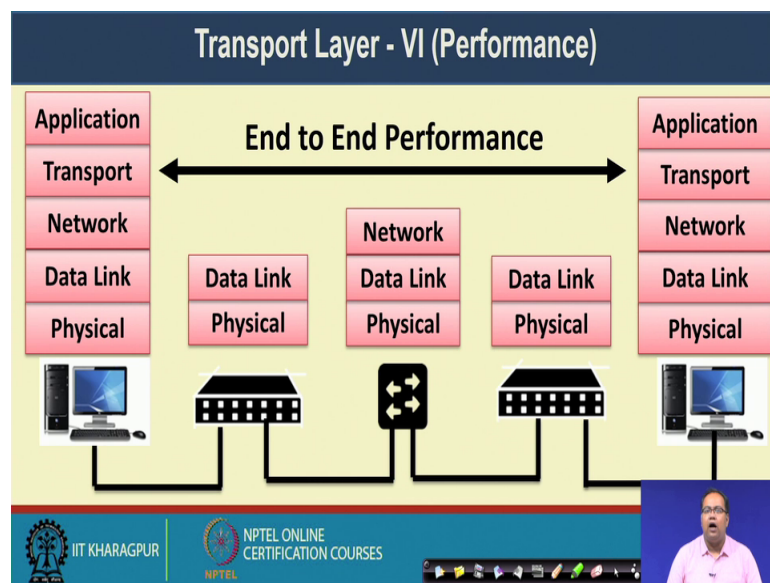


Computer Networks and Internet Protocol
Prof. Sandip Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 16
Transport Layer Performance

Welcome back to the course on Computer Network and Internet Protocols. So, in the last class we have looked into the two different services of transport layer. So, we have looked into connection establishment in details and then, the flow control and reliability support protocols like this ARQ protocols in details. We have looked into three variants of ARQ protocol; the stop and wait protocol and two different sliding window ARQ protocol go back in and selective repeat.

(Refer Slide Time: 00:54)

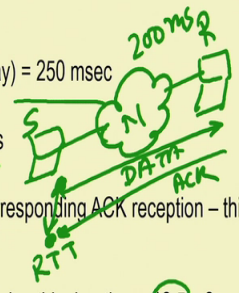


So now, we look into certain performance issues in transport layer and along with that we will look into that how we can improve the end to end performance of a protocol or during the data transmission. So, here our brought objective would be look into the details of transport layer protocol performance and how you can improve it by incorporated incorporating certain additional features over the transport layer.

(Refer Slide Time: 01:20)

Bandwidth Delay Product

- **Bandwidth Delay Product (BDP) = Link Bandwidth x Link Delay** – an important metric for flow control
- Consider Bandwidth = 50 Kbps, one way transit time (delay) = 250 msec
 - BDP 12.5 Kbit
 - Assume 1000 bit segment size; BDP = 12.5 segments
- Consider the event of a segment transmission and the corresponding ACK reception – this takes a round trip time (RTT) – twice the one way latency.
- Maximum number of segments that can be outstanding during this duration = $12.5 \times 2 = 25$ segments



The diagram illustrates the Bandwidth Delay Product (BDP) concept. It shows a network topology with a source, a router, and a destination. A data packet is sent from the source to the router, and an acknowledgment (ACK) is sent back from the router to the source. The round trip time (RTT) is indicated as the time between the packet being sent and the ACK being received. Handwritten notes include '200ms' near the router, 'DATA' on the forward path, and 'ACK' on the reverse path. The BDP is calculated as 12.5 Kbit, and the maximum number of outstanding segments is 25.

So, the first things that we are going to look into in details is a parameter which has significant impact over the transport layer protocol. And based on that you can modify the protocol parameters or you can tune the protocol parameters or some time it will also help you to choose that which variant of protocol you should use in a practical purpose. So, for example, by this time you already know three different variants of ARQ protocols; the sliding window ARQ. The two different sliding windows ARQ go back NARQ and selective repeat ARQ and along with that stop and wait ARQ.

