Electrical Engineering Department

Broadband Networks

Prof. Karandikar

Indian Institute of Technology, Bombay

Lecture - 17

TCP'S Congestion Control

So, we were discussing how congestion control can provide some kind of quality of service guarantees for the best effort traffic. So, the congestion control mechanisms operates by limiting the amount of data that can be transmitted or that can be injected into the networks and also tries to make sure that if different sources are using these congestion control mechanisms, then this mechanism is fair in some sense. And, in previous lectures we had discussed about TCP'S congestion control strategy and we had seen that TCP adopts additive increase and multiplicative decrease kind of congestion control mechanisms.

Now, the TCP'S congestion control mechanism is a window based congestion control mechanism. By window based we mean that the TCP limits the maximum amount of unacknowledged data that can be there in the transit and this maximum number of unacknowledged data that can be there in the transit is called the window size. So, if there is congestion on the networks and the TCP will reduce the window size and when there is less congestion on the network; the TCP opens up the window size. However, this increase or decrease of window size is done in an additive increase and multiplicative decrease fashions.

So, that means that TCP increases its window size in an additive fashion and whenever it detects the congestion, it suddenly drops its window size. So, the increase is something like window size is equal to window size plus constant and the decrease is window size, the new window size is equal to the old window size divide by a constant. That is a multiplicative decrease.

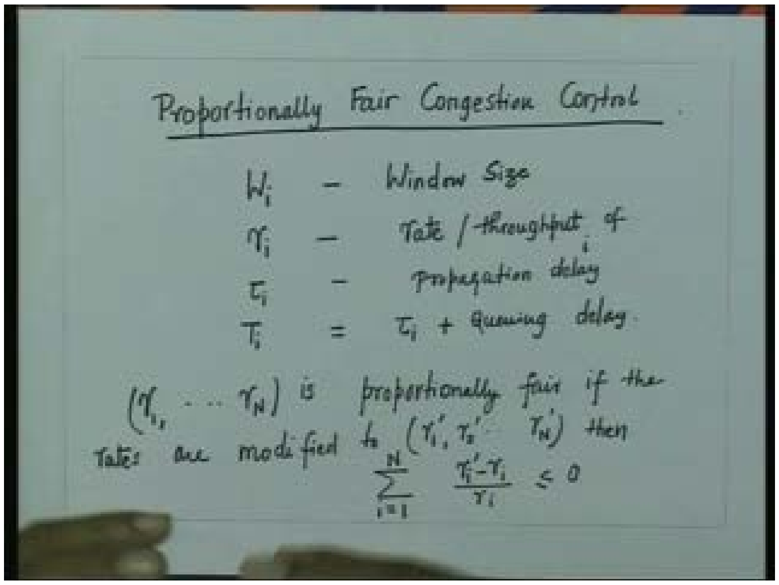
Why additive increase and multiplicative decrease? It has been found that additive increase and multiplicative decrease leads to a stable system in the networks. So, the TCP'S congestion control mechanism also leads to some kind of fairness among the different sources. So, as we are also seeing that the additive increase and multiplicative decrease - a kind of phenomenon leads to a stable operating point where different sources which have the same maximum packet size or maximum segment size and are situated at the same round trip times will lead to will lead to a point where not only the link utilization is maximum but different sources are experiencing the similar throughputs.

So, in that sense, the TCP'S congestion control mechanism does try to be somewhat fair. Of course, TCP has unfairness in the sense that if different sources are situated at different round trip times or if they have a different maximum segment size, then the TCP'S congestion control mechanisms may be unfair.

So, in that context, it is the question to be asked that how do we design an optimal congestion control mechanism that is somewhat fair; this question has been asked. So before, we discussed TCP'S congestion control mechanism in and its analysis in greater detail; we first try to ask this question that what could be a fair congestion control mechanism in the networks that will try to lead to that stable and a fair operating point.

So, let us define an optimal congestion control mechanisms. Even though it is known that the TCP has been operating in the networks for such a long period of time, even though it may not be an optimal congestion control mechanisms still. Since it is the mechanism that has been widely deployed in practice, it may be difficult to replace the TCP'S congestion control mechanisms with some other congestion control mechanism. But still we will try to see that how can we define an optimal congestion control mechanisms. So, we define a proportionally fair congestion control mechanisms.

(Refer Slide Time: 6:02)



Proportionally Fair Congestion Control

W_i - Window Size

r_i - rate / throughput of i

τ_i - Propagation delay

$T_i = \tau_i +$ Queuing delay

$(r_1, \dots r_N)$ is proportionally fair if the rates are modified to $(r_1', r_2' \dots r_N')$ then

$$\sum_{i=1}^{N} \frac{r_i'-r_i}{r_i} \leq 0$$

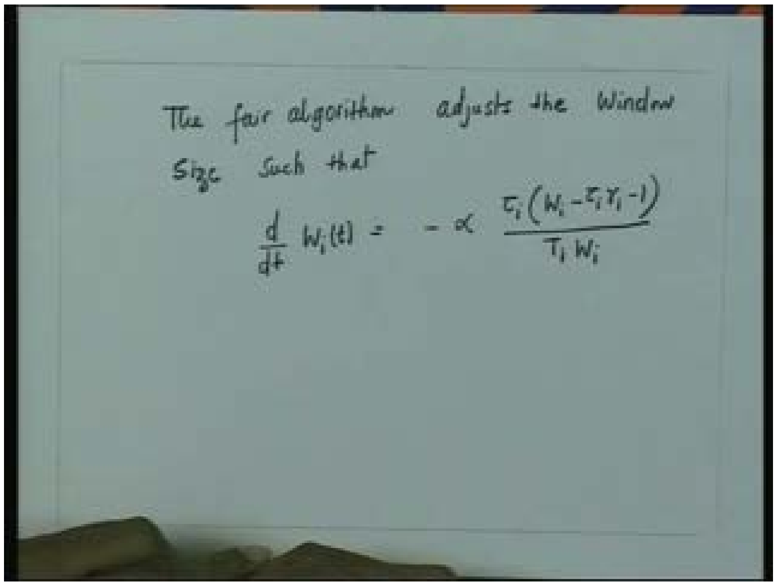
So, let me just define a proportionally fair proportionally fair congestion control. So we said, let us say that W_i is the window size of the source i, r_i is the rate or the throughput of the source i and tau $_i$ is the propagation delay and T_i is equal to tau $_i$ plus queuing delay.

