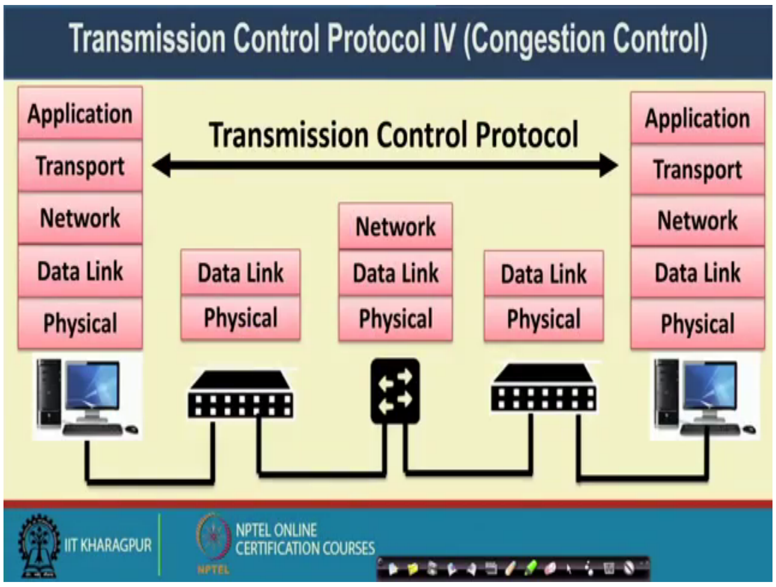
Computer Networks and Internet Protocol

Prof. Sandip Chakraborthy
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 22
Transmission Control Protocol – IV (Congestion Control)

Welcome back to the course on Computer Network and Internet Protocols. So, we are looking into the Transmission Control Protocols in detail. So, in the last class, we have looked into the flow control algorithms in TCP.

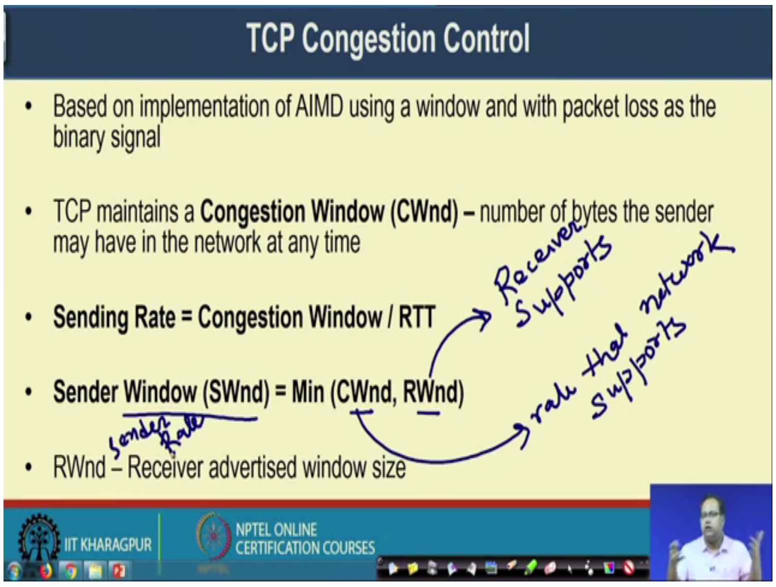
(Refer Slide Time: 00:28)


Transmission Control Protocol IV (Congestion Control)

So, in this lecture, we look into the congestion control algorithms in TCP. In the context of generic transport layer protocols, you have looked into that this congestion control algorithms in a transport layer, they use a nice principle to ensure both efficiency as well as fairness.

So, here we will look into that how TCP utilizes the concept of capacity and fairness together in the congestion control algorithm, which it incorporated.

(Refer Slide Time: 01:03)



TCP Congestion Control

- Based on implementation of AIMD using a window and with packet loss as the binary signal
- TCP maintains a **Congestion Window (CWnd)** – number of bytes the sender may have in the network at any time
- **Sending Rate = Congestion Window / RTT**
- **Sender Window (SWnd) = Min (CWnd, RWnd)**
- RWnd – Receiver advertised window size

Receiver Supports
rate that network supports
Sender Rate
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, here is the basic congestion control algorithm for TCP. So, the basic congestion control algorithm for TCP is based on the implementation of additive increase multiplicative decrease, using a window based control mechanism, and TCP considers packet loss as a notion of congestion. So, earlier we have looked into these principles of additive increase multiplicative decrease, where we have seen that well. AIMD provides a notion of maximizing the capacity as well as fairness in the network.

So, what we have looked into that whenever there are multiple flows, we are which are getting congested at the network bottleneck or that the bottleneck link, both the flows are trying to maximize their capacity. And when both the flows tries to maximize their capacity during that time, we need to ensure that every flow gets the required capacity that maximizes its fairness as well. So, considering the global network perspective, where multiple flows are contending with each other to get the information to get the data during that time; this notion of congestion and the notion of fairness since congestion is an important aspect.

And what we have seen earlier, that in case of a distributed network ensuring hard fairness is very difficult. And at the same time, hard fairness can affect your capacity the available capacity or it can under utilize the available network capacity. So, to remove the under utilization of the network capacity that in network; generally, we use the max min fairness principle and the max min fairness principle says that well, your objective would be to maximize the end to end capacity that you can get for a particular flow. In that notion, we have seen that AIMD provides this max min fairness allowing it maximizing the utilization of the link bandwidth that you have at the network bottleneck in contrast to additive increase additive decrease principle of rate control, and multiplicative increase multiplicative decrease notion of rate control.

So, TCP incorporates the same principle of additive increase multiplicative decrease, where it increases the rate additively, whenever there is sufficient amount of bandwidth available in the end to end path. And whenever TCP detects a congestion with the help of a packet loss, where packet loss gives a signal to congestion, then it drops the rate in a multiplicative way or following a multiplicative notion. So, this AIMD principle that we discussed earlier is incorporated in TCP.

So, to incorporate this notion of AIMD for congestion control, while maintaining fairness; TCP maintains a window, which is called a congestion window. So, this congestion window is the number of bytes that the sender may have in the network at any instance of time. Now, if that is the case, then you are sending rate will be equal to the congestion window divided by RTT. So, RTT is the round trip time, the time to propagate a message from one node to another node and then getting back the reply, this total round trip time.

So, if you divide the congestion window size that means, the number of bit or the number of bytes that the system can push to the network divided by the total RTT, that gives you an approximation of the sending rate, when the sender is sending data at a rate of congestion. But, as you have seen earlier, we have another notion here, which is the receiver window size at a rate of the receiver, which comes from the flow control algorithm.

