point. And secondly the sender may be very fast but the network may be congested. So the sender will push in a lot of data but that will only make the congestion worse. So whatever would have been received on the other side does not reach that side because the intervening network has been congested, or may be some buffer has overflowed or some router has dropped some packets etc so there is a limit on the window size.

The window size also has a bearing on the speed at which effective data transfer is taking place. If the window size is very small, suppose 2, then after sending two segments the sender has to wait till the acknowledgment comes back. So the efficiency of the channel goes down because for each acknowledgement the packet will reach the other side and then the acknowledgment will travel all the way back so it is a round trip delay. Now suppose the window size is only 1 so after sending each segment or each byte you wait for the round trip time and then you send the next byte. So now the overhead has become very high and now this is the minimum possible rate at which the sender is sending. On the other hand if the window is very large a large amount can be sent even without waiting for acknowledgement.

(Refer Slide Time: 39:55 – 39:58)

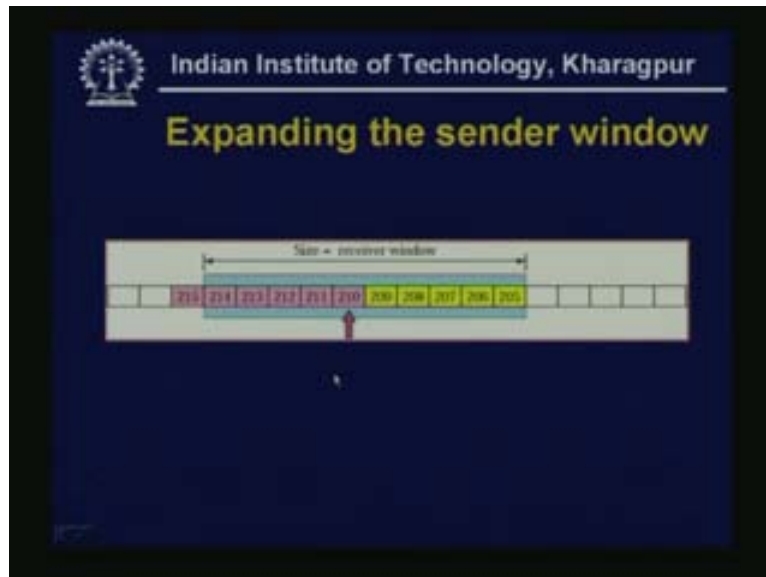


Now when everything is fine the stream of acknowledgements will also start coming in after this round trip delay and the overall efficiency would be high. Therefore between congestion and the efficiency at which the data can be transferred that is the effective data transfer speed there is a **trade-off**.

Suppose if these are the sequence numbers and now the receiver side may be 205 has dropped somewhere and 206, 207, 208, 209 etc has come. So after acknowledging 204 where the acknowledgement packet will tell that the next byte its waiting for is 205 although 206, 207, 208, 209 etc have been received but he will not send any acknowledgement. So after sometime the sender will realize that this acknowledgment for 205 has not come. So he will start pumping 205, 206, 207, 208 and 209 again.

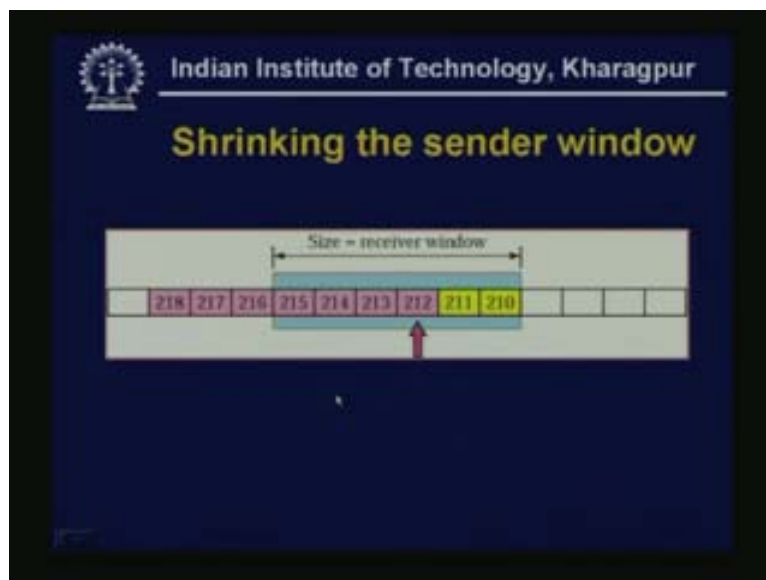
Now as soon as 205 arrive what happens is that this gap is closed and the window will slide all the way here and he will say next byte that is expected is 210. Then what will happen is, 206, 207, 208, 209 now becomes duplicated segments on the receiver side so they are simply dropped. Using the segment numbers not only we can put them in order, suppose 206 comes earlier and 205 has come but has come later but then they can be put in proper sequence because sequence numbers are there. And secondly even this 205 was lost it can still be recovered by this retransmission etc so that is how this protocol is reliable. And when this is acknowledged the window slides.