So, we define that these r_1, r_2, r_N - these are the throughputs which are observed by different sources, the source is indexed by i. So, we say that this vector r_1 to r_N is proportionally fair proportionally fair that if the rates are modified if the rates are modified to, if these rates r_1 to r_N if they are modified to r_1 prime r_2 prime so on to r_N prime; then the following conditions that is the summation of i is equal to 1 to N, r_i prime minus r_i upon r_i - this is less than or equal to 0.

So, that means if you change the rate then that result in a negative increase of their relative rates. If the sum, if you change the rates, then it results in a negative increase of the sum of their relative increases. So, we call such an algorithm to be a proportionally fair algorithm.

Now, this problem can be formulated and it can be shown that the such an algorithm which is proportionally fair in the following sense; that means if the throughput realized by the sources if they get modified to another operating point of r_1 prime to r_N primes, then this will result in a negative increase of the some of their relative increases. So therefore, this we call this algorithm to be proportionally fair.

(Refer Slide Time: 9:29)



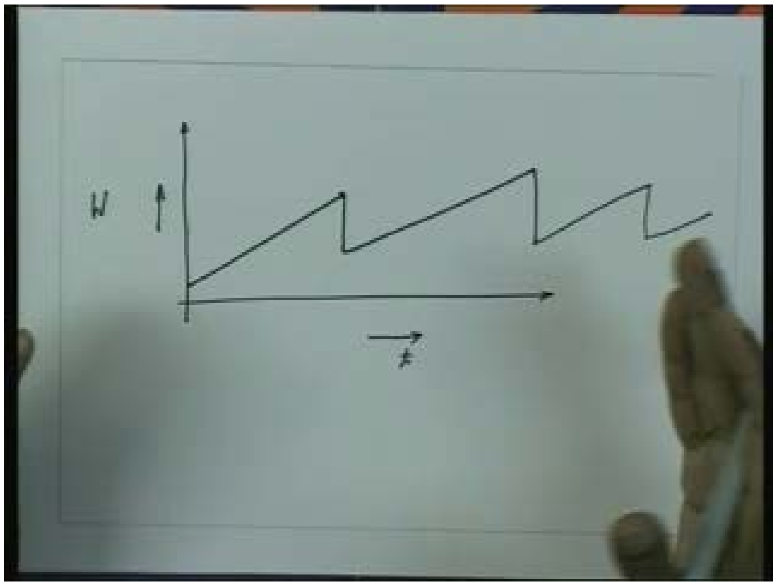
The fair algorithm adjusts the window size such that

$$\frac{d}{dt} W_i(t) = -\alpha \frac{\tau_i (W_i - \tau_i r_i - 1)}{T_i W_i}$$

The proportionally fair algorithms actually, it can be shown that this fair algorithm will adjust the window size, adjusts the window size in such a manner that d by dt of W_i (t) that is equal to some minus alpha times tau $_i$ W_i minus tau i r_i minus 1 upon T_i into W_i. So, this is the solution, this is the solution of an algorithm which is optimal congestion control algorithms in the sense of being proportionally fair. So, the solution will lead to an algorithm which will adjust the window size in such a manner that d by dt of this W_i by t is given by this expression.

So we leave at this point, we do not discuss this issue further more because as we have seen that the TCP'S congestion control mechanism is the dominant congestion control mechanism deployed internet for the control of best efforts networks and therefore we would like to see the performance of analysis of the TCP'S congestion control mechanisms in somewhat more detail.

(Refer Slide Time: 11:09)



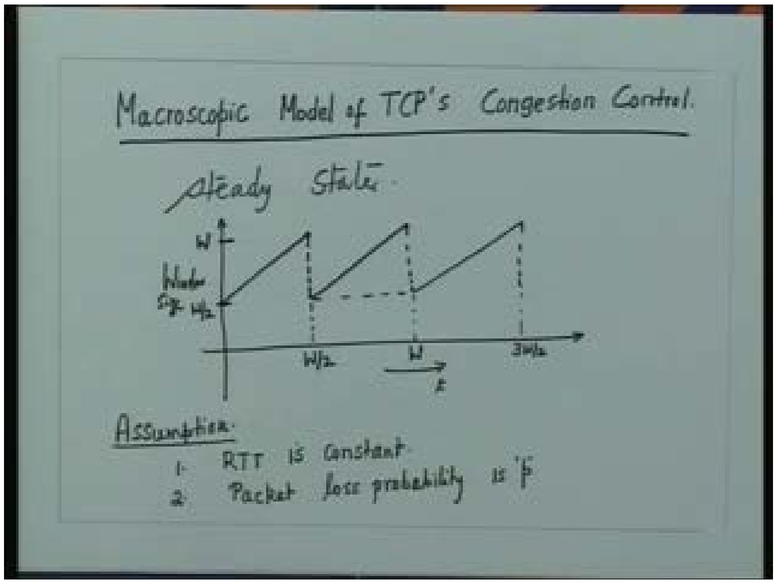
So, let us recapitulate how the TCP'S congestion control mechanism operates and the TCP'S congestion control mechanism, as we had seen follows an additive increase and the multiplicative decrease. So, here is the window size W and here is the time t. So, the window size increases from 1 and whenever it detects a packet loss, it drops its window size to half. Then again it increases its window size and if it detects the packet loss, then it drops its window size to half and so on. So, this way the evaluation of the window size as we had seen follows a saw tooth behavior in the TCP'S congestion control mechanism.