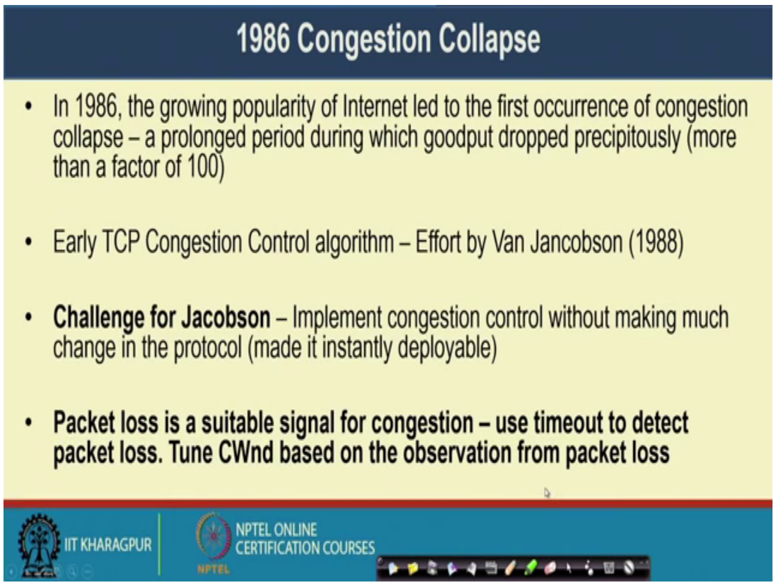
So, ideally what we have to do, we have to incorporate or we have to merge the flow control principle along with the congestion control principle. And what we have seen earlier that well, the flow control it maintain its own window size, which is based on the feedback of receiver advertised window. So, the receiver advertised window, receiver announces that that is the available place at the available buffer at the receiver. So the sender can send that much amount of data, so that buffer overflow does not occur from the receiver side. And with that, from this receiver feedback window size, the sender also control its rate.

So, ideally your sending rate should be the minimum of the receiver rate, the rate at which the receiver can receive the packet. And the rate at which the network can deliver the packet to the other rate, so that is why your sending rate should be minimum of the receiver rate and the network supported rate. Now, this concept of network supported rate, it is coming from the congestion control algorithm; and the receiver rate that is coming from the flow control algorithm.

Now, the receiver advertised window size that gives you the possible rate at which the receiver can receive the data. And the congestion window size that is giving the rate at which the network can support transmission of data over the end to end links. So that way your sender in the size should be the minimum of congestion window, and the receiver window. So, this congestion window is providing you the rate that network supports; and receiver window is giving you the rate that receiver supports. So, that way and thus we are getting the sender window that is giving you the sending rate or the sender rate.

So, this is the notion of combining congestion control and flow control together in TCP. And with this principle, TCP controls the sender in the size. So, initially your congestion window size is kept at a very minimal rate. Then, we increase the congestion window size gradually to find out that; what is the bottleneck or what is the capacity of the link. And if a congestion is detected, then we apply the AIMD principle to decrease the rate, so that is the broad idea of congestion control in TCP.

(Refer Slide Time: 07:50)



1986 Congestion Collapse

- In 1986, the growing popularity of Internet led to the first occurrence of congestion collapse – a prolonged period during which goodput dropped precipitously (more than a factor of 100)
- Early TCP Congestion Control algorithm – Effort by Van Jancobson (1988)
- **Challenge for Jacobson** – Implement congestion control without making much change in the protocol (made it instantly deployable)
- **Packet loss is a suitable signal for congestion – use timeout to detect packet loss. Tune CWnd based on the observation from packet loss**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us see that how this entire congestion control in TCP was invented, and this congestion control in TCP is a nice example of a networking or a better to say decentralized or distributed networking protocol. So, this entire concept of congestion came in 1986 from event called a congestion collapse. So, in 1986, this growing popularity of internet it led to the first occurrence of congestion in the network, so we call it as the congestion collapse. So, this congestion collapse was a prolonged period during which the goodput of the system that they dropped significantly more than a factor of 100.

So, based on this notion of congestion collapse that observation of congestion collapse; Van Jacobson was the key person who actually investigated this phenomenon of congestion in the internet. And this early TCP congestion control algorithm that was developed by Van Jacobson. And the initial proposal came around 1988, which was successor of the event of congestion collapse that happened in 1986.

But, the challenge for Jacobson was to implement the congestion control algorithm without making much changes in the protocol, it was necessary to make the protocol instantly deployable in the network, because during that time, many of the systems were using TCP for transmission of data. And the notion of internet was already there, people were accessing data over the internet. So, during that time if you design a completely different protocol, what you have to do, you have to make the change to all the machines in the system.

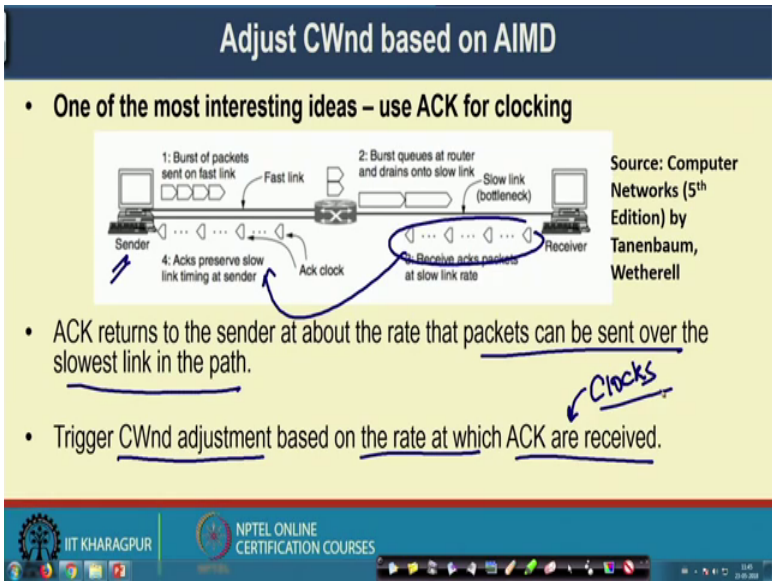
And during that time, the size of the network was may not be as large as it is today or maybe some thousand factors lesser than today, but still it was significant. And design of a complete new protocol may lead to a problem of backward compatibility. So that was the major challenge for Jacobson to design a new protocol, which would be compatible with the existing implementation of TCP, and you do not require much changes in the TCP protocol.

So, to solve this, Jacobson had a nice observation, and the observation was that he found that well packet loss is a suitable notion for congestion. Now, here there is a glitch, remember that during that time, when Jacobson invented this congestion control algorithm in TCP, the notion of wireless network was not that much popular or it was just in a very nascent stage, so most of the internet was connected by the wire network. And in a wire network, in general you do not get a loss from the channel, because the communication media is guided, it is where so your link layer protocol will take care of the interference in the channel, and that is why you will not experience a loss from the channel. So, your channels are kind of lossless in case of a wire network.