(Refer Slide Time: 39:59-40:11)



So, this is the size of the receiving window.

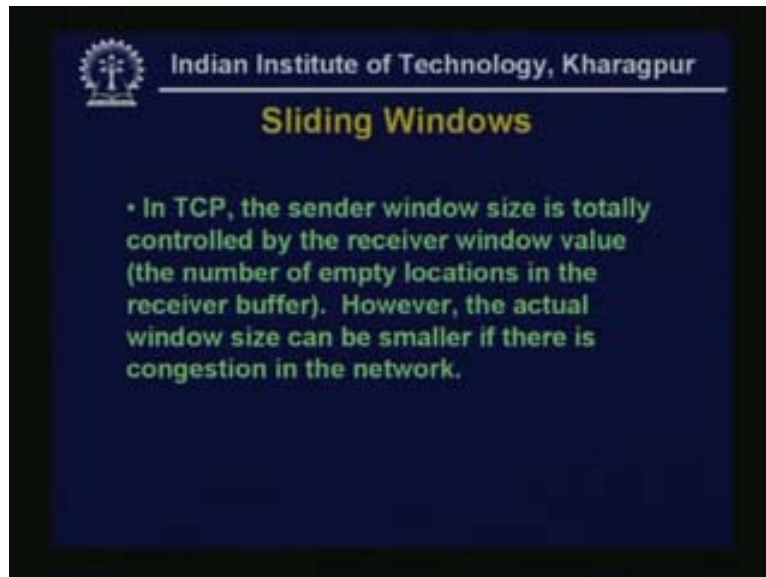
(Refer Slide Time: 41:17-41:31)



Sometimes what happens is that, if there is actual congestion in the network, you will find that only after sending some packets and seeing that some acknowledgements have not come and the packets have got lost. So, the sender thinks, one thing is, this packet may have got lost and secondly, may be the network is congested. So what will happen is that, there is a mechanism for automatically reducing the window size so that now you are going to transfer data at a much lower rate but then you are sort of trying to control the congestion. So this is the congestion control that is built into TCP. There are some

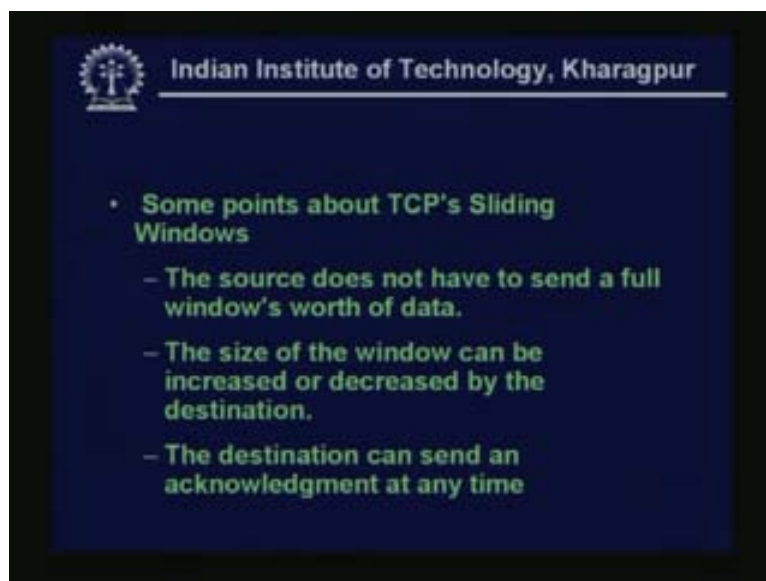
variations as to how this window size is going to be changed over time etc. So we will come to later on we when will talk in more details about congestion control.