Now, if I ask you the question that for a typical network like said some network parameter is given that the network has this capacity, this end to end capacity, this is the amount of delay for transmitting a packet. The question comes that can you tell a feasible choice of which particular sliding window type of protocols or whether you are going to use this stop and wait ARQ protocol. So, which particular ARQ protocol you are going to use for your network.

So, for that, one important parameter that will help us in this decision making is something called a bandwidth delay product. So, first we look into the details of bandwidth delay product and its implication over the selection of window size for sliding window protocol as well as the choice of choice of particular protocol in protocol designing.

So, let us look into the concept of bandwidth delay product. So, as the name suggests that bandwidth delay product is product of link bandwidth and the link delay; if you consider and, end to end link. So, if your end to end link has a bandwidth of say 50 Kbps and it has one way transit delay is 250 millisecond. Then your BDP is 50 kilo bit per second into 250 millisecond comes to be something similar to 12.5 kilo bit.

So, if you just think about TCP or transport layer segment point of view, so segment is the transport layer data that you want to transmit. So if you think about that you have a 1000 bit segment size, so, this BDP bandwidth delay product comes to be something like to 12.5 segments. So, that is the definition of bandwidth delay product.

So, let us look into that how this bandwidth delay product has an impact over your protocol performance or your design choice of a particular transport layer protocol. So, you consider the event of a segment transmission and the corresponding acknowledgement reception. So, this entire thing takes a round trip time. So, what is a round trip time?

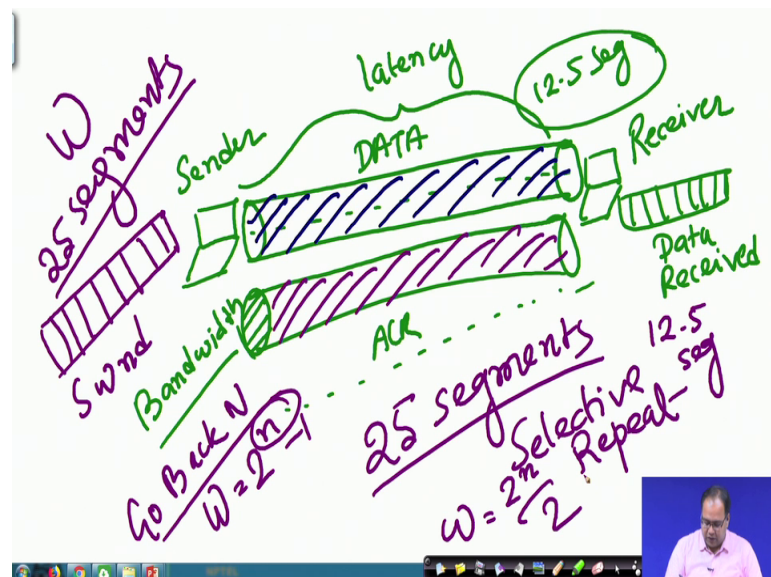
So, you have a sender here, you have a receiver here; say this is your sender, this is your receiver and in between you have the network. The sender and receiver is connected. So, you transmit a data frame and you get the corresponding acknowledgement. So, have transmitted a data and you have got an acknowledgement. So, this total time like the moment you have transmitted the data from there the moment that you have received an acknowledgement, if you find out this timing difference; this will give you a round trip time or an RTT.

So, a RTT is basically twice the one way latency. So, because you can see that it will take; if your one way latency is something similar to say 200 millisecond, then it will take 200 millisecond to transfer this data frame and another 200 millisecond to get back the acknowledgement frame. If you are thinking of that there is no congestion in the network or there is no kind of uninterrupted delay or unintended delay which is their in the network. If you just think about your network is behaving very smoothly, there is no such intermediate delay components which will increase your delay end to end delay. And your delay the total end to end delay can be approximated with the propagation delay. Then with that particular concept you can think of that well this RTT will give you an approximate time that what will be your end to end delay of transmitting a packet

because you are sending the data you are getting the acknowledgement. You are measuring the time difference between them and from there you can make an estimation of the RTT. So, this RTT becomes twice the one way latency.