So, if there is a loss of packet, that loss is certainly from the network buffers from the intermediate network devices. And because of that if there is a packet loss from the network buffer, you can certainly infer that the buffer has overflown, and because the buffer has overflown, you are having a congestion in the network so because of the congestion, the buffer can only overflow. So, that way Jacobson found out that the packet loss is a suitable signal for congestion, so you can use the timeout to detect a packet loss. And then tune the congestion window based on the observation from packet loss.

So, you increase the congestion window value, the moment you observe a packet loss that gives you an indication that well, there is a notion of congestion in the network or there is some occurrences of congestion in the network, because of which the packet has lost, which you have detected from a timeout. And whenever you are detecting a packet loss from a timeout, then you apply AIMD principle to reduce your rate based on the multiplicative decrease principle.

(Refer Slide Time: 12:09)



Adjust CWnd based on AIMD

- One of the most interesting ideas – use ACK for clocking

1: Burst of packets sent on fast link
Fast link
2: Burst queues at router and drains onto slow link
Slow link (bottleneck)
Sender
Receiver
4: Acks preserve slow link timing at sender
Ack clock
3: Receive acks packets at slow link rate

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

- ACK returns to the sender at about the rate that packets can be sent over the slowest link in the path.

Clocks

- Trigger CWnd adjustment based on the rate at which ACK are received.

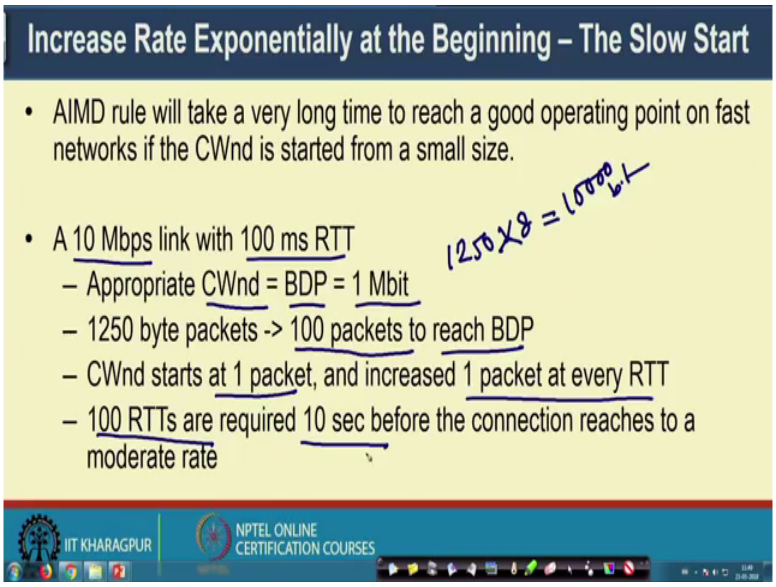
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Well. So, here is the another interesting observation that how will you adjust the congestion window based on the AIMD principle. So, one of the most interesting ideas in TCP congestion control is use of acknowledgement for clocking. So, here this picture depicts the interesting fact that well. Whenever you are sending the packet, so assume that this is the network scenario, you have the sender, and an intermediate router, and then the receiver, they are connected were two links, so it is a two hop path. So, this link is a fast link. So, this is a fast link, and the second link is a slow link. So, this second link is basically your bottleneck link. So, the congestion will happen, when lots of data will be pushed to this lower link or the bottleneck link.

Now, whenever you are sending the packet, so you are sending the packet from the sender in the form of a burst, that traverse the faster link, then in the slower link, because this link is slower, your packet will take more time to propagate to the receiver. Now, when if you look into the acknowledgement the rate of the acknowledgement will actually be the rate of sending the packet at this slower link; so whatever be the rate of acknowledgement at this slower link, the same rate of acknowledgement will perceive in this fast link as well. So, that way whenever the sender will get the acknowledgements, that acknowledgement actually indicates the rate of the slower link, which is there in your end to end path or better to say if you have multiple hop path, then the rate of the slowest link in that the rate, the acknowledgement will arrive at the sender.

Now, if sender monitors the rate of acknowledgement that gives an idea that well, possibly at that rate, the packets are being transmitted to the receiver. So, the acknowledgement returns to the center at about the rate, that the packets can be sent over the slowest link in the path. So, you trigger the congestion into adjustment based on the rate at which acknowledgement are received. So, here these acknowledgements are used as a clocks to trigger the congestion control in the network. So that was another interesting observation by Jacobson, while designing the congestion control algorithm, so well, so that was the basic principle.

(Refer Slide Time: 14:41)



Increase Rate Exponentially at the Beginning – The Slow Start

- AIMD rule will take a very long time to reach a good operating point on fast networks if the CWnd is started from a small size.
- A 10 Mbps link with 100 ms RTT
  - Appropriate CWnd = BDP = 1 Mbit
  - 1250 byte packets -> 100 packets to reach BDP
  - CWnd starts at 1 packet, and increased 1 packet at every RTT
  - 100 RTTs are required 10 sec before the connection reaches to a moderate rate

$1250 \times 8 = 10000$ bit

Now, whenever you are getting an acknowledgement, you will you will trigger or you will adjust your congestion window size, but the question comes that at what rate you will apply additive increase in the network. So, initially what you can do that you can set the congestion window to one packet, and then gradually increase the size of the congestion window, when you will receive an acknowledgement

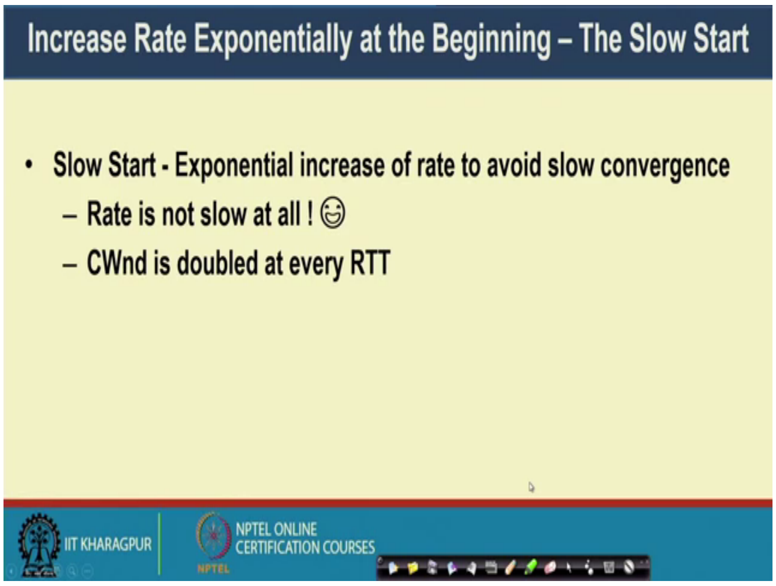
Now, let us see what happens, if you apply a additive increase principle. So, if you in apply a additive increase principle, what additive increase says, that initially you send one packet, whenever you are making an acknowledgement, then you increase the congestion window by one, so now your congestion window is two, so you send two packets. So, once you are successfully receiving that two packets, the acknowledgement for those two packets, then you increase your congestion window by one, so now your congestion window is three, so you can successfully transfer three packets. You wait for the acknowledgement for those three packets, whenever you are receiving the acknowledgement for those three packets. Again you increase the congestion window to four from three, and send four packets.