So, let us try to analysis the behavior of such a TCP'S congestion control mechanism. Now, one thing that needs to be noted is that the TCP is increasing its window size in a liner fashion and whenever it detects a packet loss, it drops its window size. So, how does it detect a packet loss? It detects a packet loss, we are assuming that the sender does not receive an acknowledgment within a certain time out interval and we call it to be a retransmission timer interval. So, if the sender does not receive an acknowledgement within a certain time out interval; then it will trigger the retransmissions, But at the same time, it perceives that since it has not received an acknowledgement; it must have been, because the packet must have been lost in the network and therefore there must have been a congestion in the network and therefore the TCP drops its window size.

So, that is the why it has this saw tooth behavior. So, at this point here, we are assuming that the source actually detects the packet loss. So, a packet loss has occurred; it detects a packet loss and therefore it drops its window size. So now, we would try to do an analysis of TCP'S congestion control mechanisms and for the analysis purposes assume that a packet loss is a random packet loss, the packet gets lost with certain random probability.

So, with that we would try to do a very rough analysis as we call it to be a macroscopic analysis of TCP'S congestion control mechanisms. We will then go into the detail about deriving expression for the throughput of the TCP'S congestion control mechanisms by using a more sophisticated analysis. But for the time being assume that the packet has been lost in the network due to some random loss and it occurs with and the loss occurs with a probability of p.

(Refer Slide Time: 13:55)



Macroscopic Model of TCP's Congestion Control.

Steady State.

Assumption.

1. RTT is constant
2. Packet loss probability is 'p'

So, let us now first understand the model: a macroscopic model of TCP'S congestion control. Now, what we do to do this analysis of the TCP'S congestion control mechanism? We take a very simplified view and also we assume that this TCP'S congestion control mechanism of the additive increase and the multiplicative decrease has reached a steady state.

So, first of all that is an analysis we do in a steady state, very important and in this steady state therefore what is happening is how the window is changing, let us see. So here, is a window size W and here is a time t. So, what we are saying is this is the change in which it is occurring, so this is window size.

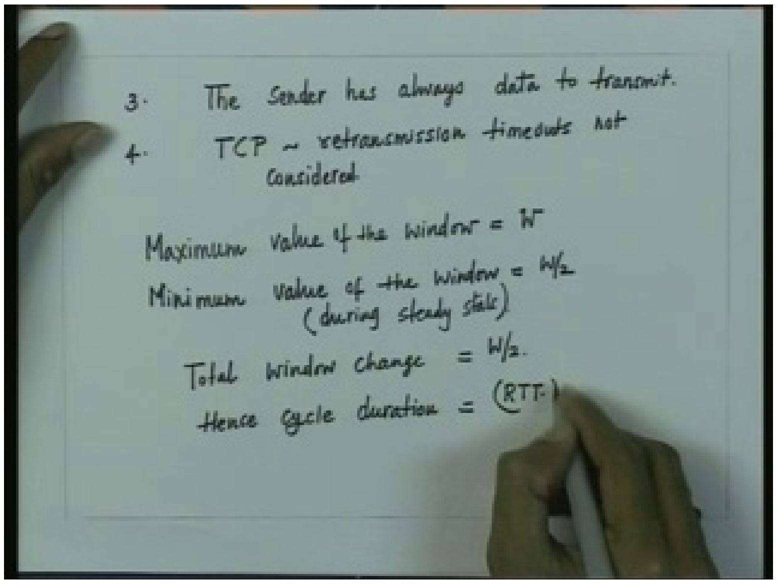
So, let us say that the window size where the packet loss occurs is let us say, W and therefore you start from W by 2 in the steady state. You go up to W, packet gets lost here and therefore it drops the window size to half. Then we again increase the window size and we are assuming that we are in the steady state and therefore the packet will get lost again here at W and so on. Now, so we make certain assumptions here and these assumptions we should write here that we have reached, we are considering a steady state.

Now, one thing that needs to be noted it here is that since the window is increases from W by 2 to W - this much window increase has occurred; so now window will open by 1 packet during every RTT. So, the window size has been of the order of W by 2. So therefore, the time this is of the order of W by 2, this is W and this is 3W by 2.

So, we are assuming that a window opens by 1 during every RTT. Then, W is equal to W plus 1; we are following this equation that W is equal to W plus 1 during every RTT. So, assumption is so we are assuming that the window opens by 1 packet during every RTT. So, assumptions here is that the RTT is constant, we are assuming that RTT is constant and packet loss probability is also constant is p and this is also constant.

So, when we say that a packet loss probability is constant that means the link actually transmits 1 by p successful packets. So, the link actually delivers approximately 1 by p consecutive packets successfully. Then, the second thing is that we are assuming that the sender has always data to transmit.

(Refer Slide Time: 18:32)



3. The sender has always data to transmit.

4. TCP ~ retransmission timeouts not considered

Maximum value of the window = W

Minimum value of the window = W/2 (during steady state)

Total window change = W/2.