Now, if you just think about an end to end link. So, the maximum number of segments that can be outstanding during this duration is equal to 12.5 that was your bandwidth delay product. The one way bandwidth delay product, then with multiplied by one way latency into your if you think about it two ways equal to 25 segments. So, the 25 segments can be outstanding. Why this is so?

(Refer Slide Time: 06:25)



So, let us look into one example that if you just think about sender; so, this is your sender and the other end you have the receiver. You can just think of this entire thing, this entire logical channel in between has a pipe well. Now this particular pipe, so whenever you are thinking about this two way communication that you are sending data through one pipe and you are receiving acknowledgement through another pipe well. So, this is the pipe for sending the data and this is the pipe for getting the acknowledgement.

So, the total number of request that can be outstanding within this two pipe is like the total amount of bits that you can push in this two pipe. Now this latency is that how much time will it take to transfer a bit from this sender to this receiver. So, if you just think about the latency; so, this latency denotes the length of the pipe. On the other hand if you just think about the bandwidth, bandwidth gives the area of their, this particular

circle; that means, what is the cross section area of this pipe. So, that is signifies by the bandwidth.

Now if you multiply bandwidth with latency, you can think of it as the amount of data that can be there inside this pipe. Now because if you have two way communication; in one way you have the data and another way you have the acknowledgement. So, by a principle of this sliding window protocol, the acknowledgement will filled up this pipe where as the data, will filled up this pipe, and that way the acknowledgement which are filling up this pipe, the data will for those particular acknowledgement will be stored inside the receiver buffer because receiver has received this data. So, this buffer will contain the data that has been received. So, the receiver has this amount of data and the receiver has started sending the acknowledgement. So, this acknowledgement if filling up this pipe and at the same time the sender has sending the data that data is filling a this data pipe.

So, that way if your bandwidth delay product is according to the earlier example is 12.5 segment. So, you can have twelve point segments of data which is filling up this pipe say this data is filling up this pipe. And another 12.5; say another 12.5 segments of data which are being there in this buffer and the corresponding acknowledgement is filling up this second pipe. So, that way total 25 segments of data can be there which is outstanding, so which is there in the link as well as which is there in this receiver buffer.

So, that way you can say that your maximum window size, the sender window size which can be there. So, if you think about this as the sender window, I am writing it as a send. So, the sender window is the maximum amount of segments that can be outstanding in the network and for that you can wait without getting an acknowledgement. So, if you make the sender window size equal to 25 segments, then you can be sure that well whatever data you are pushing in the network that data will use the entire capacity of the channel. And that way you will be able to sure that well it will have, it will provide you maximum utilization of that end to end channel, the maximum utilization of the pipe which is there in between the sender and the receiver.

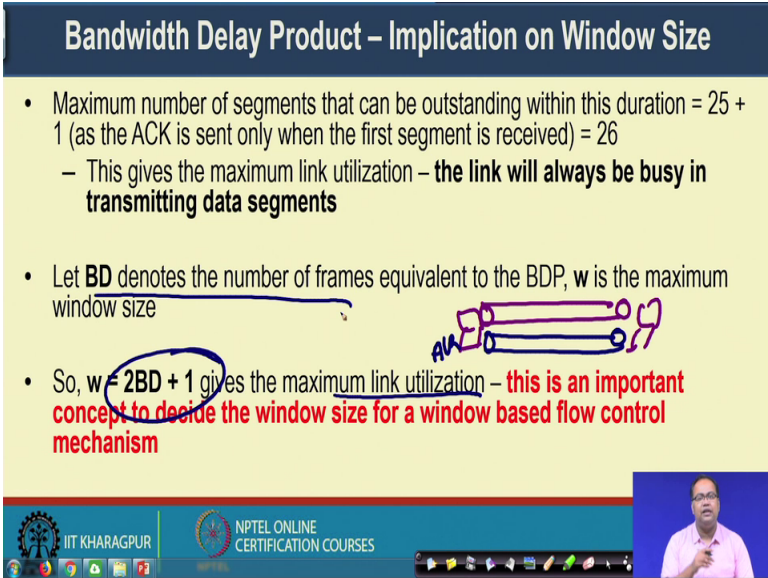
So, this gives a theoretical bound the maximum bound on this on this window size, the sender window size will which will provide you the maximum capacity. Now just relating it with the sequence number that we have discussed earlier, the relation between