Now, if you are applying this additive increase of congestion window over the network. So this AIMD rule, it will take a very long time to reach a good operating point on fast networks, because this congestion window started from the small size. So, let us see an example. So, assume that you have a 10 mbps link you have a 10 mbps link with 100 milliseconds of round trip time, and in that case, your appropriate congestion in the size should be equal to BDP the Bandwidth Delay Product, that we have seen earlier. So, with 10 mbps link and 100 millisecond RTT; your bandwidth delay product comes to be 1 megabit.

Now, assume that you have a 1250 byte packets, you have a 1250 byte packets means, you have 1250 into 8, so that means, 10000 bits packet. And if you have a 10000 bit 10000 bit packet so if you have a 10000 bit packet that means, with 1 megabit BDP, you can transfer you need to transfer at least 100 packets to reach to the bandwidth delay product. Now, if you assume that the congestion window starts at 1 packet, and the value of the congestion window is increased 1 packet at every RTT, because this RTT is based on the rate at which you are receiving the acknowledgement. So, at every RTT, you increase 1 congestion window.

So, you require 100 RTTs and 100 RTTs means with 100 millisecond per RTT, it is approximately 10 second before the connection reaches to a moderate rate or it reaches to our its maximum capacity. Now, if you think about the wave transfer the HTTP protocol, so none of the HTTP connection takes 10 second to reach at that operating point. So, by the time, you will reach at the maximum capacity, we will probably close the TCP connection, and start again a new connection, which will again increase from one packet per second. Now, to increase this rate, we apply a principle in TCP, which is called slow start.

(Refer Slide Time: 18:08)



Increase Rate Exponentially at the Beginning – The Slow Start

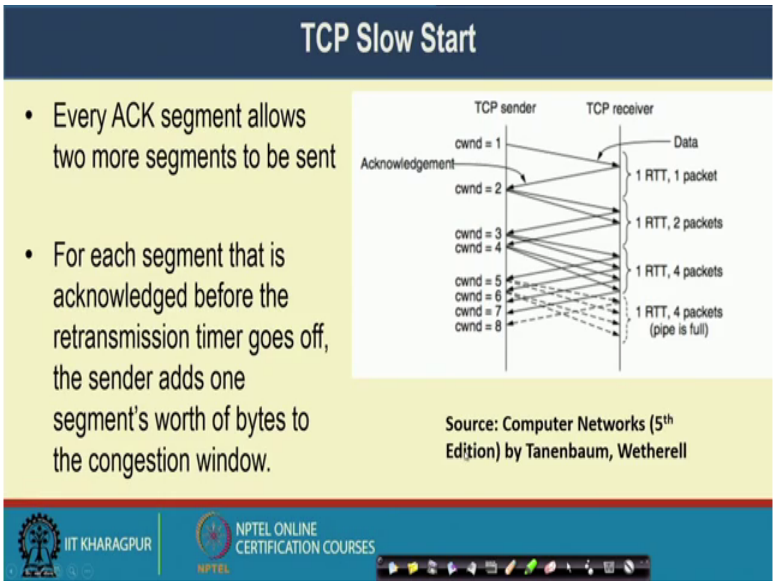
- Slow Start - Exponential increase of rate to avoid slow convergence
  - Rate is not slow at all ! ☺
  - CWnd is doubled at every RTT

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, what is the slow start, so the slow start is something like this that initially you make a exponential increase of the rate to avoid the slow convergence. Now, this is the irony in the name that slows that does not mean that your rate is not your rate is not your rate is slow, it is just like that, you are starting from a slower let rate and making a faster convergence to the high rate.

So, what we do at slow start, we increase the congestion window by two that means, the congestion window is doubled up at every RTT. So, rather than have an additive increase, initially we do a multiplicative increase by doubling up the congestion window at every round trip time.

(Refer Slide Time: 18:47)



TCP Slow Start

- Every ACK segment allows two more segments to be sent
- For each segment that is acknowledged before the retransmission timer goes off, the sender adds one segment's worth of bytes to the congestion window.

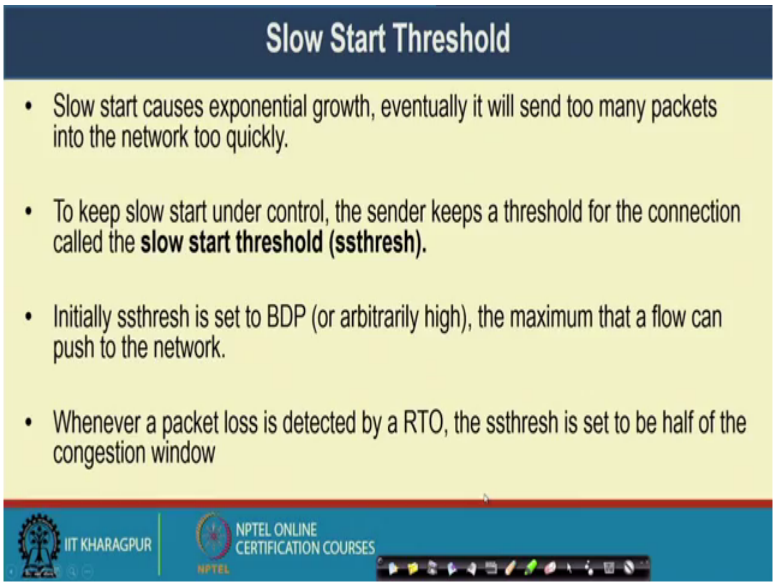
Source: Computer Networks (5th Edition) by Tanenbaum, Wetherell

So, that is the notion of TCP slow start. That every acknowledgement segment allows two more segments to be sent. So, for each segment that is acknowledged before the retransmission timer goes off, the sender adds one segment worth of bytes to the congestion window.

So, what happens here with this example, so initially your congestion window was 1. Once you are receiving the acknowledgement, your congestion window becomes two that means 2 into 1. Once you are receiving the second acknowledgements, your congestion window becomes 4. Then once you are receiving all these four acknowledgements, your congestion window becomes 8, so that way at every RTT and all of this transmission takes around one RTT.