Hence cycle duration = (RTT)

So, the third assumption is that the sender has always some data to transmit and we are assuming here that the TCP will avoid the retransmission time outs; it is not due to the retransmission time outs, you are always having a packet loss when the window size becomes W and that packet loss we are modeling it by a probability, by a random packet loss probability of p. So, we are assuming here that there is no…

The question is that how does the sender then comes to know that a packet has been lost if there is a no retransmission time out interval. The answer is that the TCP also determines the packet loss because of the other mechanism like the triple duplicate acknowledgement also. I will explain that later but for the time being assume that somehow the source knows that a packet has been lost and we are assuming that this congestion control mechanism has reached a steady state and therefore the source increases its window size from W by 2 to W in a time period of in a time period of W by 2 and then at W, the packet gets lost with a probability p; it drops its window size to W by 2 and so on.

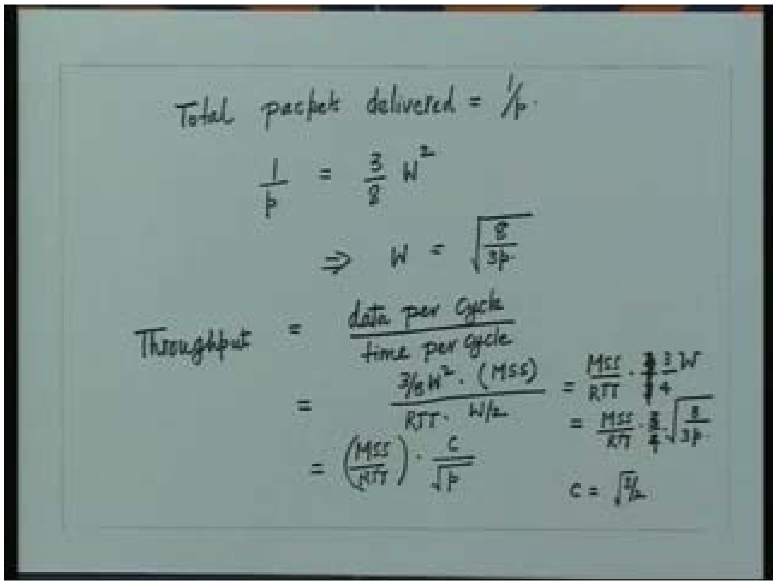
So, we are assuming that the maximum value of the window size is W and the minimum value during steady state is W by 2 and so the total change is, the total window change is, the total change in window is W by 2 and hence the cycle duration will be RTT times W by 2. So, this cycle duration will be RTT times W by 2. So, we measure this time in actually RTT’s.

So, as we know that the window opens by 1 packet every RTT and since the total window change is W by 2; so therefore obviously, the cycle duration is RTT into W by 2 and the question is how much data has been deliver during this cycle.

So, to determine how much data has been delivered we need to determine the area, this much area because the amount of data has been delivered will be determined by the area of this. So therefore, if we have to determine how much is the total data that has been delivered, so the total data which is delivered: so there are 2 parts over here, there is one - this is rectangle over here, this is W by 2 is this duration and W by 2, so this square is W by 2 square and there is a triangle here, so this is W by 2 and this height is W by 2. So, that will be half of W by 2 square.

So, the total is, total data delivered will be W by 2 that is square plus half of W by 2 square and which will be 3 by 8 W square packets per cycle and so this much is the total data that has been transmitted by the source during the particular cycle. So, now the total packets which are delivered; note that the packets are lost with a random probability of p, so therefore the link delivers 1 by p packets, 1 by p consecutive packet.

(Refer Slide Time: 24:18)



Total packet delivered = $1/p$.

$$\frac{1}{p} = \frac{3}{8} W^2$$

$$\Rightarrow W = \sqrt{\frac{8}{3p}}$$

$$\text{Throughput} = \frac{\text{data per cycle}}{\text{time per cycle}}$$

$$= \frac{3/8\, W^2 \cdot (MSS)}{RTT \cdot W/2} = \frac{MSS}{RTT} \cdot \frac{3}{4} W$$

$$= \frac{MSS}{RTT} \cdot \frac{3}{4} \sqrt{\frac{8}{3p}}$$

$$= \left(\frac{MSS}{RTT}\right) \cdot \frac{c}{\sqrt{p}}$$

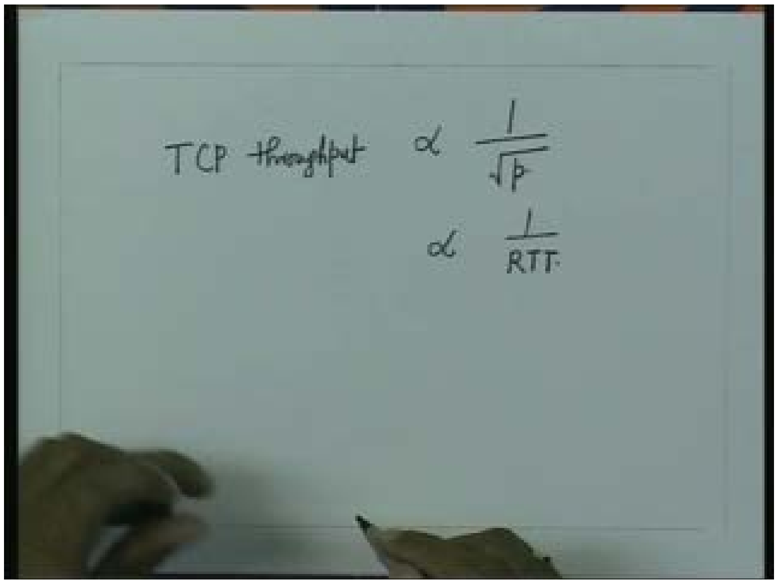
$$c = \sqrt{3/2}$$

So therefore, the total packet delivered is also equal to 1 by p and therefore 1 by p should be equal to 3 by 8 W square and from this we determine that W - maximum window size is equal to 8 by 3 p. And therefore, if I define the throughput to be equal to the data delivered per cycle upon the time per cycle which is equal to the maximum segment size.

