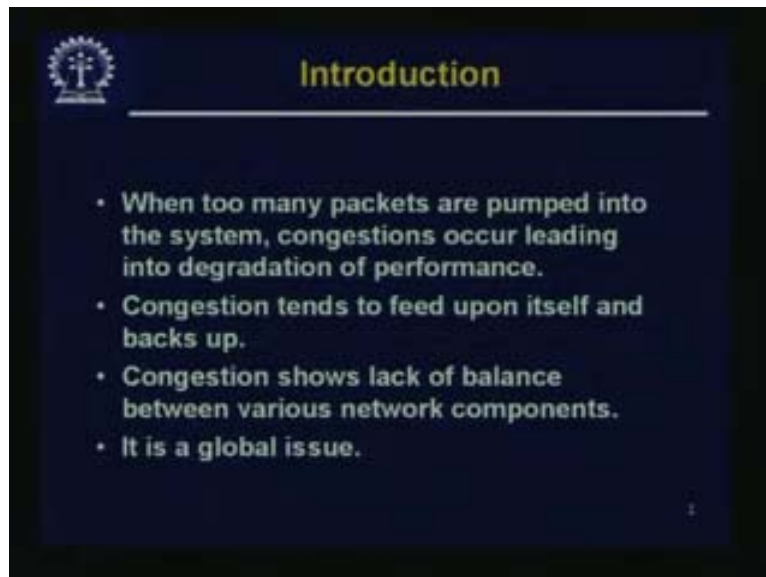


Computer Networks
Prof. S. Ghosh
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture-35

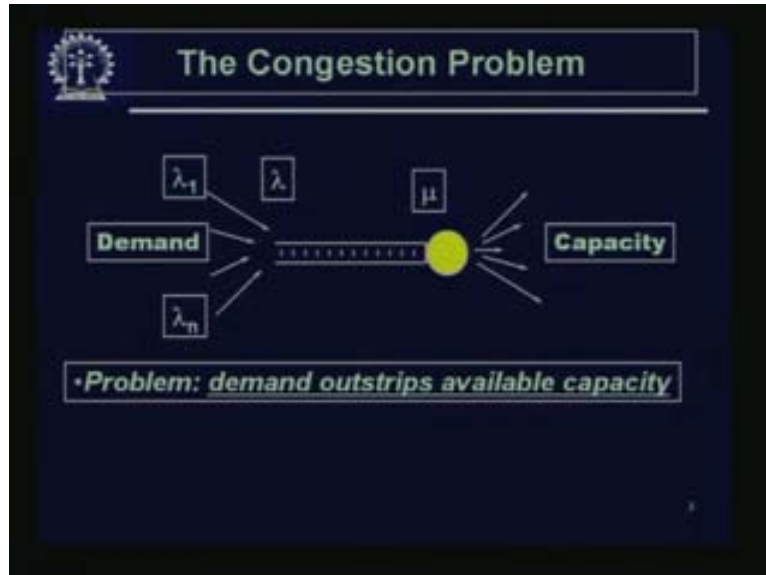
Good day. The topic for today is congestion control. The performance of a computer network depends on a large extent on the kind of congestion that is there in the network. **Once again this is a large topic. We will just touch upon some aspects of them.** One thing we know by now is, in general network, may be data network or inter network in particular although multimedia and other contents are coming in, this is a packet based network and a large data network that is announced where we make only the best effort of delivering a packet. Now, what exactly do you mean by best effort? Most of these best efforts depend on how you handle congestion.

(Refer Slide Time: 02:01-02:38)



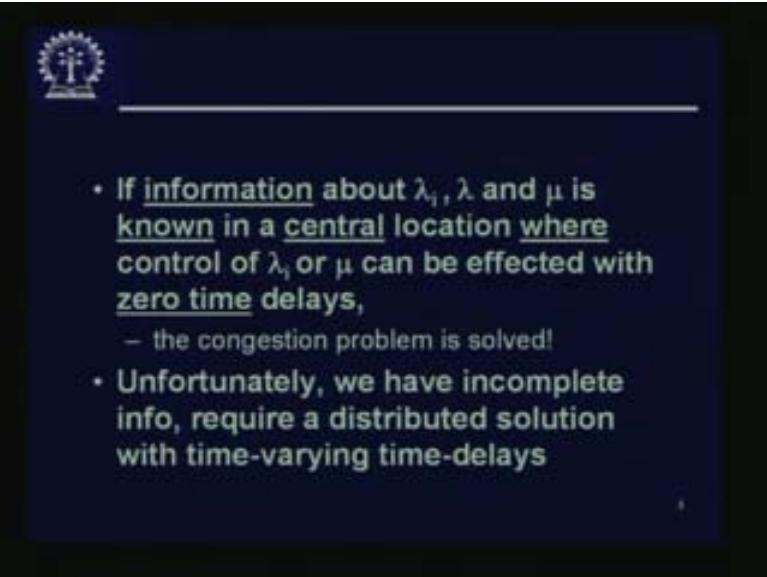
So, we know what congestion is. When too many packets are pumped into the system, congestion occurs leading into the degradation of the performance. Congestion tends to feed upon itself and backs up. Congestion shows lack of balance between various network components. Moreover it is a global issue because the congestion may happen in some intermediate router because of packets being pumped from various sources, so in that sense it is a global issue.

(Refer Slide Time: 02:39-04:18)



We have this intermediate node or channel etc and the demand is in the form of various sources pumping in packets at various rates namely λ_1 to λ_n and this is being serviced by the channel capacity or the router capacity at the rate μ and going to the various destinations. The problem is, the demand outstrips available capacity. So this is basically the congestion problem and added dimension to this problem comes from the fact that although these λ , μ etc are coming from the general queuing theory and for simple queuing theory these are the arrival rate and service rates etc. Although queues are there, but in general the statistics which the data networks follow are rather complicated. So traditionally telecom networks would follow some Poisson distribution with exponential interval time. But in data network it is seen that it follows something called as a self similar traffic or heavy tail distribution and that is a complex distribution. One of its key features is **bustiness**. That means data tends to come in busts and then there is comparatively long quotient period and again another burst comes. So, that is the problem which has to be handled and when several bust arrives at a node at the same time that particular node may get **overloaded**.

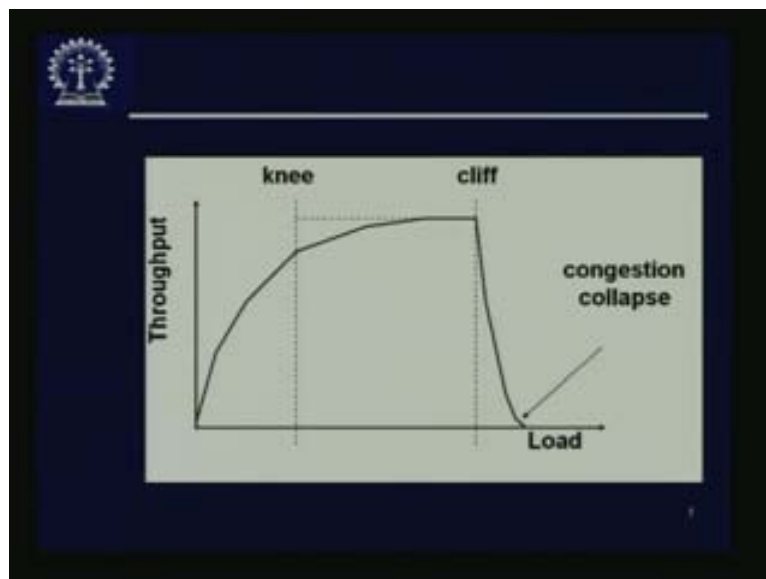
(Refer Slide Time: 04:19-04:45)



- If information about λ_i , λ and μ is known in a central location where control of λ_i or μ can be effected with zero time delays,
 - the congestion problem is solved!
- Unfortunately, we have incomplete info, require a distributed solution with time-varying time-delays

So, if information about λ_1 , λ and μ etc is known in a central location where control of λ_i and μ effected instantaneously with zero time delays then the congestion problem is solved. Unfortunately we cannot do that because we have incomplete information and we require a distributed solution with time varying time delays. This is what makes the problem a little difficult.