In the 1st RTT, you are acknowledging 1 packet. In the 2nd RTT, you are acknowledging 2 packets. In the 3rd RTT, you are acknowledging 4 packets. In the 4th RTT, your acknowledgement you are acknowledging 4 packets. And if the pipe is full, then that is the level at which you are getting converged, so that way in TCP low start at every round trip time, we double up the congestion window size ok.

(Refer Slide Time: 19:58)



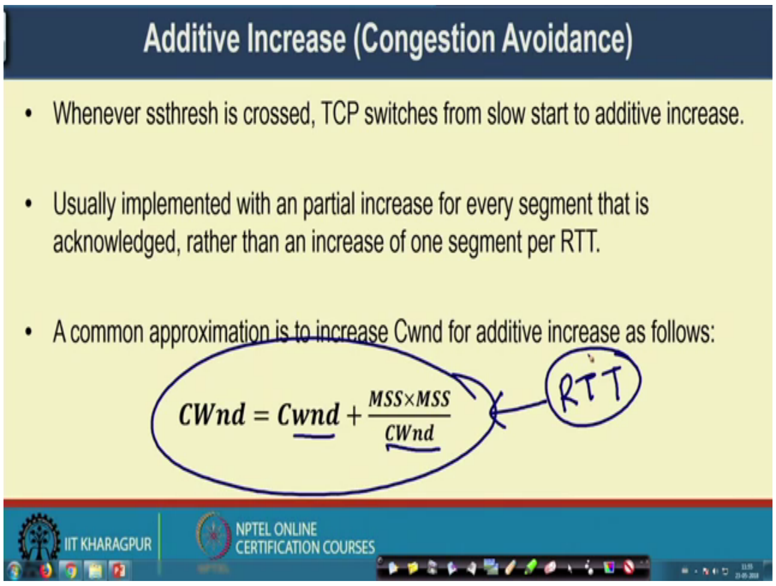
Slow Start Threshold

- Slow start causes exponential growth, eventually it will send too many packets into the network too quickly.
- To keep slow start under control, the sender keeps a threshold for the connection called the **slow start threshold (ssthresh).**
- Initially ssthresh is set to BDP (or arbitrarily high), the maximum that a flow can push to the network.
- Whenever a packet loss is detected by a RTO, the ssthresh is set to be half of the congestion window

Now, if you just make a multiplicative increase of congestion window, then again it violates the principle of max min fairness that we have seen earlier. So, MIMD multiplicative increase multiplicative decrease does not lead you to a to a optimal operating point, where both the capacity and fairness constants are satisfied. So that is why what we have to do, we have to make a additive increase at some point of time. So, what we do here? So the slow start it causes the exponential growth, so eventually it will send too many of packets into the network too quickly.

Now, to keep the slow start under control, the sender keeps a threshold for the connection, we call this threshold as the slow start threshold or ssthresh. Now, initially the slow start threshold is set to BDP or some value, which is arbitrarily high, the maximum that a flow can push to the network. And whenever a packet loss is detected by a retransmission timeout, the slow start threshold is said to be half of the current congestion window. So that is the idea.

(Refer Slide Time: 21:10)



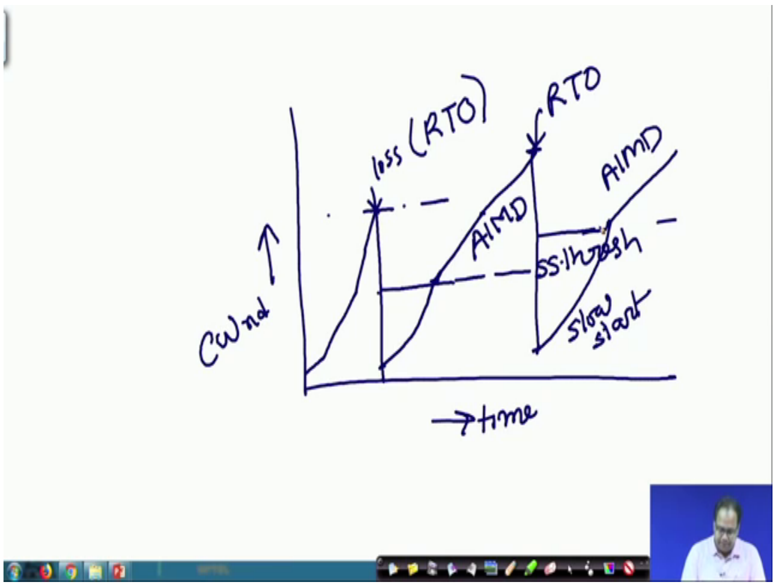
Additive Increase (Congestion Avoidance)

- Whenever ssthresh is crossed, TCP switches from slow start to additive increase.
- Usually implemented with an partial increase for every segment that is acknowledged, rather than an increase of one segment per RTT.
- A common approximation is to increase Cwnd for additive increase as follows:

$$CWnd = \underline{Cwnd} + \frac{MSS \times MSS}{\underline{CWnd}} \leftarrow RTT$$

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

(Refer Slide Time: 21:18)



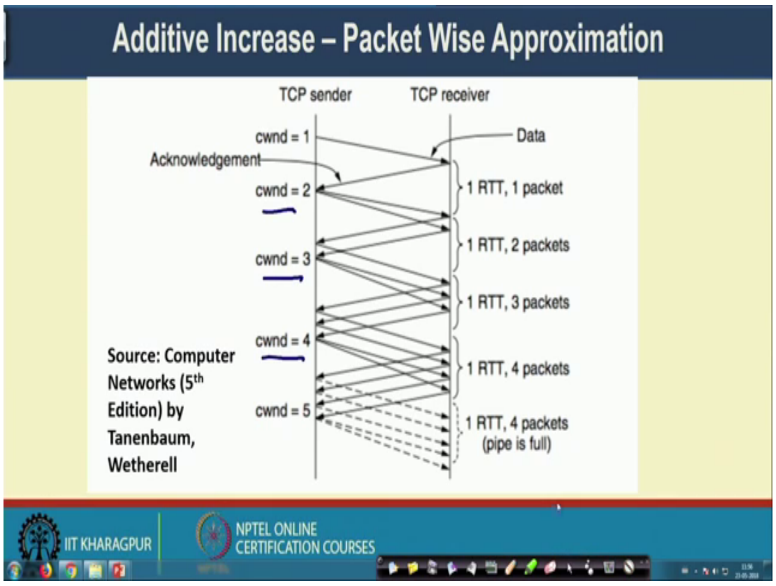
So, whenever your; so let me try to explain it with a diagram, so initially these things happen. So, at this axis, I have the time; and at this axis, I am plotting say the congestion window. So, you start with one packet, and initially you have a exponential growth. And say at this point, you get a loss or retransmission timeout, and whenever you are having a retransmission time out, so you set. So, this was the initial value. So, you set half of that as the slow start threshold.