So, data per cycle now note that the total data delivered is as we have determined 3 by 8 W square, I will just multiply this to be maximum segment size to know the data delivered in terms of bytes and the time per cycle you have to determine in terms of seconds, then this is RTT time W by 2 which means maximum segment size divide by RTT into this. This we have W, W cancels out and so this is 3 by 8. So, this is actually you can see is some of p maximum segment size, so some constant - c divide by square root of p. So, which is determined from, this W cancels out here. So, we have 3 by 4. So, this is MSS by RTT and since 2 4 by 4 into W and in place of W, I write as 8 by 3p. So, this is sorry this is this should be 3 by 4 and so there should be 3 by 4.

So therefore, the constant c is actually equal to root 3 by 2. So, what we are actually trying to say is that the throughput of the TCP source is inversely propositional to the square root of the packet loss probability.

(Refer Slide Time: 27:39)



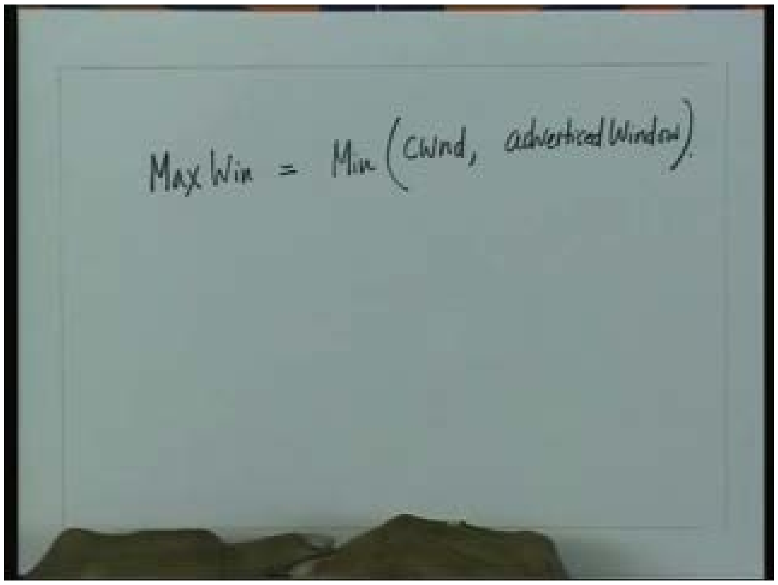
TCP throughput $\propto \frac{1}{\sqrt{p}}$
$\propto \frac{1}{RTT}$

So, that is an important result where we are trying to really say that the TCP throughput is inversely propositional to the square root of the packet loss probability. It is also inversely propositional to the round trip time. So, that means if some sources are situated at a greater distance, then the throughput of that source is likely to be lesser than a source which is situated at a smaller distance from the designation.

So therefore, the TCP throughput it inversely propositional to the round trip time, it is also inversely propositional to the square root of the packet loss probability. Now, at this stage we would like to see; basically, what we have seen is that the TCP increases its window size in a linear fashion and then whenever it detects a packet loss, it drops its window size to W by 2.

Now historically, actually, what happened? Now note, one thing is there that the receiver is also conveying to the source about its advertised window scale. So, the window size which we are talking of in the congestion control mechanism is called as a congestion window size.

(Refer Slide Time : 29:16)


Max Win = Min (CWnd, advertised Window)

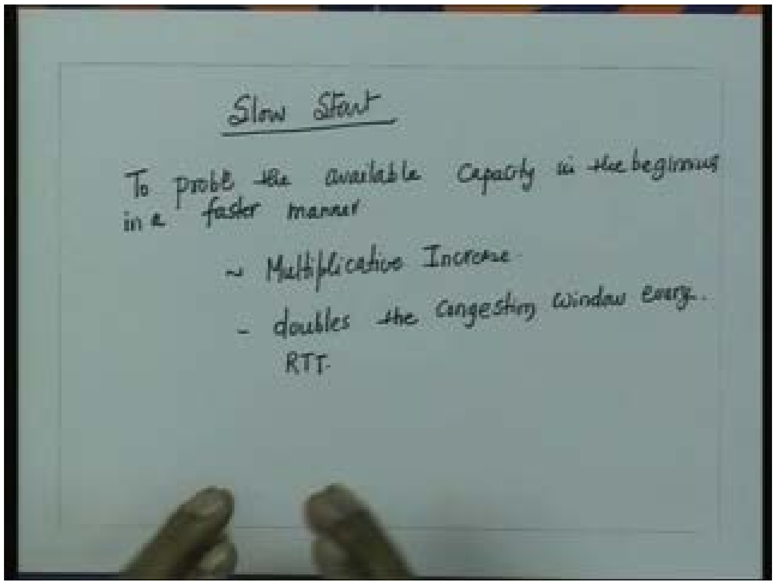
So therefore, the maximum window size the maximum window size can only be the minimum of these 2. So, the max win that is the maximum window size can only be the minimum of the congestion window that is CWnd and the receivers advertised window. This only will determine - what is the maximum amount of window size that the receiver can see. Now, the evolution of the evolution of the window size in additive increase and multiplicative decrease, we are talking of the congestion window - the CWnd variable. The advertised window, the advertised window is actually explicitly conveyed by the receiver to the source. So, all we are talking is of the CWnd.

Now, what happened is historically, when the TCP does not know anything about the capacity of the network and the receiver conveys him about the maximum window size; then historically what happen is that the source tried to transmit the packets worth of the entire receiver window size. Now, when that was transmitted obviously sometimes it led to a congestion collapse in the network. In the sense that even though the receiver may be ready to absorb that much data, but since the network is congested; obviously you cannot transmit so much data.