the window size and the sequence number. So, once you choose the window size in this way say you have chosen the window size w in this way. Now assume that you are using a go back N ARQ protocol, if you are using a go back N ARQ protocol and you know that in that case, your maximum window size can be $2^n - 1$. So, from there you can find out that what should be what should be your sequence number space. So, how many bits you should reserve for the sequence number such that you can have the expected window size and at the same time that window size will fill up the end to end capacity of the network.

Similarly if you are using selective repeat ARQ for a selective repeat ARQ, you know that w is equal to $2^n / 2$. So, you can you can select the window size in you can select the sequence number in such a way. So, that this particular relation holds.

So, that way am you can you can find out the maximum window size which will provide you the maximum utilization of the channel. And accordingly you can set up the sequence number space for different flow control algorithm.

(Refer Slide Time: 12:16)



Bandwidth Delay Product – Implication on Window Size

- Maximum number of segments that can be outstanding within this duration = $25 + 1$ (as the ACK is sent only when the first segment is received) = 26
 - This gives the maximum link utilization – **the link will always be busy in transmitting data segments**
- Let **BD** denotes the number of frames equivalent to the BDP, w is the maximum window size
- So, $w = 2BD + 1$ gives the maximum link utilization – **this is an important concept to decide the window size for a window based flow control mechanism**

The slide includes a diagram showing a sequence of frames being transmitted over a link, with an acknowledgment (ACK) being received. The diagram illustrates the relationship between the bandwidth delay product (BDP) and the window size.

NPTEL ONLINE CERTIFICATION COURSES

So, now the thing is that like this. This is the description that I have given like ideally what we can think of that the maximum number of segments that can be outstanding within this duration is these 25 segments which is equal to the channel capacity plus 1.

So, this plus 1 is like that the acknowledgement for one frame which has just received by the sender. So, the sender has not processed it yet.

It is just received by the sender. So, that is why we adopt this 1 here which gives you the maximum link utilization. So, it is just like that you have filled up this entire two pipes and one acknowledgement has just received at the sender. So, you have filled up the 2 pipes which are there in between and so, one data pipe and one acknowledgement pipe and one acknowledgement has just received at the sender. So, that way it comes equal to $2BD$ plus 1.

So, the window size equal to $2BD$ plus 1, it will give you the maximum link utilization where BD denotes the number of frames equivalent to BDP. So, this is an important concept to decide the window size for a window based flow control algorithm; so the example that I have given you earlier.

(Refer Slide Time: 13:43)

Implication of BDP on Protocol Design Choice

- Consider the link bandwidth = 1Mbps, Delay = 1ms
- Consider a network, where segment size is 1 KB (1024 bytes)
- Which protocol is better for flow control?
 - (a) stop and wait,
 - (b) Go back N,
 - (c) Selective Repeat
- $BDP = 1 \text{ Mbps} \times 1 \text{ ms} = 1 \text{ Kb} (1024 \text{ bits})$
- The segment size is eight times larger than the BDP -> the link can not hold an entire segment completely
- Sliding window protocols do not improve performance
- **Stop and Wait is better – less complexity**

Handwritten annotations on the slide include a circle around '1 KB (1024 bytes)', a circle around '1 Kb (1024 bits)', and a hand-drawn diagram of a pipe labeled 'DATA+ACK'.