(Refer Slide Time: 41:32-41:51)



In TCP the sender window size is totally controlled by the receiver window value, the number of empty locations in the receiver buffer. However, the actual window size can be smaller if there is congestion in the network.

(Refer Slide Time: 41:51- 42:15)

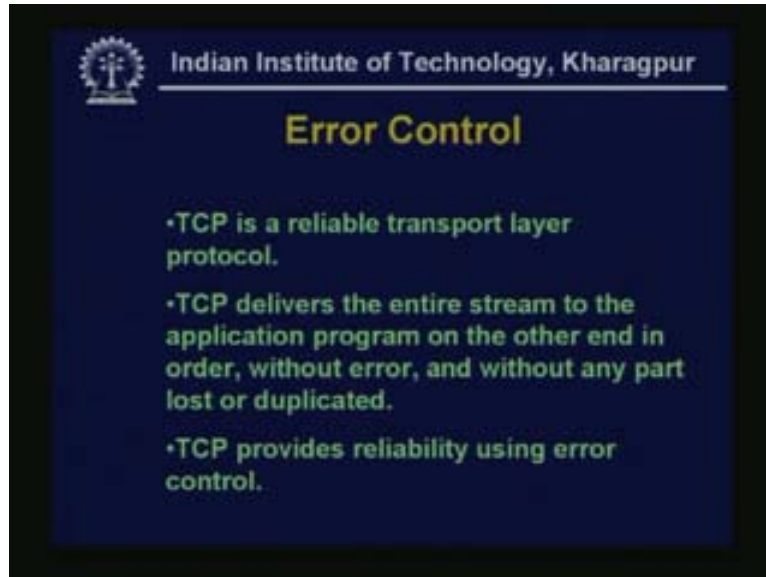


The source does not have to send a full window's worth of data. When you have limited data then you will send only that data which is available. The size of the window can be



increased or decreased by the destination. The destination can send an acknowledgment at any time.

(Refer Slide Time: 42:18 -43:50)



Indian Institute of Technology, Kharagpur

## Error Control

- TCP is a reliable transport layer protocol.
- TCP delivers the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.
- TCP provides reliability using error control.

Error Control:

The TCP is a reliable transport protocol. TCP delivers the entire stream to the application program on the other end in order without error and without any part lost or duplicated. So TCP provides reliability using error control.

(Refer Slide Time: 43:51 – 44:01)



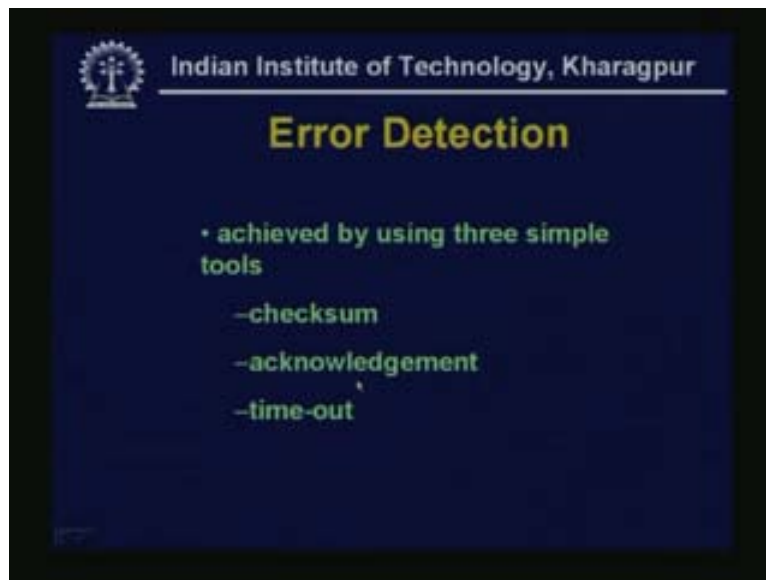
Indian Institute of Technology, Kharagpur

## Error control includes mechanisms for

- detecting corrupted segments
- detecting lost segments
- detecting out-of-order segments
- detecting duplicated segments
- correcting errors after they are detected