So therefore, what was then thought that we would then try to increase the window size from W is equal to 1. Now, the question really is that if the network is uncongested or the network is having a large amount of buffer space and so on; then if the congestion window is increasing from W is equal to 1, then it would take a long time for the source to reach the capacity of the network.

So therefore, what was suggested is that we should initially, when we are starting up, initially, when we start up; we should increase the congestion window size in a little bit fast manner and so therefore a mechanism was introduced in the TCP which is called as slow start phase.

(Refer Slide Time: 31:48)



Slow Start

To probe the available capacity in the beginning in a faster manner

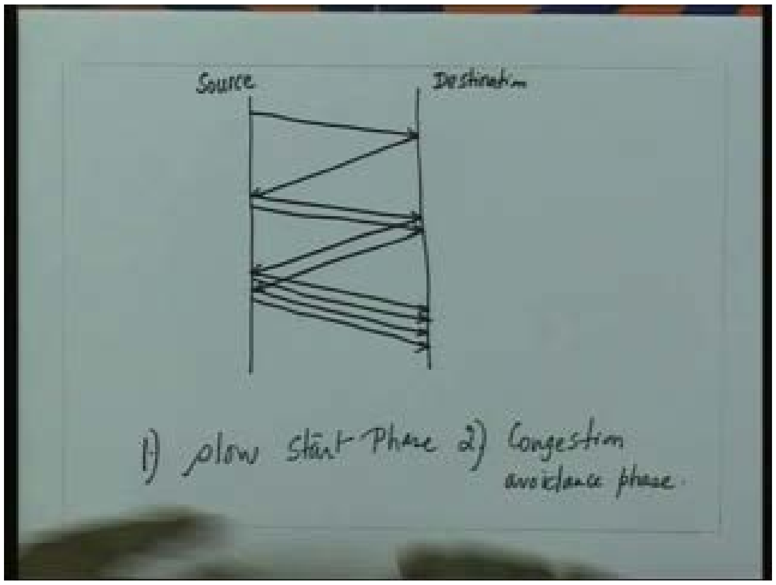
~ Multiplicative Increase

- doubles the congestion window every RTT.

So, we have the slow start mechanism in the TCP. The objective is really to probe the available capacity to probe the available capacity in the network in a faster manner. So now, what is done in the slow start manner is that slow start phase actually follows a multiplicative increase. So, probe the available capacity in the beginning in a faster, follows a multiplicative increase. So, what does it do? It doubles the congestion window size, it doubles the congestion window every RTT.

So earlier, as we had seen that we are following the additive increase. In the additive increase, the congestion window size was increased by only 1 every RTT. For an every acknowledgement, the congestion window size was increasing by 1 by W and after all W packets have been acknowledged, the congestion window size was increasing by only 1. In this case, the congestion window size is doubling every RTT. That means for every acknowledgement, for every acknowledgement; instead of increasing the congestion window size by 1 by w, here we are increasing the congestion window size by 1 packet.

(Refer Slide Time: 33:59)



So, the mechanism work something like this that here is a source here is a source and here is a destination. So, first the window size is 1 - W is equal to 1; so you send one packet. When you received the acknowledgement of this packet, the window size now becomes 2 and therefore you will send 2 packets. Now, when you receive the acknowledgement of these 2 packets, the window size will become 4 and you will now transmit 4 packets and so on.

So, the window size, the congestion window size is doubling every RTT. Now, this will increase till a certain window size which we call it to be a slow start threshold. So, this will keep on happening. So, the congestion window size doubles every RTT till slow start threshold, till you reach a point which is called as a slow start threshold or a packet loss occurs.

So, it is like an exponential growth, it is an exponential growth because you are following a multiplicative increase mechanism; it is an exponential growth. But still ironically, we are calling it to be a slow start because remember that in the early days of the TCP'S implementations, the window size was not increased from W is equal to 1 but it was increased or it was kept directly equal to the receivers advertised window. So, instead of directly keeping it to the receivers advertised window; we are keeping it, you are starting it from W is equal to 1 but trying very fast to reach something equal to the receivers advertised window.

Now, we can set a slow start threshold and we can try to increase to the slow start threshold. But obviously, if a packet loss occurs earlier; then we have to update the slow start threshold variable because obviously our slow start threshold variable was not kept properly. So, after the slow start threshold variable, then the TCP will flow an additive increase and multiplicative decrease mechanism. So, now the additive increase and multiplication decrease mechanism that we are discussed earlier is now called the congestion avoidance phase.

So therefore, now the TCP has two phase; one, we are calling it to be a slow start phase, another we call it to be a congestion avoidance phase. This is slow start phase and then we have a congestion avoidance phase. Now, these are the mechanisms which are which have been introduced into the TCP, the slow start phase essentially has been introduced; so, when the window size can be ramped up faster to the available capacity in the networks.