NPTEL ONLINE CERTIFICATION COURSES

So, let us see an example of this. So, consider a network with link bandwidth end to end link bandwidth as 1 Mbps, the delay equal to 1 millisecond and you consider a network where as segment size is 1 kilo byte equal to 1024 bytes. Now the question is that which particular protocol would be better for flow control whether you are going to use a stop and wait protocol or a go back n protocol and selective repeat protocol. So, for to solve this problem, so, we first compute the BDP; we see that the BDP comes to be 1 Milli

byte 1 Mbps into 1 millisecond equal to 1 kilobyte; that means 1024 byte. So, the segment size is eight times larger than the BDP.

So, here am here your BDP is 1 kilo bit and your segment size is 1 kilobyte because your segment size is one kilobyte; that means, the link cannot hold an enter segment completely. So, the pipe that you are considering here between the sender and the receiver; so here this pipe assume that this pipe considers both the data and the acknowledgement data plus ACK pipe. So, this ACK pipe this data plus ACK pipe it will not be able to hold this entire segment inside that because BDP comes to be 1 kilo bit where as your segment size is 1 kilo byte.

Now in this case, the sliding window protocols do not improve performance because why because we will not be able to send multiple segments in parallel even one segment is not able to fill up the your pipe completely; because one segment is not able to fill up your pipe completely. There is no reason to send multiple segments in parallel because any way you will not be able to get the advantage of parallelization in this particular case where your link bandwidth is 1 mega bit per second and delay is 1 millisecond.

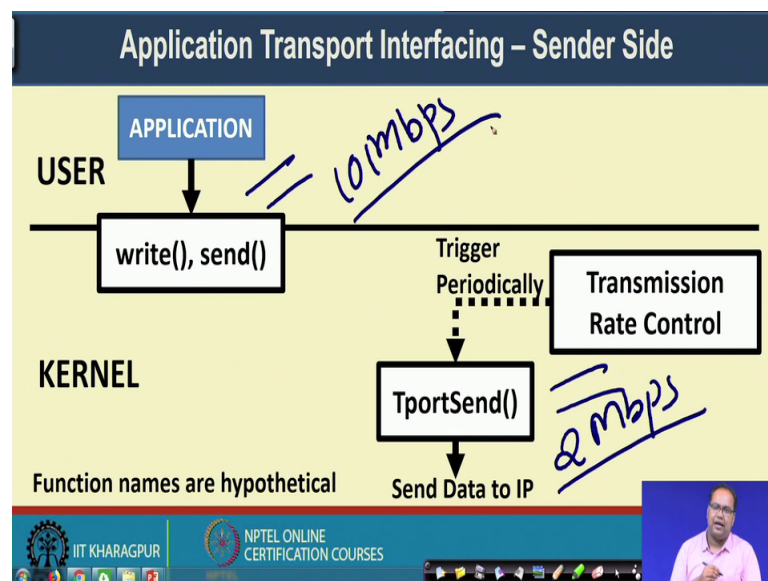
So, under this particular case it is always good to choose a stop and wait protocol because stop and wait protocol has the least complexity. So, sliding window protocol as you understand, because of the design choice, it has more complexity, you have to maintain the sender window, you have to maintain the receiver window. Then you have to maintain the sequence number fill; all this different overheads are there. But with a stop and wait protocol, the logic is pretty simple that you send segment and then wait for acknowledgement once you are getting acknowledgement; you send the next segment.

So, that way your stop and wait protocol will have significantly more sorry significantly less overhead compared to a sliding window protocol. And because here, we see that we are not getting the advantage of parallelization, we always prefer to use a stop and wait protocol under this particular example scenario.