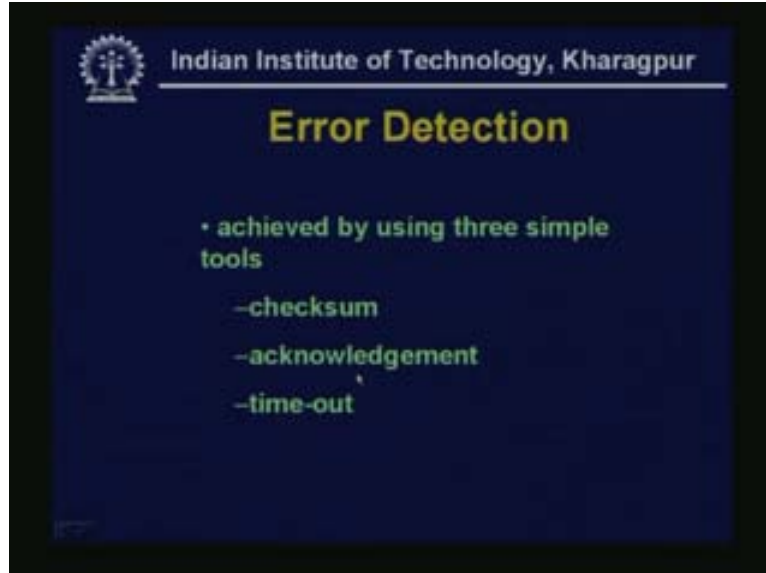
For error control there are the different things that TCP would do. It detects corrupt segments, by detecting lost segments, by detecting out of order segments, by detecting duplicated segments and by correcting errors after they are detected. Detecting corrupt segments is, when checksum would be wrong in that case you can drop it. Or if you can correct the error using the checksum then you may also correct it. Detecting lost segments is achieved by getting the help of the sequence number. When you have the sequence numbers and when one particular segment in between is lost you know that the segment number is lost because that particular segment number would be missing. Detecting segments that are out of order is, when the segments are in out of order we have got the sequence number and so we put them in order. You can detect and correct it at the same time. Detecting duplicated segments: If a particular segment has come twice, may be its acknowledgement is lost or delayed, in that case once again by the serial number we can see that it is duplicated and we can drop it.

(Refer Slide Time: 44:22 – 44:39)



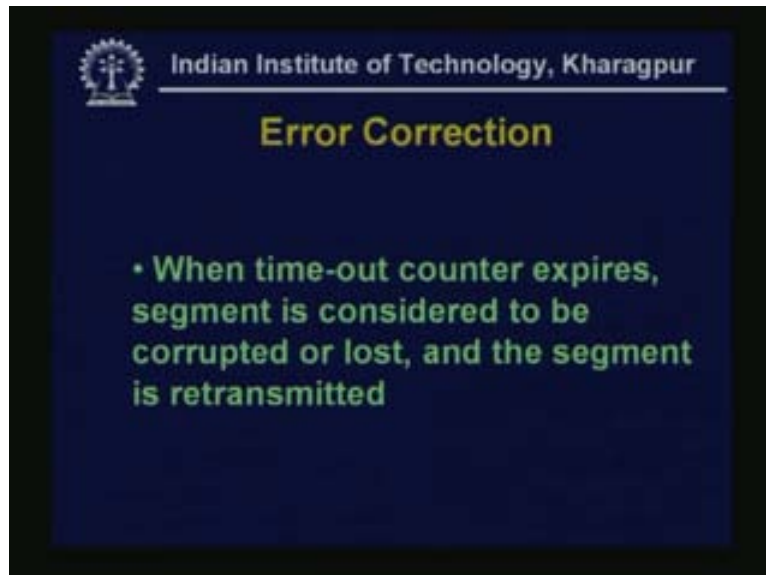
Error detection is achieved by three simple tools: Checksum, acknowledgement and time-out. Suppose you had sent the first segment and the acknowledgement never came then after sometime you have to give a time-out and that is how the sender detects that this segment has been lost and it has to be re-transmitted.

(Refer Slide Time: 44:38 -44:52)



Checksum, acknowledgement and time-out are the three things together offer reliability and error detection and correction capability of TCP.

(Refer Slide Time: 44:52-48:05)



For error correction when time-out counter expires segment is considered to be corrupted or lost and the segment is retransmitted.