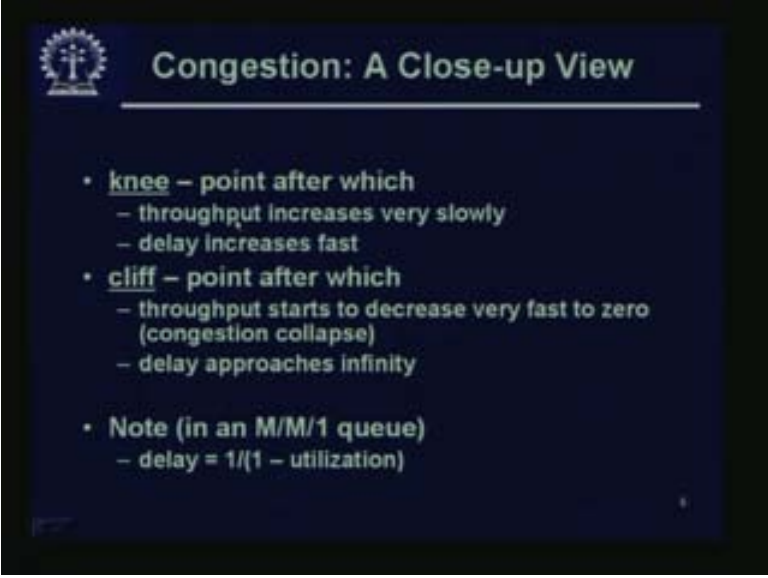
(Refer Slide Time: 04:46-05:43)



Already we have seen this kind of throughput versus load curve we have seen already when we saw **alocha networks**, ccna cd etc. But, in general this is what happens. As the load increases the throughput also keeps on increasing at the same rate and then it starts

sort of going down and it keeps going down because of the intermediate delays and other bottlenecks coming into the picture. And then there is an area where the throughput does not increase any longer. If you increase the load beyond that there is a catastrophic fall in the throughput. So this part is known as the knee and this part is known as the cliff and this catastrophic fall is called congestion collapse.

(Refer Slide Time: 05:44-06:12)



The slide is titled "Congestion: A Close-up View" and features a list of bullet points. The first bullet point defines the "knee" as the point after which throughput increases very slowly and delay increases fast. The second bullet point defines the "cliff" as the point after which throughput starts to decrease very fast to zero (congestion collapse) and delay approaches infinity. A third bullet point provides a note for an M/M/1 queue, stating that delay is equal to $1/(1 - \text{utilization})$.

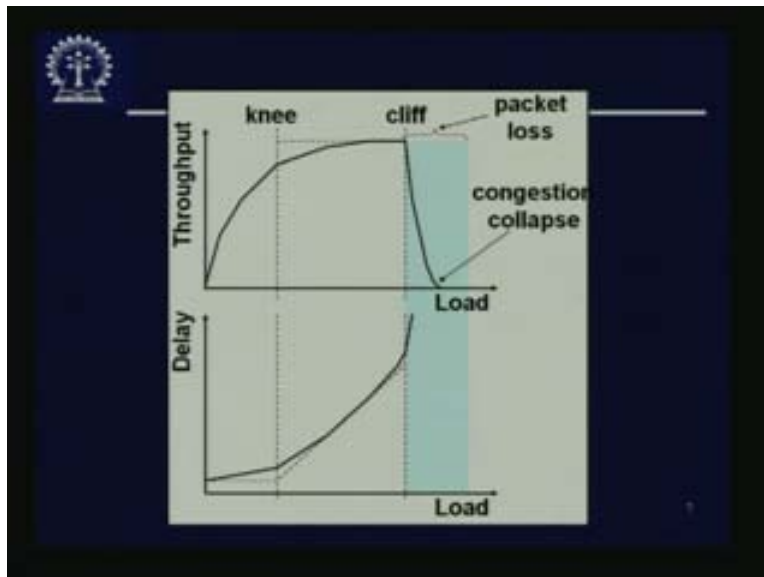
- knee – point after which
 - throughput increases very slowly
 - delay increases fast
- cliff – point after which
 - throughput starts to decrease very fast to zero (congestion collapse)
 - delay approaches infinity
- Note (in an M/M/1 queue)
 - delay = $1/(1 - \text{utilization})$

So, knee is the point after which throughput increases very slowly and delay increases fast. Cliff: Point after which throughput starts to decrease very fast to zero and this is congestion collapse and delay approaches infinity.

Note in an M/M/1 queue delay is equal to $1/(1 - \text{utilization})$.

It does not follow this kind of a simple formulation.

(Refer Slide Time: 06:13-06:42)



If this was the previous curve and as you plot the delay, the delay is low in the beginning and starts increasing. And in this area where there are lots of packet losses and there is a congestion collapse and there are throughput collapses the delay also becomes very high and it becomes a hyperbolic curve. So obviously you have to take all precautions not to fall into this area.

(Refer Slide Time: 06:43-07:03)

Congestion Control vs. Congestion Avoidance

- Congestion control goal
 - stay left of cliff
- Congestion avoidance goal
 - stay left of knee
- Right of cliff:
 - Congestion collapse

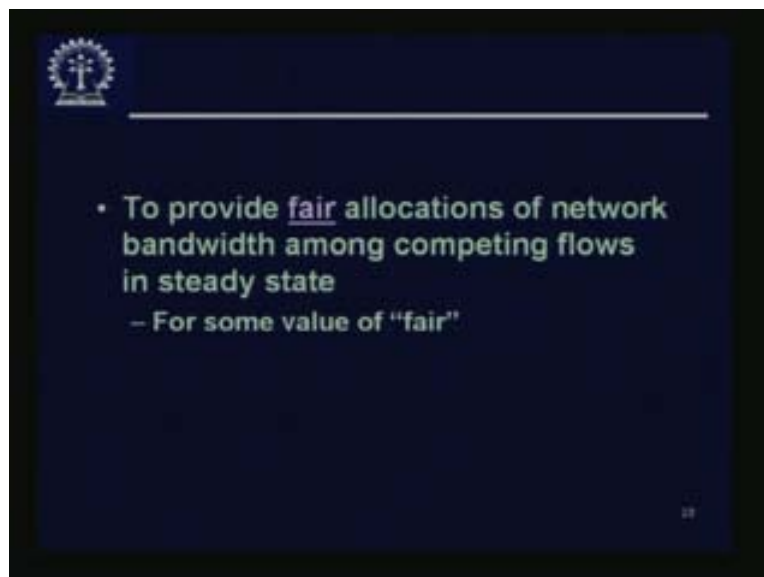
So, we talk about congestion control whose goal is to stay on the left of cliff that means not to go into the congestion collapse. Congestion avoidance: Goal is to stay left of the knee and right of cliff is of course the congestion collapse region.

(Refer Slide Time: 07:04-07:45)



So, the goal of congestion control is to guarantee stable operation of packet networks and a sub goal is to avoid congestion collapse. To keep networks working in an efficient manner, for example: high throughput, low loss, low delay high utilization are the goals not always achievable especially because we have distributed systems with insufficient information about the global picture but anyway that is there.

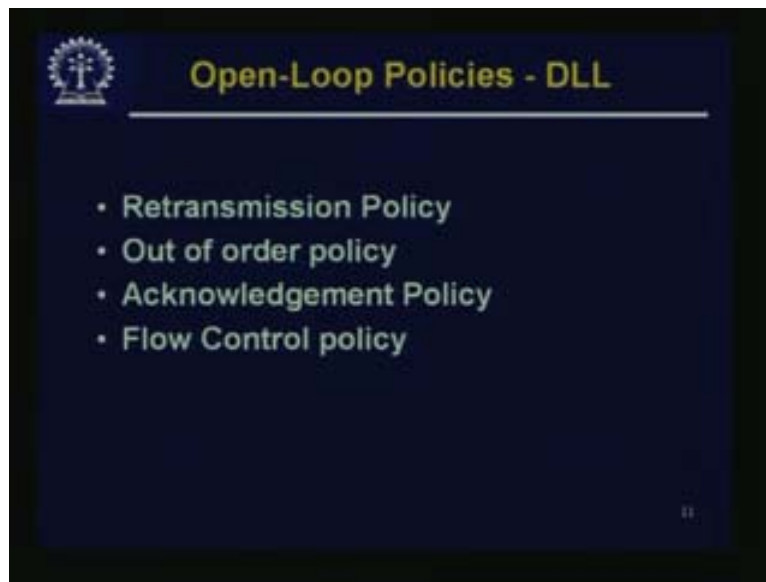
(Refer Slide Time: 07:46-09:15)



To provide fair allocation of network bandwidth among competing flows in steady state. So, there has to be some kind of fairness, Sometimes all are not taken as equal. First of all you must see, if there is congestion at an intermediate node, what would happen is that

there would be lot of packet loss over there. So, various packets from various sources would be lost and the delay would become high. Many of them were running a TCP protocol. Therefore the question is who would again start retransmitting. Hence what would happen is that, more packets would be lost and more and more packets will keep on getting pumped. This is just like a traffic jam, it starts at one place and then if the jam does not resolve soon then it becomes bigger and bigger and it starts getting pushed towards the source. So the overall network throughput goes down and people tend to push more packets. These are the kinds of scenario we would like to avoid.