So, you drop the rate, and now your slow start threshold is here. So, again you start with one congestion window, so you go exponentially up to the slow start threshold. After you have reached slow start threshold, you increase true AIMD. So, here your AIMD starts. Now, again after AIMD at this point, if you have an RTO, then you make half of that at the updated slow start threshold drop the rate, again make an exponential increase up to slow start threshold, and start AIMD after that. So, your slow start is up to the slow start threshold and after that, you are going with AIMD.

So, after slow start threshold, we move to the additive increase, so in this additive increase, which we call as the congestion avoidance. So, whenever the slow start threshold is crossed, TCP switches from slow start to additive increase. So, it is usually implemented with a partial increase for every segment that is being acknowledged, rather than an increase of one segment part RTT. So, this one segment part RTT is your slow start phase.

So, to do that, we make a common approximation is to increase the congestion window for additive increase based on this formula. So, the congestion window is increased as the current value of the congestion window plus the maximum segment size into maximum segment size divided by the congestion window. This gives an approximation of the additive increase that how much data or how much new byte need to be added to the congestion window to follow the; or to increase the congestion window value based on additive increase at every round trip time. So, at every round trip time, we approximate the increase of congestion window based on this formula. So, this formula is applied at every round trip time to have the congestion window followed additive increase principle.

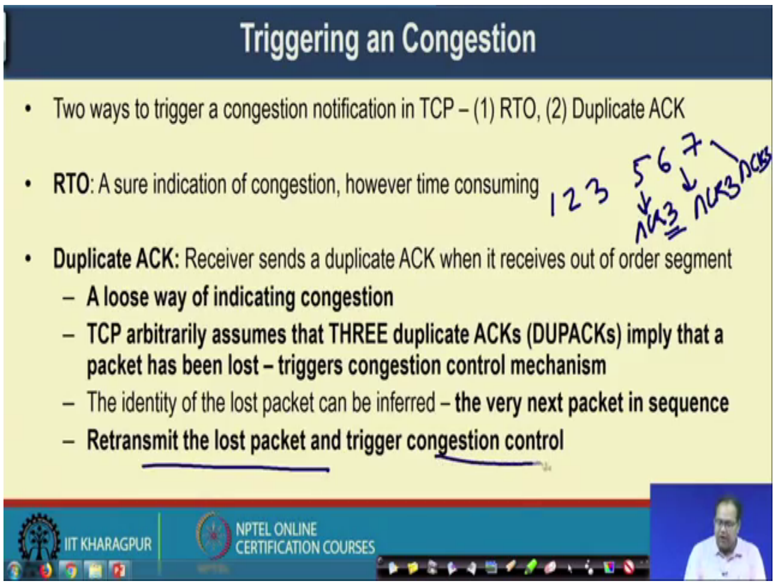
(Refer Slide Time: 24:16)


Additive Increase – Packet Wise Approximation

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

So, it looks like this. In additive increase, if you do the packet wise approximation, so in additive increase, initially they say the congestion window was 1. Once you are getting an acknowledgement, you make the congestion window to 2. Then you are getting these two acknowledgement, you are you are making congestion window to 3. Then in the next RTT, you are getting the acknowledgement for those three packets, so you are making congestion window to 4. So, this additive increase, we approximate based on the formula that we have given earlier, so that is the broad idea, so of increasing the congestion window by using the additive increase principle ok.

(Refer Slide Time: 24:56)


Triggering an Congestion

- Two ways to trigger a congestion notification in TCP – (1) RTO, (2) Duplicate ACK
- **RTO:** A sure indication of congestion, however time consuming
- **Duplicate ACK:** Receiver sends a duplicate ACK when it receives out of order segment
  - **A loose way of indicating congestion**
  - **TCP arbitrarily assumes that THREE duplicate ACKs (DUPACKs) imply that a packet has been lost – triggers congestion control mechanism**
  - The identity of the lost packet can be inferred – **the very next packet in sequence**
  - **Retransmit the lost packet and trigger congestion control**

So, to trigger a congestion, as we have mentioned that we normally trigger a congestion with the help of a retransmission time out, that indicates a packet loss, but there is another way to trigger the congestion. So, in general, TCP follows two different ways to trigger a congestion; one is the retransmission timeout, and the second one is by using a duplicate acknowledgement. So, duplicate acknowledgement means, you are continuously receiving the acknowledgement of the same packet.

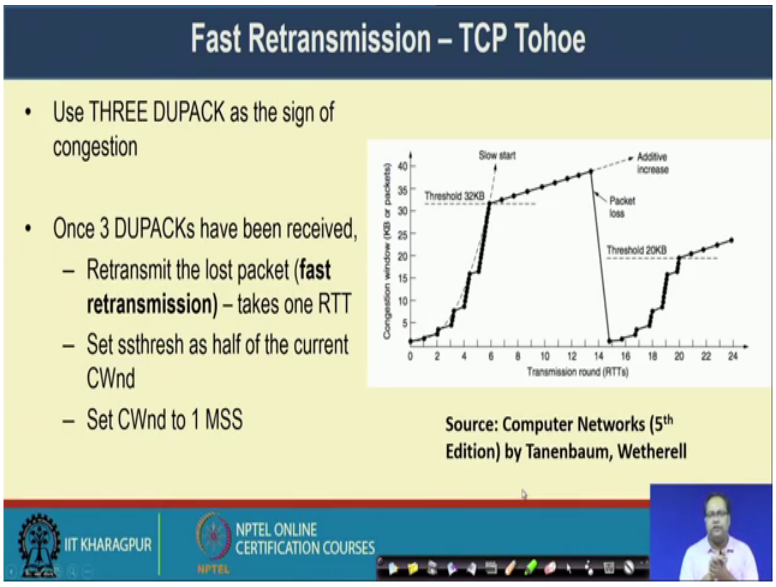
Now, why we use basically the duplicate acknowledgement to trigger a congestion control, because if you use retransmission timeout, you have to wait for that timeout duration to detect a congestion. Whereas, duplicate acknowledgement gives you a early notion of congestion, and you can trigger the congestion control behavior much earlier compared to waiting for a complete timeout period.

Now, interestingly this retransmission timeout RTO is a sure indication of congestion, but it is time consuming. Whereas, this duplicate acknowledgement, here as we know that the receiver sends a duplicate acknowledgement, when it receives out of order segment, which is a loose way of indicating a congestion. So, TCP assumes that if you are receiving three duplicate acknowledgements, then it implies that the packet has been lost, and it triggers the congestion control mechanism. Now, this assumption of three duplicate acknowledgement is something arbitrary, there is as such no logic behind that, so Jacobson assumed that well, you can take three triple duplicate acknowledgement to indicate a congestion.