(Refer Slide Time: 48:05 – 49:35)



TCP Timers: TCP has to maintain a number of timers inside. TCP has to do a lot of work to give that reliability which is necessary on this unreliable IP service. That is why this protocol is a little more complex than others. But then at the same time this is necessary. For example, when you are sending a File Transfer Protocol for example, you are transferring a file from one machine to another then even the misplacing of few bits will make the whole file useless. So there are applications and specifically protocols like FTP and ACTP etc use the TCP protocol because there is reliability of the connection and absolute error control, it is error free, and the nature of communication is very high. Then there may be applications where a loss of few bits or bytes here and there does not really matter.

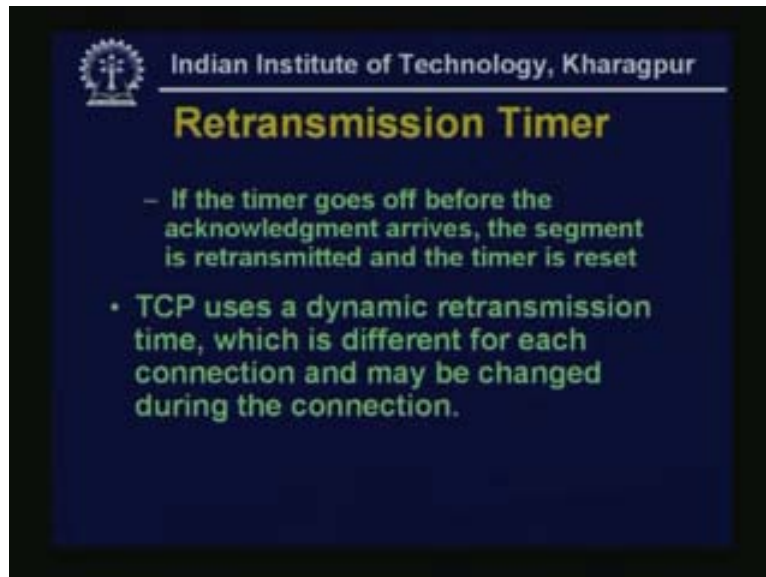
For example, suppose we are carrying a voice communication, now if you are carrying voice communication what would happen is that even if a few bits here and there are lost the quality is not impaired that much. Of course voice or other multimedia kind of communication is very sensitive to other kinds of network parameters. But the point is that, even if few bits are lost here and there then the other person will also be able to make out. So in this case, on the other hand, if there is a delay and differing rate of transmission because if the window size is large the rate at which you can transmit is high and if the window size is small the rate at which you can transmit is small. So, if the rate at which transmission is taking place keeps on varying or takes a lot of time etc is not acceptable for voice communication. So those are the cases where we will not use TCP. There are cases where we definitely want to use TCP because of its reliability and then there are cases where we do not want to use TCP. So FTP and HTTP etc are examples where TCP is necessary and we use them. TCP uses four timers: Retransmission timer, persistence timer, keep-alive timer and time-waited timer.

(Refer Slide Time: 49:35 -51:56)



Retransmission timer: Used to control a lost or discarded segment. Suppose a segment is lost, now, after what time will it be retransmitted? So in this retransmission timer using the timer stamp you can get an idea of the round trip delay and this you will use to set your retransmission timer. And there will be a buffer because it is not constant. By this way you can set up your retransmission timer. As soon as the segment is sent immediately the retransmission timer starts. When the next segment is also sent because your window size happened to be more than 1 then you will need another retransmission timer for this segment. So, for each segment we are maintaining this retransmission timer and as soon as one of these timers is timed-out it means that the packet was sent and the acknowledgement was not received. Therefore, immediately you have to retransmit that packet. So this is the retransmission timer. TCP creates a retransmission timer when it sends a segment. If an acknowledgement is received before the timer goes off the timer is destroyed because this timer is not required. For each new segment you start a new retransmission timer.

(Refer Slide Time: 51:57-52:03)



If the timer goes off before the acknowledgement arrives, that means the acknowledgement has not been received before the estimated time then the segment is retransmitted and the timer is reset. Now this timer will be reset because even though you retransmit the retransmitted segment may also get lost so you have to start the retransmission timer once again. And so TCP uses a dynamic retransmission time which is different for each connection and may be changed during the connection.