So, this gives you an intuition or an example that how this parameter BDP bandwidth delay product help you to make a design choice that which particular sliding window protocol, you should use to improve the network performance with having minimal complexity. And the same time the example that I have given you earlier that bandwidth delay product will help you to choose the optimal window size that what window size

you should use such that you can utilize the maximum capacity of the network. And once you have selected that window size and your happy with a sliding window protocol based on this philosophy, you can find out that what sequence numbers space you should use such that there is no am no confusion in the protocol design during the execution of the protocol. Like the examples that we have looked earlier in the case of sliding window protocols; different variance of sliding window protocols like the go back N protocol or the selective repeat protocol well.

(Refer Slide Time: 17:32)



So, from here let us look into that how we basically interface the application layer with the transport layer at the sender side. This will give a design idea that how will be able to design a transport layer protocol. So, the example that I have taken is from the line of separating system. So, you have the user space and the kernel space at the user space you are running certain application that is sending the data. So, you have certain system call at the kernel the right system call and the same system call will look into all this system calls later on whenever we discuss about the socket programming. So, there are this system calls through which you can send the data to the kernel for the application.

Now, here you have this transmission rate control module. This transmission rate control module based on your flow control algorithm, it will use this function. So, this name of this function are hypothetical not directly matches to it what is implemented in the

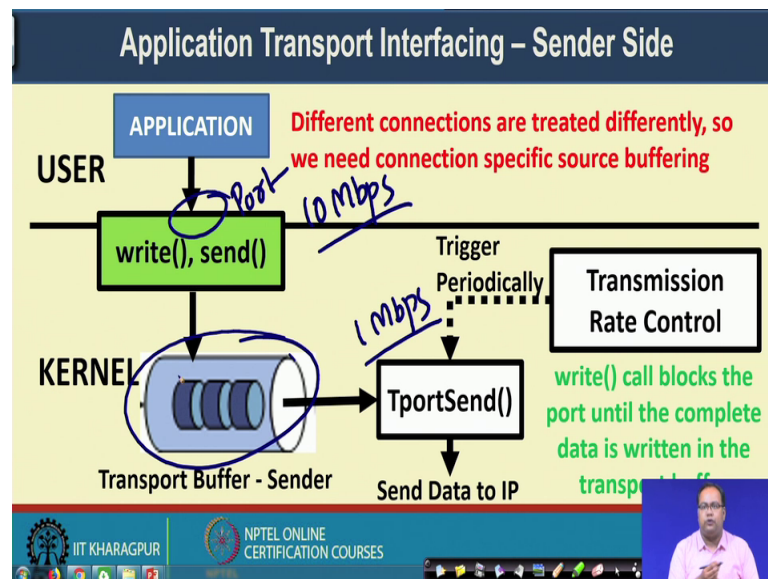
Linux, just to give you an idea about how you can implement your own transport protocol.

So, you have a function called `TportSend`, it is triggered periodically based on your transmission rate control your based on your flow control algorithm; based on your window size that how much data you can send. So, this particular function is being called and the data is sent to IP. The next layer of the protocol stack the network layer of the protocol stack.

Now you can think of that this rate and this rate are asynchronous. So, here the application can generate data at a more high higher rate compare to the rate at which the transport layer can send the data to the other end. So, this transmission rate control may find out that well the optimal rate is something equal to 2 Mbps whereas the application generates rate at a 10 Mbps.

Now if this is the case, the application generates rate at 10 Mbps and the transmission rate control generates rate of 2 Mbps. Obviously, you need to have intermediate buffer which will store that data.