Now, another important or interesting observation from this duplicate acknowledgement is that (Refer Time: 26:47) TCP uses cumulative acknowledgement. By looking into the duplicate acknowledgement, you can identify the lost packet, that which packet has been lost. So the very next packet in the sequence that have been lost, so that is why, you are getting this duplicate acknowledgement. So, what may happen, say you have received 1, 2, 3, then you have lost packet 4, and you are receiving 5, 6, 7. Whenever you are receiving 5, 6, 7, you will receive an acknowledgement up to 3. So, every time you will receive a duplicate ACK for ACK 3.

Now, in this case, you can infer that the packet 3 is actually the packet that has been lost. So, you retransmit the lost packet, and then trigger the congestion control algorithm.

(Refer Slide Time: 27:35)



Fast Retransmission – TCP Tohoe

- Use THREE DUPACK as the sign of congestion
- Once 3 DUPACKs have been received,
  - Retransmit the lost packet (**fast retransmission**) – takes one RTT
  - Set ssthresh as half of the current CWnd
  - Set CWnd to 1 MSS

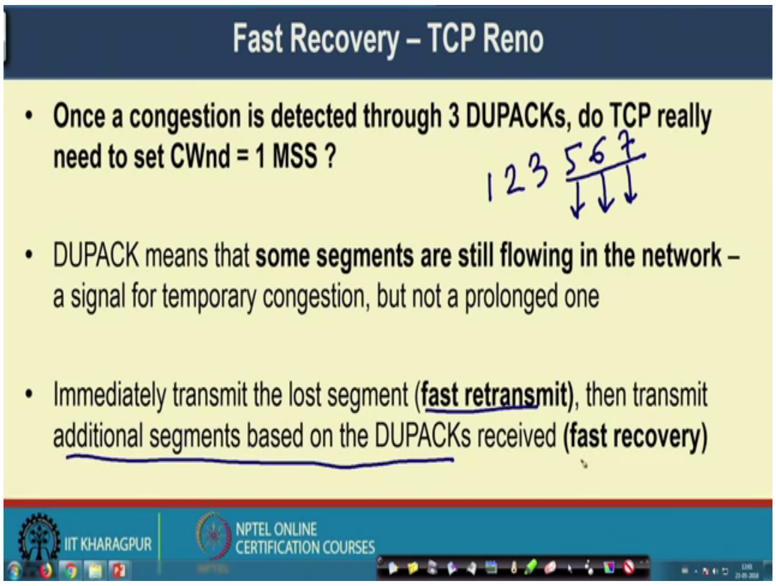
Slow start
Additive increase
Threshold 32KB
Packet loss
Threshold 20KB
Congestion window (KB or packets)
Transmission round (RTTs)

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

So, this concept is known as fast retransmission, which is incorporated in one variant of TCP congestion control, which is called a TCP Tohoe. So, in TCP Tohoe, we use three duplicate acknowledgement as the sign of congestion. So, once you receive three duplicate acknowledgement, you retransmit the lost packet that is the fast way retransmission, because you can infer the identity of that lost packet, it takes one round trip time. Then you set the slow start threshold as half of the current congestion window. And set the congestion window to 1MSS, so that is the idea of TCP Tohoe.

So, initially you have the slow start phase, so you reach the slow start threshold. Once you are reaching the slow start threshold, you are applying additive increase. At this point, you have detected a congestion by say packet loss. Whenever you are detecting a congestion, you drop the packet, again increase and meet the threshold to half of the current slow start threshold. So, earlier your slow current congestion window, so your current congestion window is 40; so you make the threshold as 20, and have slow start up to that threshold.

(Refer Slide Time: 28:51)



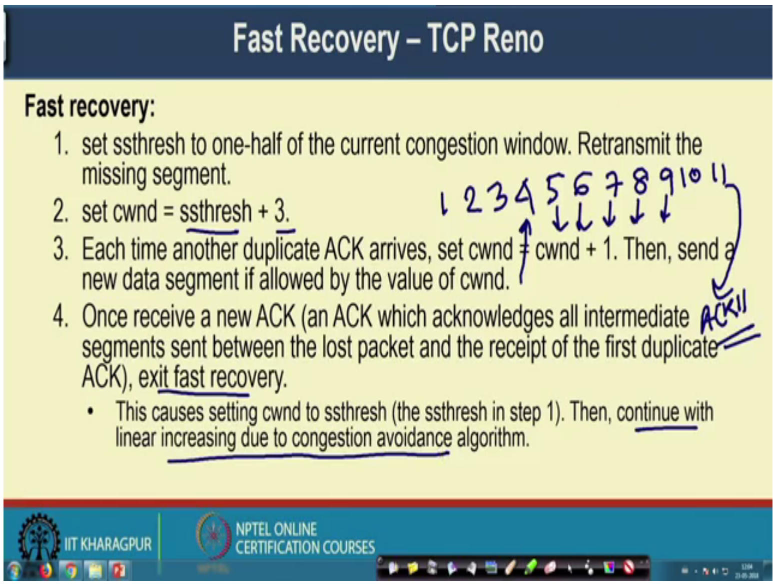
Fast Recovery – TCP Reno

- **Once a congestion is detected through 3 DUPACKs, do TCP really need to set CWnd = 1 MSS ?**
- DUPACK means that **some segments are still flowing in the network** – a signal for temporary congestion, but not a prolonged one
- Immediately transmit the lost segment (**fast retransmit**), then transmit additional segments based on the DUPACKs received (**fast recovery**)

Now, in TCP Reno, there is another interesting observation from the implementation of TCP Reno. So, once you are detecting a congestion through 3 duplicate acknowledgement, do TCP really need to set congestion window to 1 MSS? Now here, this duplicate acknowledgement; it means that some segments are still flowing in the network, it is not like that you are not able to send any packet. So, if you are having a retransmission timeout. That means, you are not receiving any acknowledgement, but whenever you are receiving some acknowledgement that means, possibly that packets say in the earlier example that I was showing, possibly packet 3 has been say packet 4 has been lost, but the receiver is receiving packet 5, 6, and 7.

So, when the receiver is receiving packet 5, 6, and 7, it is sending back a duplicate acknowledgement, so that means, you have lost possibly lost packet 3, but it is not like that this the link is entirely getting choked, some packets are still flowing in the network, so that is why the congestion is not that much severe. So, you do not need to reduce the congestion window again at 1 MSS, and start the slow start again. So, what we do here, you immediately transmit the lost segment, that we call as the fast retransmit, and then transmit additional segment based on the duplicate acknowledgement that has been received. We call this concept as the fast recovery.

(Refer Slide Time: 30:24)



Fast Recovery – TCP Reno

Fast recovery:

1. set ssthresh to one-half of the current congestion window. Retransmit the missing segment.
2. set cwnd = ssthresh + 3.
3. Each time another duplicate ACK arrives, set cwnd = cwnd + 1. Then, send a new data segment if allowed by the value of cwnd.
4. Once receive a new ACK (an ACK which acknowledges all intermediate segments sent between the lost packet and the receipt of the first duplicate ACK), exit fast recovery.
   - This causes setting cwnd to ssthresh (the ssthresh in step 1). Then, continue with linear increasing due to congestion avoidance algorithm.