TCP is a protocol and there may be a lot of connections going through this TCP protocol. Just think a simple case of a web server. Now there may be a number of people for very active websites and a number of people may be hitting it at almost the same time. So if you have a concurrent server and you have spawn so many processes on the application side and each of the server is sort of serving one particular client and a number of them. So a number of TCP services are going on at the same time but of course they are using different port numbers. So whenever any acknowledgment or anything comes you will see the port number to determine to which process this belongs to. Secondly, now assume that two clients using the same server at the same time. Now the two clients may be connected in two different networks and the round trip time for each of these clients may be different. Since the roundtrip time is different you cannot use the same kind of retransmission time for each of these connections. This has to be dynamically assessed from the time stamp so it may be different for each connection and may be changed during the connection.

(Refer Slide Time: 52:03-52:26)

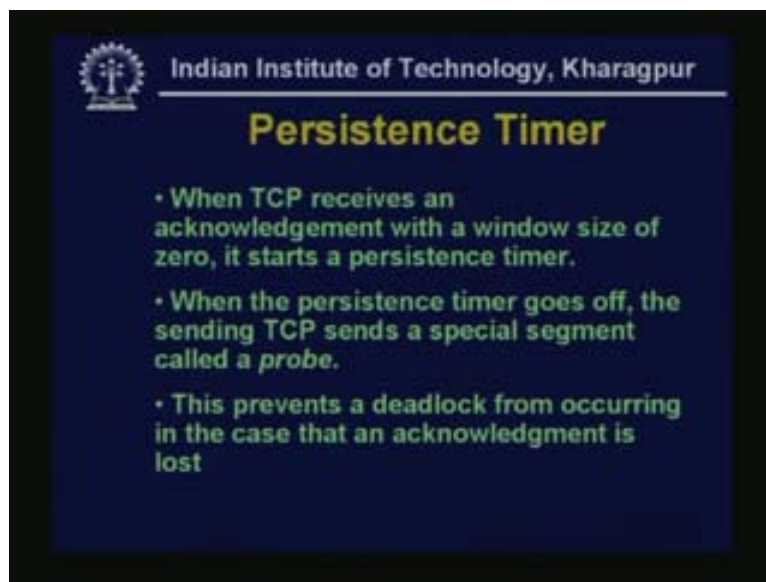


Indian Institute of Technology, Kharagpur

## Retransmission Timer

- If the timer goes off before the acknowledgment arrives, the segment is retransmitted and the timer is reset
- TCP uses a dynamic retransmission time, which is different for each connection and may be changed during the connection.

(Refer Slide Time: 52:40-52:50)



Indian Institute of Technology, Kharagpur

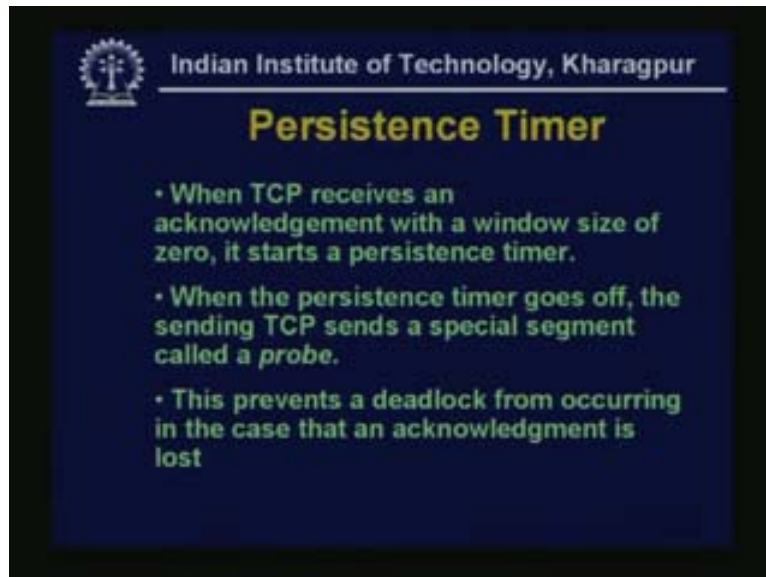
## Persistence Timer

- When TCP receives an acknowledgement with a window size of zero, it starts a persistence timer.
- When the persistence timer goes off, the sending TCP sends a special segment called a *probe*.
- This prevents a deadlock from occurring in the case that an acknowledgment is lost

There is a Persistence timer. When TCP receives an acknowledgment with the window size of 0 it starts a persistence timer. Window size of 0 means, it does not accept anything at the moment because either it is overwhelmed or something is sort of wrong in between. So anyway it is very congested and is not willing to receive anything. But now what would the sender do?