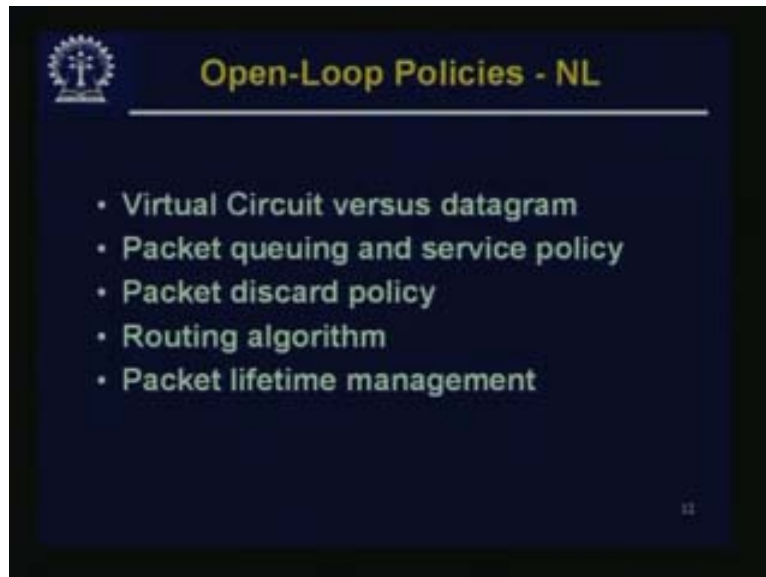
(Refer Slide Time: 09:16-10:47)



Now, there are various policies at various levels that we can take for congestion control. Let us look at the data link layer, the open loop policies. One is retransmission policy. How would you retransmit? One example of this re transmission policy is, suppose you have a Ethernet network with csmacd going on and then you have detected a collision, the question is, are you becoming persistent or non persistent or you do a random back off or exponential back off or what is your retransmission policy. Hence you will try again.

Similarly, there are other things like out of order policy. Out of order policy is when you receive a packet when it is out of order. Acknowledgement policy is, do you acknowledge or do not acknowledge it. For example, if you have an acknowledgement then the acknowledgement for each packet also takes up resources. So, if you acknowledge every packet then there is going to be as many packets sent as many acknowledgements. Therefore this is a lot of acknowledgement for the network and there is a high overhead. May be you take a policy of not acknowledging all the packets. It could be some kind of a flow control policy. Therefore we have seen some kind of flow control in TCP. We will look into more details and variations of it.

(Refer Slide Time: 10:48-13:18)



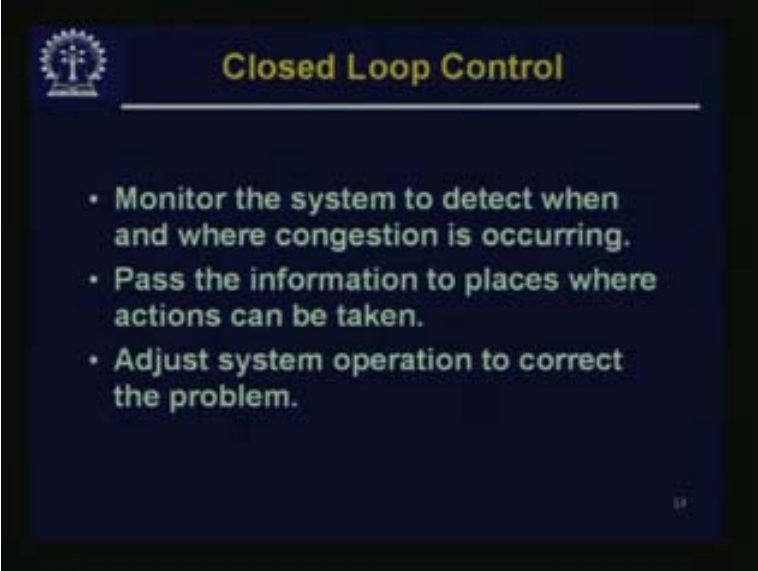
In the network layer you can have the virtual circuit versus datagram. This is an important issue and we will look at it in more detail when we discuss QOS and multimedia communication. Suppose there is a very important communication going on between two hosts, mission critical or whatever it may be, now they will be exchanging a lot of packets. Let us say the packets are flowing in only one direction so lot of packets will be sent and we want some premium service for this. If you want to give a premium service to this particular pair of nodes, may be they pay more or something then in that case you have to distinguish among the packets and assume that they form some kind of a flow. So you need to have a kind of virtual circuit between these two points in order to distinguish them. If all packets are on their own then that is a different kind of a situation where it will be more difficult to distinguish the flow between the two specific nodes. Virtual circuit versus datagram may be an important issue. Once again we will see more details of this when we look at RSVP, diffserv in the QOS when we discuss QoS a little bit more in detail.

Packet Queuing and Service policy means, in the router a number of packets may come and they are going to be serviced one by one and they are going to be put in a queue. Now the question is, do you put them in one queue or do you put them in several queues? Do you have the same priorities for all the queues or do different priorities for different queues and so on. Packet discard policy has to do with the buffer management of the router. If the buffer becomes full, which packet do you drop?

Routing algorithm: What kind of routing algorithm will you use?

Packet lifetime management: This means the lifetime of the packet is over and you drop the packets. These are important in the network layer. So far we were discussing about open loop policies.

(Refer Slide Time: 13:19-14:25)



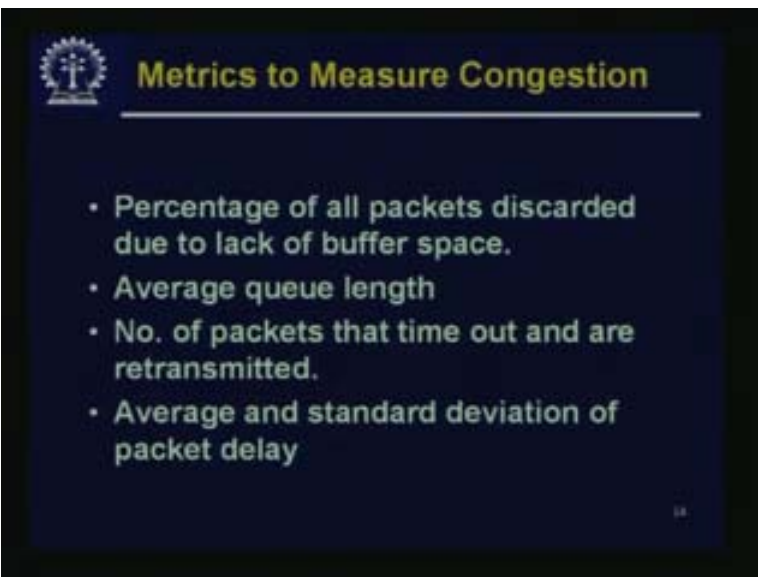
Closed Loop Control

- Monitor the system to detect when and where congestion is occurring.
- Pass the information to places where actions can be taken.
- Adjust system operation to correct the problem.

18

Now let us take a look at closed loop control. This means monitoring the system to detect when and where congestion is occurring. Pass the information to places where actions can be taken. Adjust system operation to correct the problem. This is more sophisticated and better. The point is, if one router in between is congested and now if it can identify the chief sources of trouble where it cannot handle a lot of packets, therefore if you could send a feedback back to the source so that he can control this behavior by sending less number of packets then the situation can be handled. Or you can adjust some system parameters, may be the window size in TCP etc. These are the examples of the kind of thing you can do with close loop control.

(Refer Slide Time: 14:26-15:26)



Metrics to Measure Congestion

- Percentage of all packets discarded due to lack of buffer space.
- Average queue length
- No. of packets that time out and are retransmitted.
- Average and standard deviation of packet delay

18

If you say that there is some congestion then we need to have some metrics for measuring congestion. Some examples are percentage of all packets discarded due to lack of buffer space. This may be one measure.