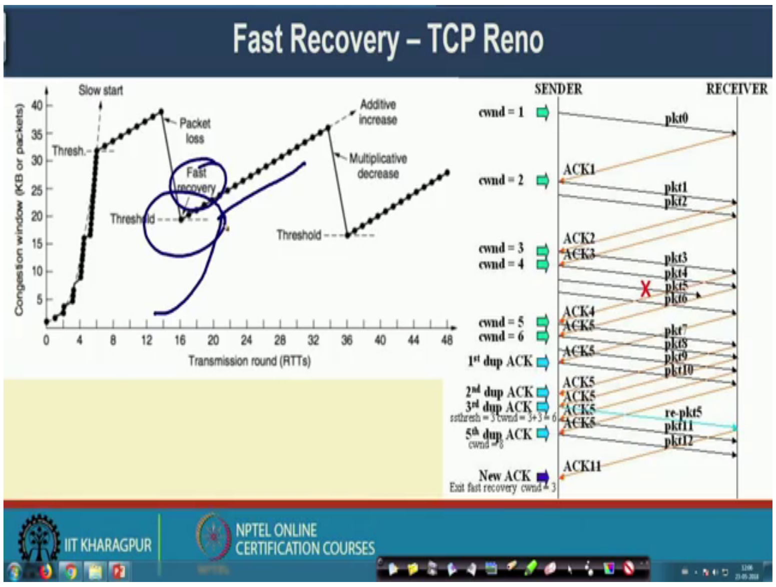
So, what happens in fast recovery, that you set the slow start threshold to one-half of the current congestion window, retransmit the missing segment that is the first retransmit. Then you set the slow start threshold to the current, then you set the congestion window to the current slow start threshold plus 3. Why 3, because you have received three duplicate acknowledgement that means, the receiver has already received three more packets. So that means, you can increase the congestion window, and send three more packets to the network, because that has been received by the receiver although out of order.

So, each time you receive another duplicate acknowledgement, you set the congestion window to congestion window plus 1, you increase the congestion window value. Then, send a new data segment, if allowed by the current value of the congestion window. So, that way, whenever you are receiving the duplicate acknowledgements, you are gradually increasing the congestion window value and sending more packet to the network, so that we call as the fast recovery. And the fast recovery ensures that, because some packets are flowing in the network, you do not need to again wait for sending the data at a very low rate.

So, once you receive a new acknowledgement, so you are receiving a new acknowledgement, because you have retransmitted the lost segment in fast retransmit, so that lost segment if it reached at the receiver. So, in that earlier example say 4 has been lost, the receiver has received 5, 6; so 7, 8, 9. And here you have received three duplicate acknowledgement corresponds to 5, 6, 7. And at that time, the sender has retransmitted segment 4, and it was receiving this 8 and 9 duplicate acknowledgement corresponds to that. And with every duplicate acknowledgement, you keep on sending the further packets.

And whenever the receiver received is missing 4 during that time, the receiver sends accumulative acknowledgement say ACK 11. And once you are receiving this new acknowledgement, not a duplicate acknowledgement that means, all the packets that was there, that have been acknowledged. So, at this stage, you exit from the fast recovery. So, this causes setting the congestion window to the slow start threshold, as we have done earlier. And continue with the linear increasing due to the congestion avoidance algorithm.

(Refer Slide Time: 32:49)



So, this is the idea of fast recovery that has been incorporated in TCP Reno, the next version of TCP Tohoe. So, in fast recovery what we do, initially you apply slow start up to slow start threshold, then you go for additive increase. At this stage, you have detected a packet loss through triple duplicate acknowledgement. Whenever you are detecting a packet loss through three duplicate acknowledgement, you reduce the congestion window value to the updated slow start threshold that is half of the current slow start, and apply fast recovery at that point. So, after applying fast recovery, whenever you are receiving a new acknowledgement, again you go (Refer Time: 33:30) the additive increase.

So, here if you compare with the TCP Reno variant, we are not making the congestion window. So, TCP Reno, it makes the congestion window again to 1 MSS, and then apply slow start threshold up to slow start up to the slow start threshold, and then applies the additive increase. So, here we are applying additive increase much faster, that will help you to reach to the operating point that much faster compared to TCP Reno due to the implication of this fast recovery algorithm. And avoiding the slow start phase, again whenever you are detecting a congestion through the triple duplicate acknowledgement.

But, if you are detecting a congestion by a retransmission timeout, then you always set the congestion window to one 1 MSS, and start with the slow start threshold, because a start with the slow start phase, because a retransmission timeout will give you an indication that a severe congestion has been happened. And whenever there is a severe congestion in the network, it is always better to start with a very low rate that means, setting the congestion window value to 1 MSS, so that is the broad difference.

So, if you are detecting the congestion by a retransmission timeout, you set the congestion window value to 1 MSS. Apply the slow start, whenever you will reach the current slow start value, which is the half of the congestion window that was detected during the congestion detection, and then apply additive increase. And if you are detecting a congestion through triple duplicate acknowledgement, you do not need to again perform the slow start phase, because some packet are still flowing in the network, you directly apply fast recovery, and then move with the additive increase. So, this is the variant of the TCP Reno.

After that, many other variants of TCP came into existence like TCP new Reno, then TCP selected acknowledgement or sack, so originally there are normal TCP protocol, it uses the principle of go back in flow control principles or go back in ARQ principle. Whereas, the TCP sack the selective acknowledgement variant of TCP, it works in the principle of this selective repeat ARQ, where explicitly we send the sack packet to indicate that which packet has been lost. And the sender retransmit that packets without sending the whole packet of the current window.

So, there are lots of such variants of TCP. And after that, many of the variants also came into practice. So, I am not going to the details of all those variants. If you are interested, you can go with that. The basic notion of TCP congestion control is this three phases, the slow start followed by the congestion avoidance, then the fast recovery, and the fast retransmit. And after that, people have incorporated few other optimizations. So, if you are interested, you can look into the RFCs to (Refer Time: 36:26) in better.

So, this gives us an broad overview of the congestion control algorithms, which are there in TCP. And we have given you a fairly detailed description of the transport layer protocol along with a focus on this TCP. So, as I have mentioned that a huge amount or even more than 90 percent of the internet traffic, it actually use a TCP protocol. And TCP is possibly the most widely deployed protocol in internet the transport layer protocol in internet, but well there are certain applications in the internet that use UDP, which does not support reliability, flow control, congestion control, as like TCP.

So, in the next class, we will look into the details of the UDP protocol.

So, thank you all for attending this class. See you again.