(Refer Slide Time: 19:42)



So, at whatever rate the application is generating the data, some time it may be higher than the rate at which this transmission rate control module works. So, this application, it will write the data in the buffer and then this `TportSend` function. It will pick up the data

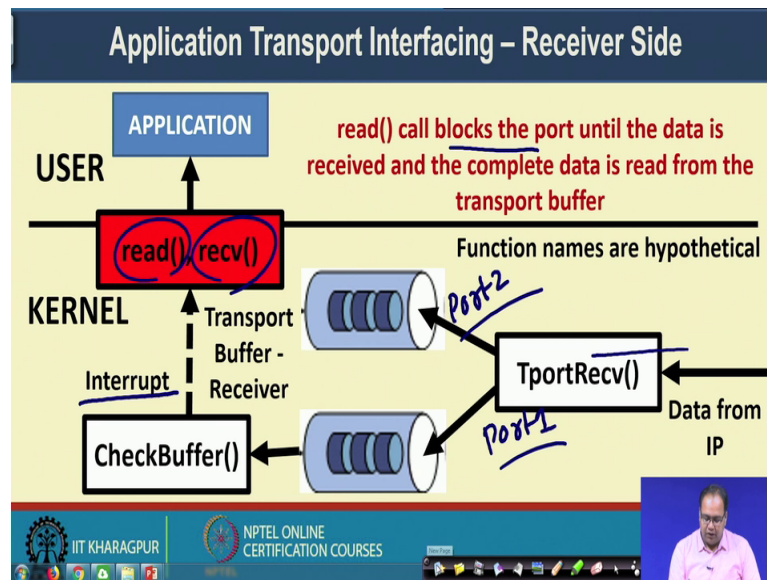
from the buffer based on the rate that is being provided by the transmission rate control module and the data will be sent to the next layer of the protocol stack.

Now, in this case it may happen that different connections, you may have different connections at the application layer. They are treated differently. So, we need connection specific source buffering. So, this particular buffer we call it as a source buffer. So, for every independent connection that you have from the application layer, we have one transport layer buffer associated with it. And then there is another interesting fact about this write call, the write call through which you are sending data from the application. This write call blocks the port. So, here is your port through which you are uniquely identifying an application. So, it blocks the port until the complete data is written in the transport buffer.

So, it is just like that it may happen that well some time your transmission rate control is sending data at a rate of 1 Mbps and application is generating data at a rate of 10 Mbps, the example that I have given earlier. So, application is sending data at a more higher rate compare to what the transmission rate control is sending the data to the lower layer of the protocol stack.

So, after some time; obviously, this buffer has a finite space. So, the buffer will get filled up. Once the buffer gets filled up, so then the transport layer it blocks the application to write any more data in that buffer to avoid the buffer overflow from this transport layer buffer.

(Refer Slide Time: 21:39)



Now, let us look in to the receiver side. So, the receiver side the idea is again similar. So, your TportRecv, the transport receives function; it will receive the data from the network layer. So, once it has receive the data from the network layer, it will look into the port number in the transport layer header. So, by looking into the port number, it will decide that in which application which transport layer q it should fill it. So, this transport layer q is for one application, this transport layer q is another application.

So, every such q is bounded to it one port because as you have seen earlier that this port number it uniquely identifies an application. So, based on that, you put the data in the buffer. Now from the application side you make a read call or a receive call which you through which you will read the data from this buffer. And here the method is something like that whenever you are making a receive call; it will wait on this CheckBuffer function. It may happen that whenever the application is making a receive call during that time, this received buffer is empty it has not received any data. So, the call will be getting blocked here. And the movement you receive the data in this buffer, it will send the interrupt signal to this call and this call will get the data from this buffer and send it to the application.

So, with the help of this interrupt, we can make this receive call to interact with this buffer. Now here you can that this receive call, it is a blocking call the read call or the

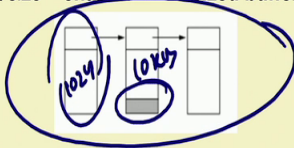
receive call; it is a blocking call until the data is received, then the complete data is read from the transport buffer.