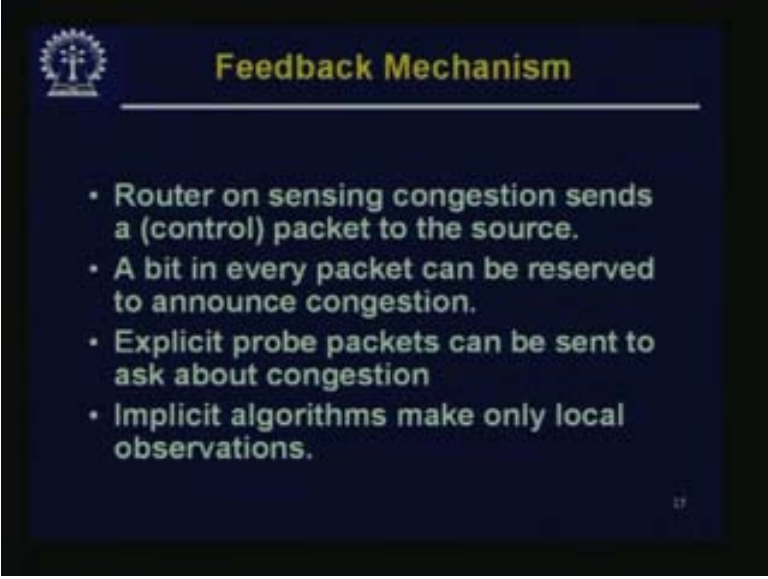
Average queue length in the buffer

No of packets that time out and are retransmitted

Average and standard deviation of packet delay

These may be metrics with which you measure congestion. So, if these metrics go beyond a certain level then you might decide that some congestion is taking place and you need to take some action in order to prevent the performance degradation in a sharp manner.

(Refer Slide Time: 15:27-16:07)

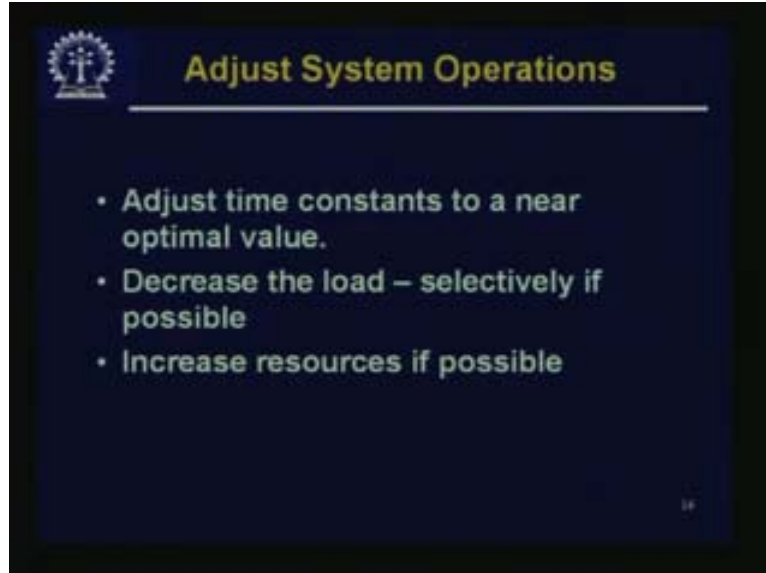


The slide is titled "Feedback Mechanism" in yellow text on a dark blue background. In the top left corner, there is a small circular logo featuring a stylized tower or monument. Below the title, there is a horizontal line. The main content consists of four bullet points, each preceded by a small white square. The text is in a light blue or white font. The slide number "17" is visible in the bottom right corner.

- Router on sensing congestion sends a (control) packet to the source.
- A bit in every packet can be reserved to announce congestion.
- Explicit probe packets can be sent to ask about congestion
- Implicit algorithms make only local observations.

Feedback mechanisms: It can be many. As we have mentioned, router on sensing congestion sends a control packet to the source. A bit in every packet can be reserved to announce congestion. Explicit probe packets can be sent to ask about congestion. Implicit algorithms make only local observations.

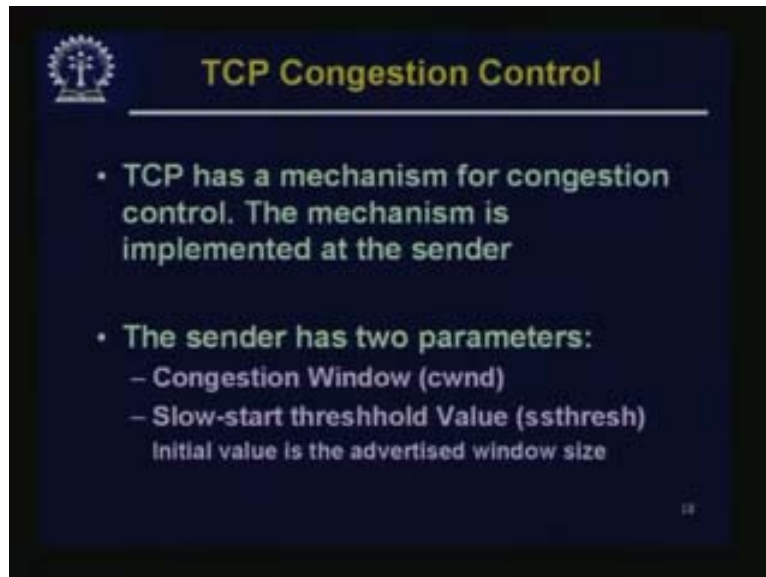
(Refer Slide Time: 16:08-16:44)



Then you can try adjusting system operations. Adjust time constants to a near optimal value. Decrease the load selectively if possible. May be if one source somehow can decrease then all the others can be served very well because it goes below a threshold. Increase resources if possible, this is usually difficult.

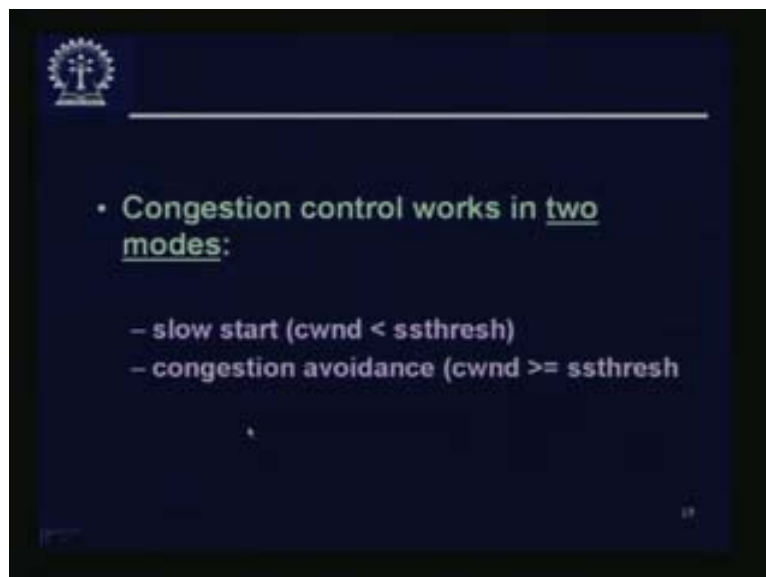
Now, let us look at the one aspect of congestion control which is very important and which is done by TCP all the time. TCP congestion control, if you remember uses a sliding window protocol. We have a window and a sender can send right up to the window size to the other side and it will wait for acknowledgements and he will keep on acknowledging and once he gets the acknowledgement the window will slide. This is about the basic TCP. What we are going to see now is some variance of TCP. Since TCP is a very important protocol and application protocols like FTP, STP, etc use TCP, a lot of important traffic on the net is actually carried on TCP and that is why whatever we do at the TCP level is very much important. And one of the chief tools for doing any congestion control by TCP is by adjusting the window size. So, there are various variants.

(Refer Slide Time: 17:57-18:29)



TCP has a mechanism for congestion control. The mechanism is implemented at the sender. The sender has two parameters, congestion window with a variable called cwnd and slow start threshold value with a variable called sssthresh. So, initial value is the advertised window size. So, with a TCP connection there is an advertisement of window size and this window size is taken as the initial sssthresh value.