(Refer Slide Time: 53:15- 53:23)



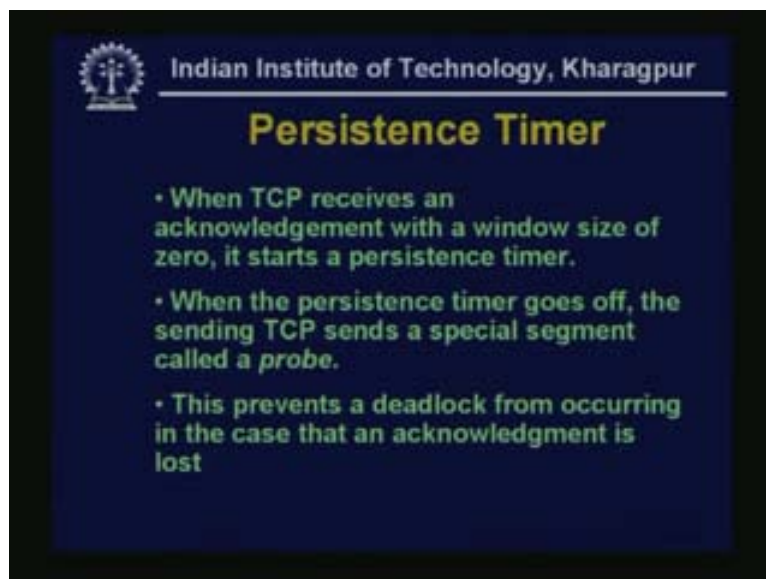
Indian Institute of Technology, Kharagpur

## Persistence Timer

- When TCP receives an acknowledgement with a window size of zero, it starts a persistence timer.
- When the persistence timer goes off, the sending TCP sends a special segment called a *probe*.
- This prevents a deadlock from occurring in the case that an acknowledgment is lost

He will start a persistence timer and when the persistence timer goes off the TCP sends a special segment called a probe. What would have happened is that the buffer may have become full. So now, before sending a packet you have to find out whether it has recovered so you have to send a probe segment.

(Refer Slide Time: 53:23-53:34)



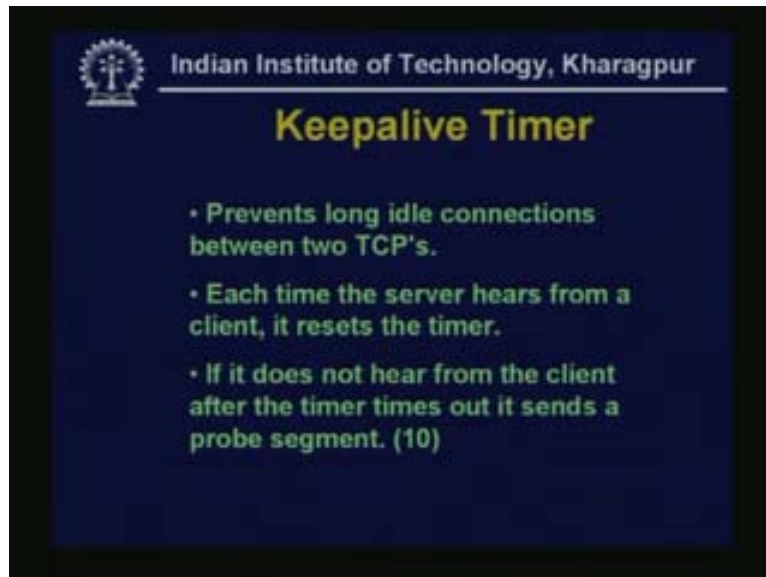
Indian Institute of Technology, Kharagpur

## Persistence Timer

- When TCP receives an acknowledgement with a window size of zero, it starts a persistence timer.
- When the persistence timer goes off, the sending TCP sends a special segment called a *probe*.
- This prevents a deadlock from occurring in the case that an acknowledgment is lost