So, it is like that whenever you have made a receive call, during that time the call is getting blocked at this port until you are getting a data in this buffer. And once you are getting a data in this buffer, then use a this check buffer function. It will send interrupt to the read call and the receive call and it will get this entire data from the buffer and release this particular call.


So, that way you can see that both of this calls are kind of blocking call at the sender side as well as the receiver side. So, the sender call gets blocked when the buffer is full, the receiver call gets blocked when the buffer is empty.

(Refer Slide Time: 24:04)

Organizing Transport Buffer Pool

- If most segments are nearly the same size, organize the buffer as a pool of identically sized buffers (one segment per buffer)
- For variable segment size – **chained fixed sized buffer** (buffer size = maximum segment size)

- Space would be wasted if segment sizes are widely varied
- Small buffer size – multiple buffers to store a single segment – added complexity in implementation

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES



So, the question comes that how can you organize this buffer pool. So, there are multiple ways you can organize the transport layer buffer. It is a software buffer. So, in case your segments are of same size. So, all the segments are of same size, you can organize the buffer as a pool of identically size buffer; that means, you can hold one segment at every individual buffer. So, a segment contains certain number of bytes. So, your individual buffer size will be equal to your segment size and every individual buffer contains one segment and this buffer pool they can contain multiple segments all together. Now for variable segment size you can use this chained fixed size buffer. So, it just like a linked list.

So, your individual buffer size is the maximum segment size and say individual buffers are connected by a link list kind of data structure and they entirely construct the buffer pool for a particular transport layer port number corresponds to an application. Now in this case, if you are using chained fixed size buffer the space would be wasted if segment sizes are widely varied.

So, if one segment is 1024 kb, another segment is 10 kb in that case, you can have a significant amount of free space here which is being wasted. Now if you make a small buffer size, then you need multiple buffers to store a single segment which adds the complexity in the implementation.

(Refer Slide Time: 25:41)

Organizing Transport Buffer Pool

- **Variable size buffers (b)**
 - Advantage: better memory utilization
 - Disadvantage: Complicated implementation
- Single large **circular buffer** for every connection (c)
 - Good use of memory only when connections are heavily loaded

Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, in this case we use the variable size buffers. So, with the variable size buffer, so here is an example of a variable size buffer. So, you have the pool of buffers which are connected wire linked list data structure and they will have variable size. The advantage is that you can have a better memory utilization, you can reduce the amount of loss or amount of memory space wastage. If you have a large data large segment, you put it in the large buffer; if you have a small segment, you put it in a small buffer. But the disadvantage is you have a complicated implementation because individual buffer spaces dynamic.

So, you have to dynamically allocate the memory. The third solution that we prefer is to use single large circular buffer for every connection. So, we have a kind of circular buffer and that circular buffer contains individual segment. So, in a circular buffer you can have one segment of say 1 kb size, another segment of having 4 kb size, another segment of having 10 bit size and that way. So, you can put individual segments of different sizes one after another and ultimately you can have a unused space which can be used to store the next incoming segments.

So, this single large size circular buffer, it provides a good use of memory when the connections are heavily loaded. Because if the connections are heavily loaded again, you are going to use or going to waste huge amount of space inside the buffer because you are using a fix size buffer, then in that case the variable size buffer may perform well.

So, that way your choice of designing a transport layer buffer depends on what type of applications, you are going to use and what type of data the applications are going to generate; so based on that you can decide that which particular buffer will be more suitable for your application. So, this gives you this particular lecture give you a broad idea about your design choice of multiple transport layer parameters and how it impacts the transport layer protocol performance. And we have looked into a hypothetical implementation of different transport layer functions and how the transport layer calls are getting interfaced with the application layer and in that particular scenario what type of transport layer buffers you can use based on your need of the application.

So, in the next class we will continue from here and look into one another service at transport layer like the congestion control mechanism. And then, we will go to the details of the transport layer protocol implementation. We will talk about the TCP and UDP protocol implementation in details.

Thank you all for attending this class.