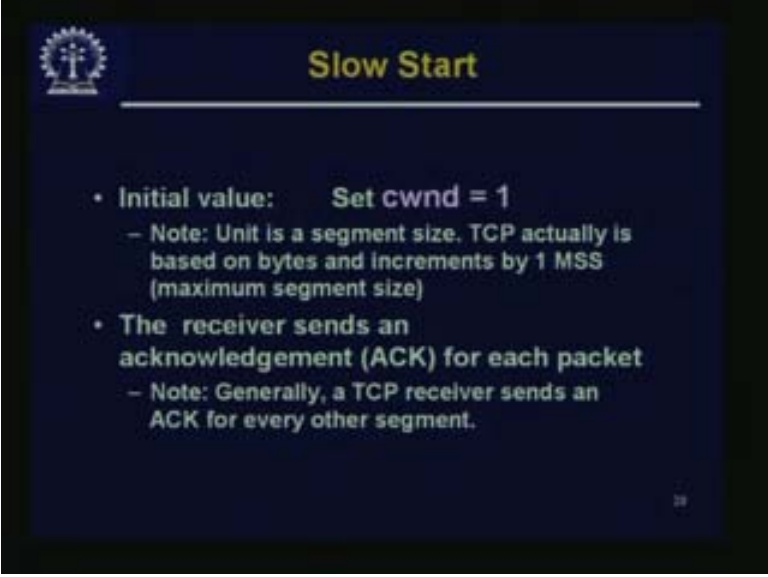
(Refer Slide Time: 18:30-19:16)



Congestion control works in two modes. One is slow start and the phase is slow at start when the cwnd value is less than sssthresh and congestion avoidance means that cwnd value is greater than equal to sssthresh. So, basically we are trying to figure out whether

we are on the left of knee or in the right of the knee. So, if you are on the right of the knee but left of the cliff we are going to be careful. If you are on the left of the knee and if things are going fine then we can try to increase the load stress to increase the overall throughput. This is the basic idea.

(Refer Slide Time: 19:17-20:06)



The slide is titled "Slow Start" in yellow text on a dark blue background. It features a small logo in the top left corner. The content is organized into two main bullet points, each with a sub-note.

- **Initial value: Set cwnd = 1**
 - Note: Unit is a segment size. TCP actually is based on bytes and increments by 1 MSS (maximum segment size)
- **The receiver sends an acknowledgement (ACK) for each packet**
 - Note: Generally, a TCP receiver sends an ACK for every other segment.

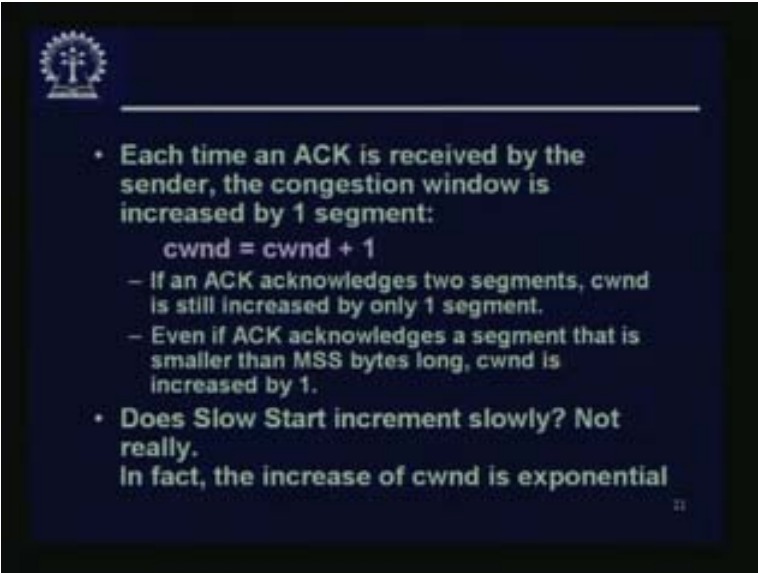
Knowing initial values in a slow start i.e. set cwnd is equal to 1. Naturally if the window size is small i.e. one so one unit will go and the acknowledgement will come back and then only something else will go from this side. That is why we are being very conservative and we are sending only a small bit of information.

Note: the unit is a segment size i.e. one of a second. TCP is actually based on bytes and increments by 1 MSS (Maximum Segment Size).

The receiver sends an acknowledgement (ACK) for each packet. So this is the slow start. So, the receiver must acknowledge every packet, so the first packet it receives it can send an acknowledgement.

Note: Generally a TCP receiver sends an acknowledgement (ACK) for every other segment.

(Refer Slide Time: 20:08-21:17)

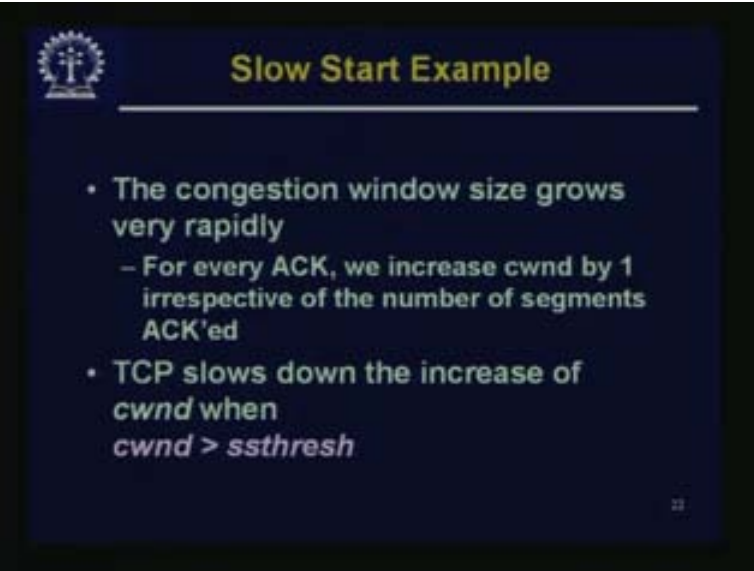


- Each time an ACK is received by the sender, the congestion window is increased by 1 segment:
$$cwnd = cwnd + 1$$
 - If an ACK acknowledges two segments, cwnd is still increased by only 1 segment.
 - Even if ACK acknowledges a segment that is smaller than MSS bytes long, cwnd is increased by 1.
- Does Slow Start increment slowly? Not really.
In fact, the increase of cwnd is exponential

Each time an ACK is received by the sender, the congestion window is increased by 1 segment. So what happens is that, the sender has sent one packet so it has got the acknowledgement, so actually the sender decides that things are fine and may do better. That means it is the increase in congestion window size (cwnd). So we increase cwnd by 1 i.e. cwnd is equal to cwnd plus 1.

We make cwnd is equal to 2. If an ACK acknowledges two segments cwnd is still increased by only 1 segment. That means for every ACK it increases by 1. If it acknowledges only one segment or two segments then cwnd is increased by one only. Actually the reason to acknowledge every other segment is to decrease the number of acknowledgements. Now, even if ACK acknowledges a segment that is smaller than MSS bytes long cwnd is still increased by one. So, at anytime you get an ACK you increase cwnd by one when you are in the slow start phase. Although it starts slowly does it increment slowly? Not really. In fact, the increase of cwnd is exponential.

(Refer Slide Time: 21:18-21:35)



Slow Start Example

- The congestion window size grows very rapidly
 - For every ACK, we increase *cwnd* by 1 irrespective of the number of segments ACK'ed
- TCP slows down the increase of *cwnd* when *cwnd* > *ssthresh*

22

The congestion window size grows very rapidly, *cwnd* rises very rapidly. For every ACK we increase *cwnd* by 1 irrespective of the number of segments ACK'ed. The TCP slows down the increase of *cwnd* when *cwnd* > *ssthresh*.

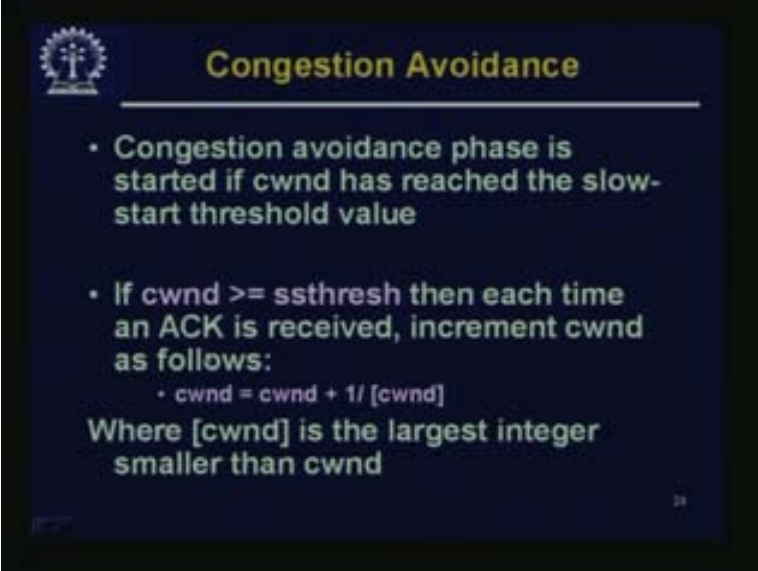
(Refer Slide Time: 21:36-22:24)



As you can see, suppose if it sends one segment it receives one acknowledgement, the *cwnd* is increased from one to two. Now you can send two segments, segment two and segment three. It will get back the acknowledgement for segment two and acknowledgement for segment three. Now *cwnd* has become 4. It will send 4, 5, 6, etc, so three of them it has sent and the acknowledgement for 4, 5, 6 will come. Now *cwnd* has

become 7 and it will send more. You can see here, 1, 2, 4, 7 is increasing quite fast because for each acknowledgement it is increased by one and when you are sending so many segments at a group you will get many acknowledgements. Therefore cwnd is increasing exponentially.