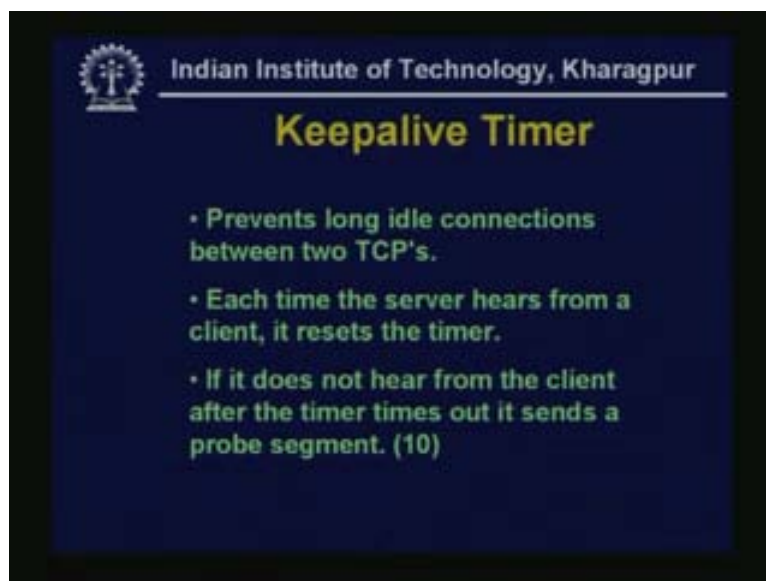
This prevents a deadlock from occurring in case an acknowledgment is lost.

(Refer Slide Time: 54:29-54:51)



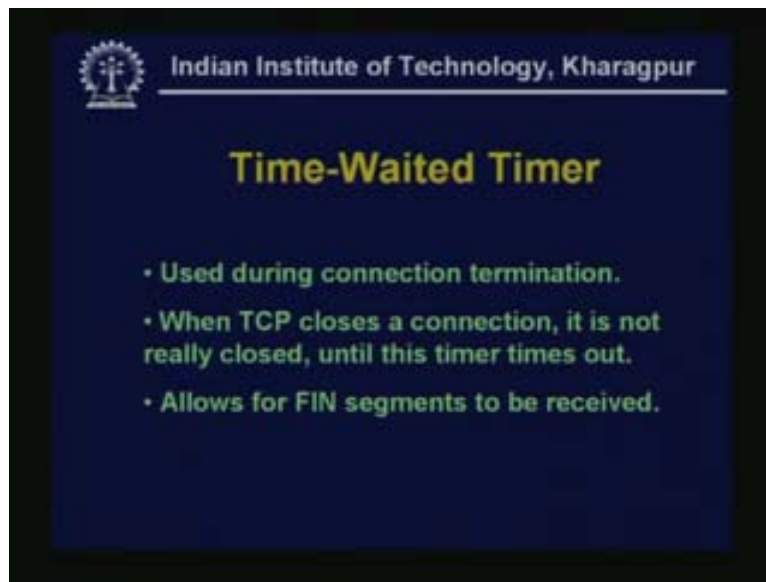
There is a keep-alive timer that prevents long idle connections between two TCP's. Suppose two sides the connection has been setup and nobody has closed the connection but both sides are idle for a long time. Now, as soon as the TCP connection is setup and after the last communication this keep-alive timer is automatically started. When the keep-alive timer goes off, that means there has not been any communication between the two for a long time. What might have happened is that the other side might have got switched off. Therefore at times there can be disruption in connection. Hence it is not a graceful connection.

(Refer Slide Time: 55:22- 55:38)



Each time the server hears from a client it resets the timer and starts the countdown from the beginning. If it does not hear from the client after the timer times out it once again sends a probe segment to see where the other side is alive or not. If the other side is indeed alive and wants to keep the connection on it will reply to the probe segment and then it will know that it still wants to do something but then it is much occupied at the moment and that is why it is not sending. So this is the keep-alive timer. If on the other hand it has been switched off then this probe will go unanswered. So after sometime this side will take some action to close everything.

(Refer Slide Time: 55:22- 55:38)



Time-Waited Timer is used during connection termination. When TCP closes a connection it is not really closed until this timer times out. It allows for FIN segments to be received. In the previous case the last acknowledgment has not been received because it was lost. So everything was fine but the other side has actually gracefully terminated the condition and stopped it. So it has to wait for sometime which is more than twice the round trip delay. Therefore it waits for the time-waited timer and once that is over then the connection can be terminated. With this we close our discussion on TCP and there is some more variance of TCP regarding how you control the window sizes etc. We will discuss that when we discuss congestion control.