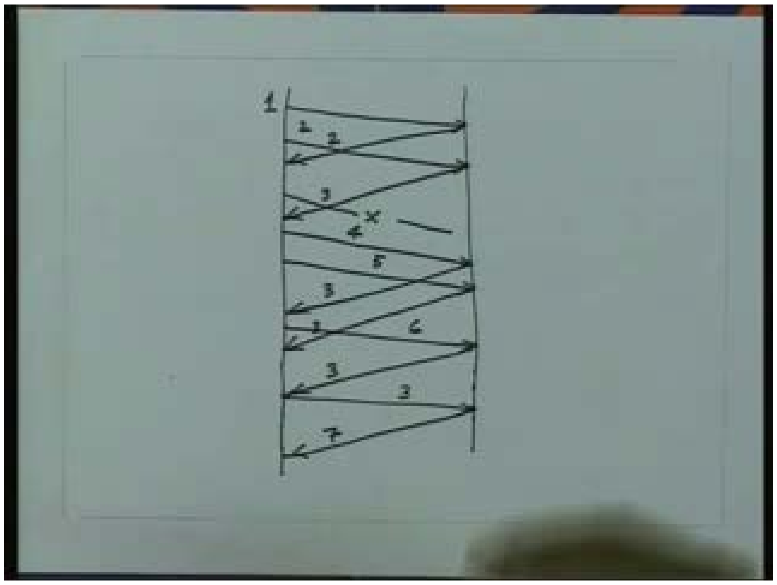
Now, therefore this slow start phase the slow start phase definitely occurs always in the beginning of a connection. In the beginning of a connection, you are always there in the slow start phase. So, you have an exponential growth upto the slow start threshold and then from there you follow a additive increase and multiplicative decrease and then you will always remain in the congestion avoidance phase.

Now, 2 mechanisms have also been introduced. Let me talk of that and then we will again come back to the slow start mechanism. Now, we were saying that when does the source determine that a packet loss has occurred? We were saying that whenever the TCP transmits a windows worth of data, it starts a retransmission timer. Now, the retransmission time out interval is approximately equal or approximately takes into account the time that it will take for a packet to reach the destination and for the acknowledgement to travel it back.

Now, if your acknowledgement does not come within a certain time out interval, then obviously the TCP determines that the packet must have been lost. There is however another way of determining that a packet has been lost and that is because of the particular mechanism of the TCP'S acknowledgement.

Now, note that the TCP'S acknowledgement does not explicitly acknowledge a segment or does not explicitly acknowledge certain byte. But it always tells you what is the next expected sequence number. That is very important because the TCP is not acknowledging explicitly the bytes, it will always tell you what is the next expected sequence number.

(Refer Slide Time: 40:26)



Just to give you illustrate that with an example; for example, now let me just to take an example by using the packet mechanisms even though the TCP'S acknowledges, TCP'S sequence number or the acknowledge numbers is for the bytes because the TCP is a byte orientated protocol. But we will take the example with a packet.

So, when the TCP sends let us say the packet number 1, now the acknowledgement will tell you that I am expecting the packet number 2; that is what the acknowledgement will be. Now, the TCP sends the packet number 2; then the acknowledgement will tell that I am expecting the packet number 3. Now suppose, this packet number 3 was sent and it was lost somewhere, it did not reach here; however the packet number 4 reached here, now in the response to the reception on the packet number 4, the receiver will still say that I am expecting the packet number 4.

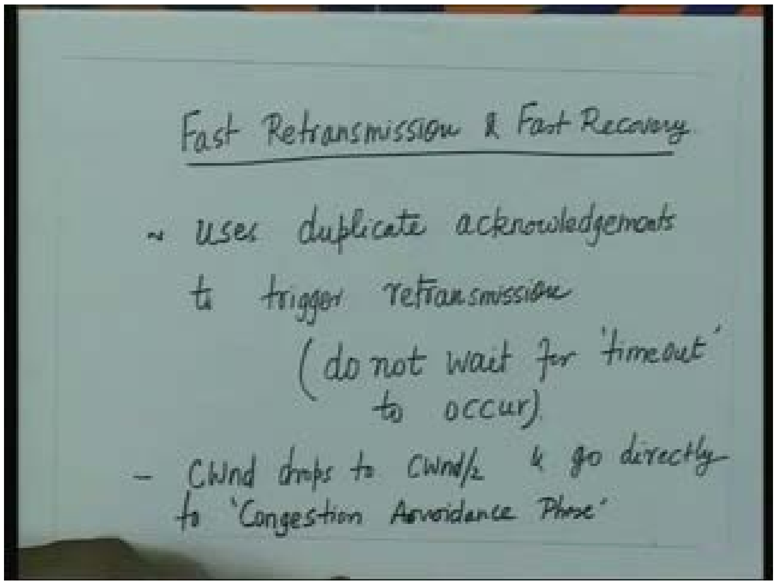
Now, the packet number 5 also might reach here and now in the response to the packet number 5, the receiver will always say that I am expecting the packet number 3. Here, the packet number 6 may be sent here and in the response to packet number 6, again the receiver will say that I am expecting the packet number 3. Now, note that the receiver's acknowledgement will come only on the receipt of a segment.

Now, what is happening here if the receiver is repeatedly telling you that I am expecting the packet number 3, I am expecting the packet number 3; it also implicitly means that the receiver is receiving certain segments. So, if you receive repeatedly these duplicate acknowledgements and the TCP standard says that if you receive 3 duplicate acknowledgement, so if you receive repeatedly these duplicate acknowledgement 3 time - that means I am expecting packet number 3, I am expecting packet number 3, I am expecting the packet number 3; then it is reasonable to conclude that because of certain congestion in the networks, the packet number 3 must have been lost and in that case the TCP can retransmit the packet number 3. The TCP can actually retransmit the packet number 3; this is what is called as the fast retransmit.

So, you can so the fast retransmit mechanism is that means that you can use, you can retransmit the packet number 3. Now, you retransmit the packet number 3; in response to the packet number 3, now what the receiver will say? Which packet I am expecting? Note that the receiver has already received packet number 4, 5 and 6 successfully. Now, when the packet number 3, the retransmitted packet number 3 is received by the receiver successfully; the receiver will say that I am expecting now the packet number 7 because the receiver, the TCP'S acknowledgement mechanism is a cumulative acknowledgement mechanism. It cumulatively acknowledges all the pervious packets also that have been received.