(Refer Slide Time: 22:25-23:00)



The slide is titled "Congestion Avoidance" in yellow text on a dark blue background. It features a small logo in the top left corner. The content includes two bullet points and a definition. The first bullet point states that the congestion avoidance phase starts when cwnd reaches the slow-start threshold value. The second bullet point states that if cwnd is greater than or equal to ssthresh, then each time an ACK is received, cwnd is incremented according to the formula: $cwnd = cwnd + 1 / [cwnd]$. Below this, it defines [cwnd] as the largest integer smaller than cwnd. A small number "28" is visible in the bottom right corner of the slide.

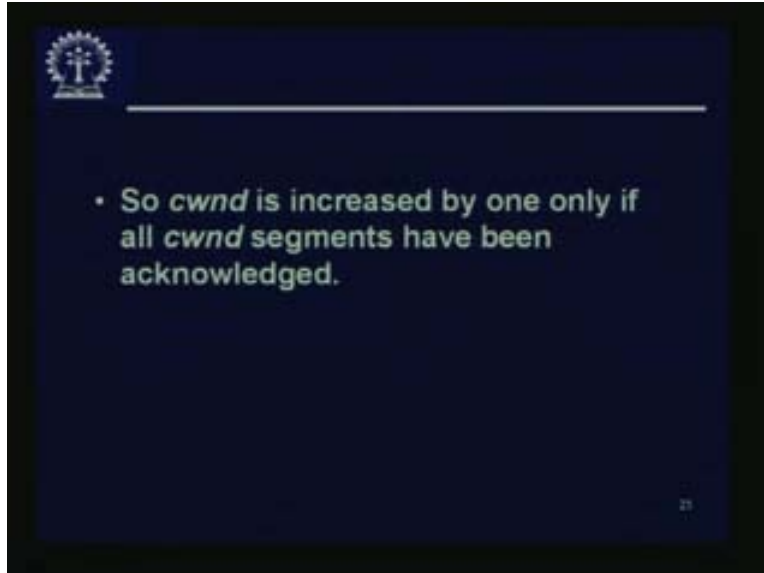
- Congestion avoidance phase is started if cwnd has reached the slow-start threshold value
- If $cwnd \geq ssthresh$ then each time an ACK is received, increment cwnd as follows:
 - $cwnd = cwnd + 1 / [cwnd]$

Where $[cwnd]$ is the largest integer smaller than cwnd

Congestion avoidance phase is started if cwnd has reached the slow start threshold value (ssthresh).

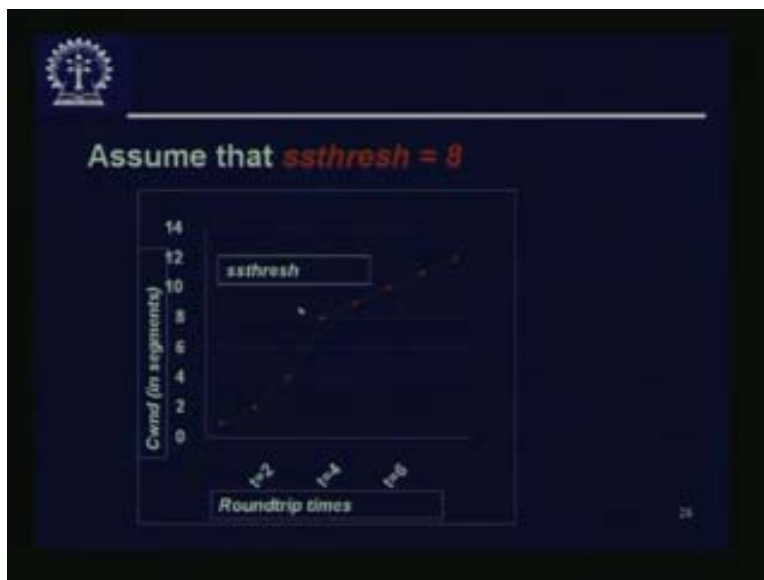
If $cwnd \geq ssthresh$. Then each time an ACK is received, increment cwnd as follows: i.e. $cwnd = cwnd + 1 / [cwnd]$ where $[cwnd]$ is the minimum or the larger integer and is smaller than cwnd. So this is only increased by a fraction while sending. Of course you will not send a fraction and whatever be the current cwnd value that is **floored** that many segments you can send.

(Refer Slide Time: 23:01-23:21)



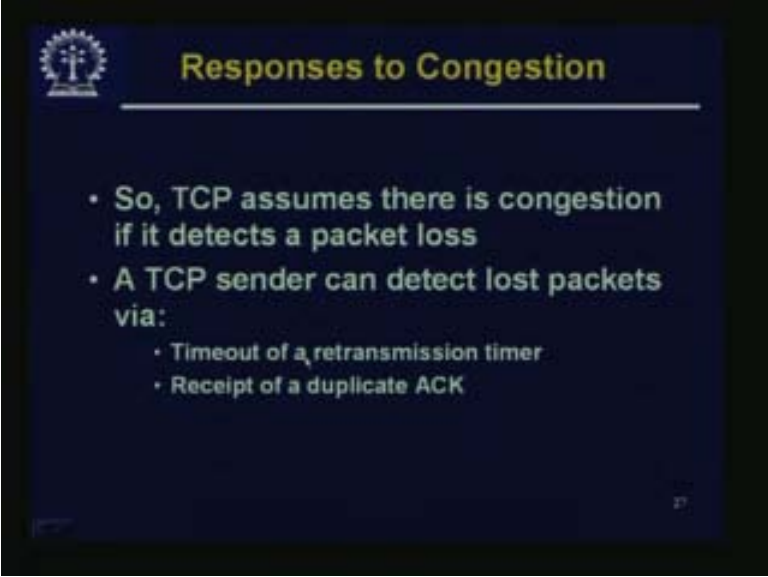
So, *cwnd* is increased by one only if all *cwnd* segments have been acknowledged. That means, if all the *cwnd* have been sent or acknowledged, then *cwnd* increases only by one. So we are very cautious while we move to the right of the knee.

(Refer Slide Time: 23:22-23:42)



So, assume that *sssthresh* is 8 therefore what will happen is, round-trip time is equal to 2, 4, 6, etc. As time is going so *cwnd* is first increased exponentially. It reaches the *sssthresh* value and then it increases slowly.

(Refer Slide Time: 23:43-24:26)

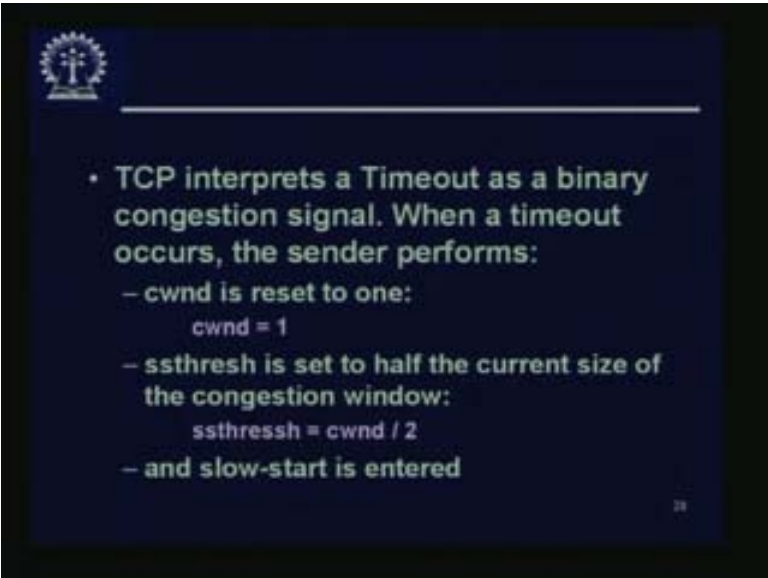


The slide is titled "Responses to Congestion" and features a list of bullet points. The first bullet point states that TCP assumes congestion if it detects a packet loss. The second bullet point states that a TCP sender can detect lost packets via two methods: a timeout of a retransmission timer and the receipt of a duplicate ACK. The slide includes a logo in the top left corner and a small number '27' in the bottom right corner.