So, this is what is called as the fast retransmit mechanism where the TCP uses duplicate acknowledgements to trigger the retransmissions. Now, in this case note that since the packet number 3 has been lost, obviously there has been congestion in the networks. However, that congestion does not appear to be that severe as due to the retransmission time of interval and therefore in this case, the TCP drops its window size only to W by 2. So, that is what is called as the fast recovery. After that there is no there is no slow start phase. So, this is what is called as the fast retransmission and fast recovery mechanisms. So, fast retransmissions and fast recovery.

(Refer Slide Time: 44:44)



Fast Retransmission & Fast Recovery

- uses duplicate acknowledgements to trigger retransmission (do not wait for 'timeout' to occur)
- CWnd drops to CWnd/2 & go directly to 'Congestion Avoidance Phase'

Now, in the fast retransmission; what does it do? It uses duplicate acknowledgement instead of time out, it uses duplicate acknowledgements to trigger retransmissions. So, what you say? You do not wait for do not wait for time out to occur, the TCP does not wait for the retransmission time timer to time out and then retransmit; it uses duplicate acknowledgement to trigger the retransmission. It knows that the acknowledgements are duplicated and that means this packet, particular packet must have been lost and therefore it triggers the retransmissions.

Also, fast recovery means that the window, the congestion window CWnd drops to CWnd by a half and go directly to the congestion avoidance phase, congestion avoidance phase of additive increase and multiplicative decrease. Now suppose, so let us now recapitulate how the TCP'S operating.

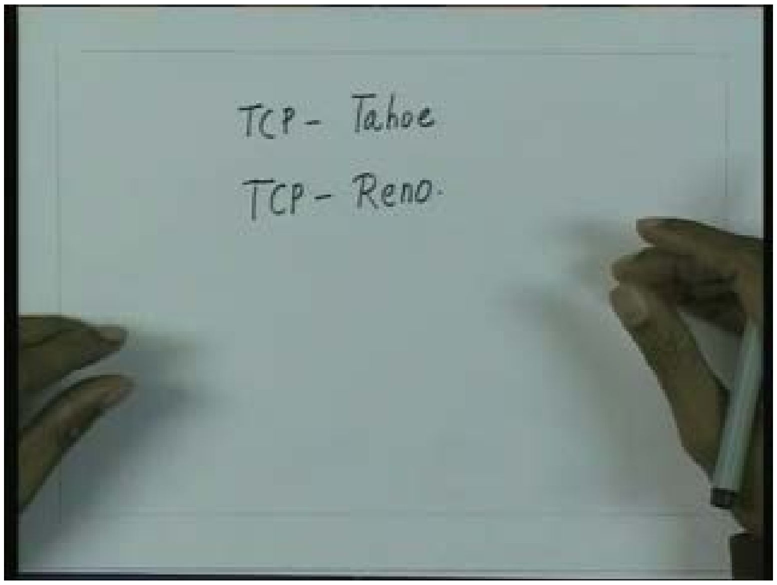
In the beginning in the beginning the TCP has a slow state phase which is actually an exponentially growth. So, it is not quite state but it is following an exponential increase. Now, the TCP has this exponential slow start phase, goes upto the slow start threshold the slow start threshold. After the slow start threshold, it then has an additive increase and multiplicative decrease phase which is the congestion avoidance phase.

Now, if in the congestion avoidance phase, if your retransmission timer times out and if you are not receiving an acknowledgement; then the TCP now will drop its window size to W is equal to 1. It will not drop its windows size to W is equal to half but it will drop its window size to W is equal to 1 if the retransmission timer times out. And then, the slow start phase will begin again.

However, if you are in the congestion avoidance phase and if you detect the packet loss or if you receive the duplicate acknowledgement, then you will drop the window size to W by 2. Now, in the first phase that we had discussed, where the retransmission timer times out and the windows size drops to W is equal to half; the slow start threshold, the assess thresh is now set to the window size by 2, window size the maximum window size by half, where the window size has reached.

So, now there is a new slow start threshold and that is how the TCP'S congestion control mechanism will operate. Now, this fast retransmissions and fast recovery has been implemented in new versions of the TCP which are called as the TCP Reno and earlier that is the retransmission time out mechanism that has been implemented in a mechanism which was called as TCP Tahoe.

(Refer Slide Time: 49:15)



So, there are 2 actually versions which is one that is called as the TCP Tahoe and the TCP Reno. The TCP Tahoe versions, it implements the retransmission time out mechanisms; on other hand, TCP Reno is implementing fast retransmissions and the fast recovery. Now again, I want to reiterate slow start phase therefore operates in 2 modes.

One is that you operate in the beginning of every connection, so the slow start phase operates in the beginning of every connection. So, you start with the slow start phase of the exponential growth and then you enter it into an additive increase or multiplicative decrease fashions and when the retransmission timer times out; then essentially, if you have basically if you have sent the entire windows worth of data and if you are not receiving any acknowledgment, if you are not receiving any acknowledgement and now you are waiting for the retransmission timer to time out and when the retransmission timer actually times out, then again you have to start with the slow start phase.

So therefore, the slow start phase will happen in I mean can happen in 2 ways: either because of the retransmission timer time out interval or retransmission time out or in the beginning of the phase. Otherwise, we will always going to the congestion avoidance phase if you are detecting the congestion due to duplicate acknowledgement.

Now, what we will do next is that that we will try to analyze the TCP’S throughput when the slow start phase is operating and try to determine how much is the TCP’S throughput when the during the slow start phase and during the congestion avoidance phase, particularly in a network with high bandwidth delay product.

So, we will also define what is meant by a high bandwidth delay product and then try to analyze, derive an expression for the TCP throughput in networks with high bandwidth delay products.