- So, TCP assumes there is congestion if it detects a packet loss
- A TCP sender can detect lost packets via:
 - Timeout of a retransmission timer
 - Receipt of a duplicate ACK

TCP assumes there is congestion if it detects a packet loss. Now, what is the response to congestion? TCP assumes that if congestion detects a packet loss a TCP sender can detect a lost packet via timeout of a retransmission timer or receipt of a duplicate ACK. Duplicate ACK has been received which means that previously may be some acknowledgement has been dropped and there is a duplicate ACK. So, when something is dropped it means that there may be congestion. Now there are different ways to respond to this congestion.

(Refer Slide Time: 24:27-25:02)



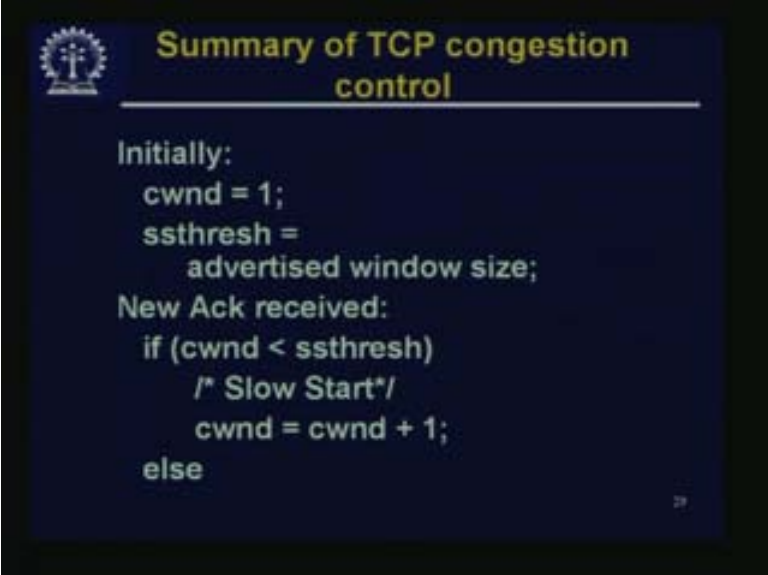
The slide is titled "Responses to Congestion" and features a list of bullet points. The first bullet point states that TCP interprets a Timeout as a binary congestion signal. When a timeout occurs, the sender performs three actions: cwnd is reset to one (with the formula $cwnd = 1$), ssthresh is set to half the current size of the congestion window (with the formula $ssthresh = cwnd / 2$), and slow-start is entered. The slide includes a logo in the top left corner and a small number '28' in the bottom right corner.

- TCP interprets a Timeout as a binary congestion signal. When a timeout occurs, the sender performs:
 - cwnd is reset to one:
 $cwnd = 1$
 - ssthresh is set to half the current size of the congestion window:
 $ssthresh = cwnd / 2$
 - and slow-start is entered

One is, TCP interprets a time-out as a binary congestion signal which means there is congestion as soon as there is a timeout. Therefore when the sender performs cwnd is now reset to one i.e. cwnd is equal to 1 so once again it becomes very conservative. ssthresh is set to half the current size of the congestion window Ssthresh is equal to cwnd/2.

Before sending it to one whatever be the cwnd value you divide it by two and make it the new threshold value and enter the slow start again.

(Refer Slide Time: 25:03-25:20)



The slide is titled "Summary of TCP congestion control" in yellow text on a dark blue background. It contains the following logic:

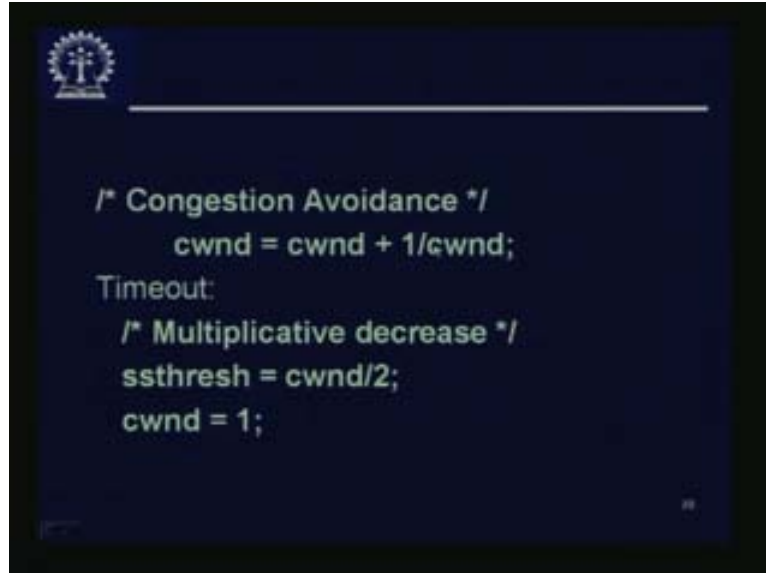
```
Initially:  
  cwnd = 1;  
  ssthresh =  
    advertised window size;  
New Ack received:  
  if (cwnd < ssthresh)  
    /* Slow Start*/  
    cwnd = cwnd + 1;  
  else
```

A small logo is visible in the top left corner of the slide, and a small number "29" is in the bottom right corner.

So initially: cwnd is equal to 1 i.e. cwnd is equal to 1 and ssthresh is equal to advertised window size.

New acknowledgement (ACK) is received: If (cwnd < ssthresh)
/* Slow Start*/ cwnd is equal to cwnd plus 1; else

(Refer Slide Time: 25:21-25:36)

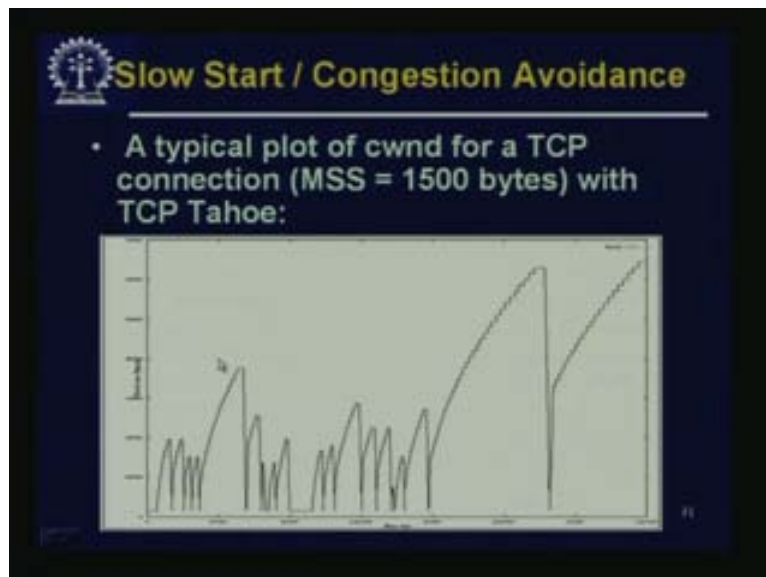


Congestion avoidance cwnd is equal to $cwnd + 1/cwnd$

Cwnd is equal to $cwnd + 1/cwnd$.

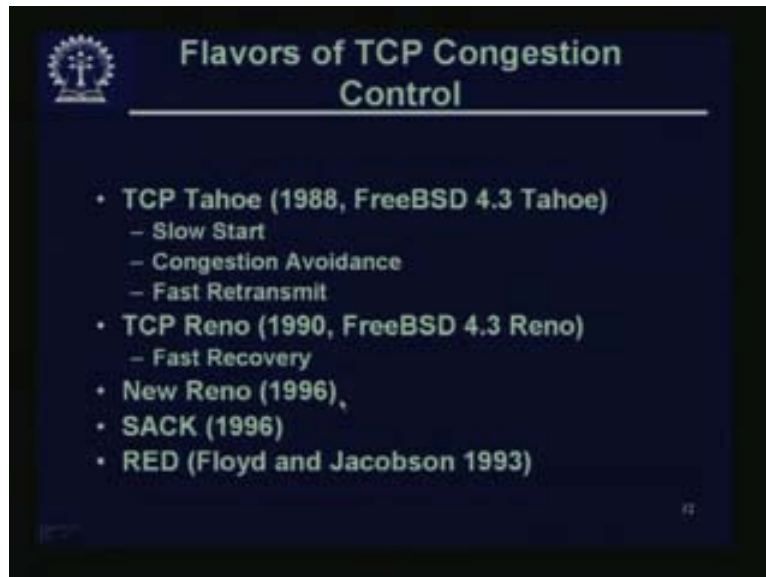
If there is timeout it is multiplicative decrease i.e. ssthresh is equal to $cwnd / 2$ and cwnd is equal to 1.

(Refer Slide Time: 25:37-25:58)



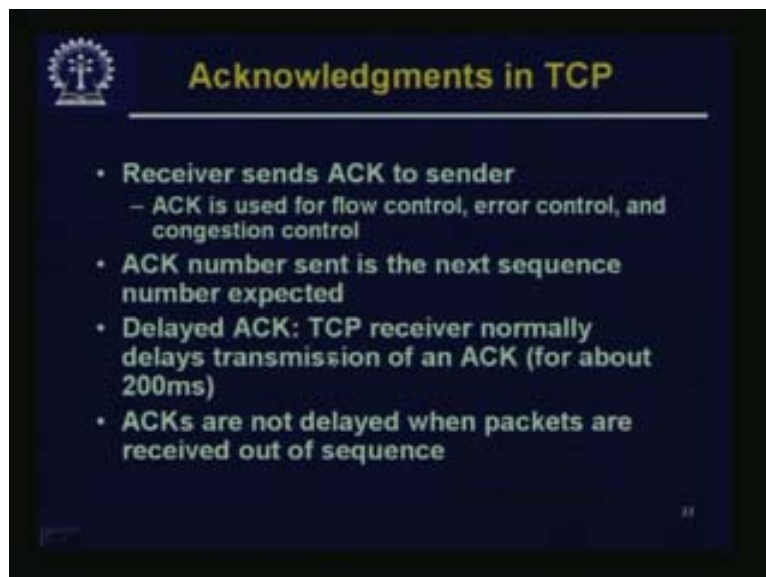
This is the typical plot of cwnd for a TCP connection (MSS is equal to 1500 bytes) with TCP Tahoe: TCP Tahoe is one flavor of TCP we have been discussing. We will discuss about some other flavor also. So, if cwnd goes on increasing, decreasing and then after sometime again increasing while things are good then this may be a typical plot.

(Refer Slide Time: 25:59-26:32)



TCP Tahoe uses one flavor,
slowstart for every acknowledgement
Congestion avoidance that means only beyond the ssthresh it increases slowly
Fast retransmit.
In TCP Reno there is also another version of TCP
Uses fast recovery
And then there are some versions like New Reno, SACK, RED, etc.

(Refer Slide Time: 26:33-27:36)

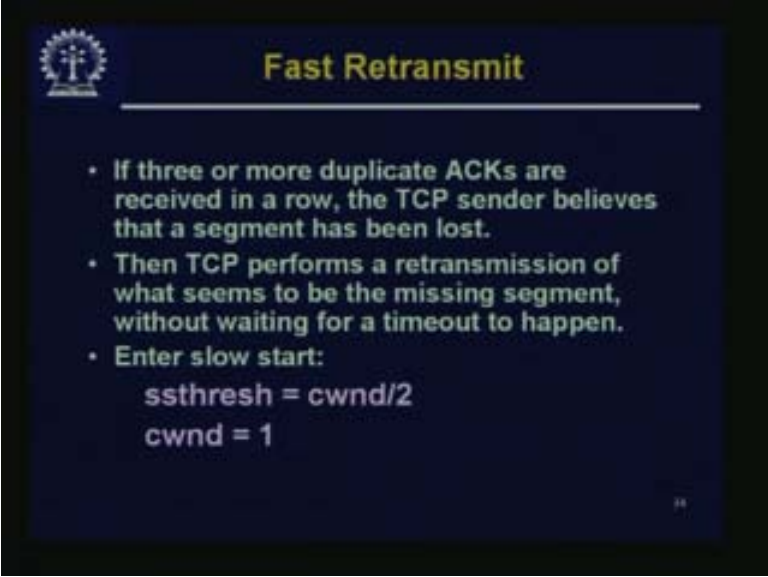


Acknowledgements in TCP: Receiver sends acknowledgement (ACK) to sender

Acknowledgement is used for flow control, error control and congestion control. In error control if the acknowledgement is not received then you send a retransmit. In congestion control we find that ACK is used for controlling this. ACK number sent is the next sequence number expected.

Delayed ACK: TCP receiver normally delays transmissions of an ACK for about 200 ms because it allows the packets to arrive thinking that it can send less number of acknowledgements this way, and ACKs are not delayed when packets are received out of sequence i.e. a little out of ordinary, may be they came from two different paths, so you do not delay the ACK but send it immediately.

(Refer Slide Time: 27:38-28:42)



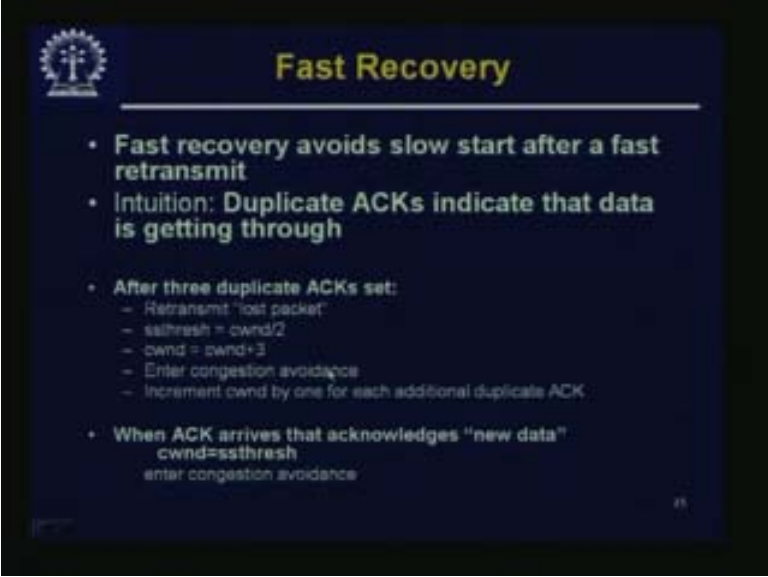
The slide is titled "Fast Retransmit" in yellow text on a dark blue background. It features a list of three bullet points in white text. The first bullet point states that if three or more duplicate ACKs are received in a row, the TCP sender believes a segment has been lost. The second bullet point states that then TCP performs a retransmission of what seems to be the missing segment without waiting for a timeout. The third bullet point states that it enters slow start, followed by the formulas $ssthresh = cwnd/2$ and $cwnd = 1$. A small logo is in the top left corner, and a small number "28" is in the bottom right corner.

Fast Retransmit

- If three or more duplicate ACKs are received in a row, the TCP sender believes that a segment has been lost.
- Then TCP performs a retransmission of what seems to be the missing segment, without waiting for a timeout to happen.
- Enter slow start:
 $ssthresh = cwnd/2$
 $cwnd = 1$

Now fast retransmit, if you remember that the TCP RENO uses fast retransmit. If three or more duplicate ACKs are received in a row, the TCP sender believes that a segment has been lost. This means ACKs have come meaning some earlier packets are gone. This means, possibly the later packets or segments may be lost. So what it does is, without waiting for the timeout to occur for this particular segment which has been sent it assumes that it has been lost and it sends one more again. TCP performs a retransmission of what seems to be the missing segment without waiting for a timeout to happen and then it enters slow start. That means it brings down the multiplicative decrease of $ssthresh$ and sets the $cwnd$ to 1 i.e. $ssthresh$ is equal to $cwnd/2$ $cwnd$ is equal to 1. This is fast retransmit.

(Refer Slide Time: 28:43-29:50)

A presentation slide titled "Fast Recovery" with a blue background and white text. The slide contains a list of bullet points explaining the Fast Recovery process in TCP. In the top left corner, there is a small circular logo featuring a gear and a cross. The slide number "28" is located in the bottom right corner.

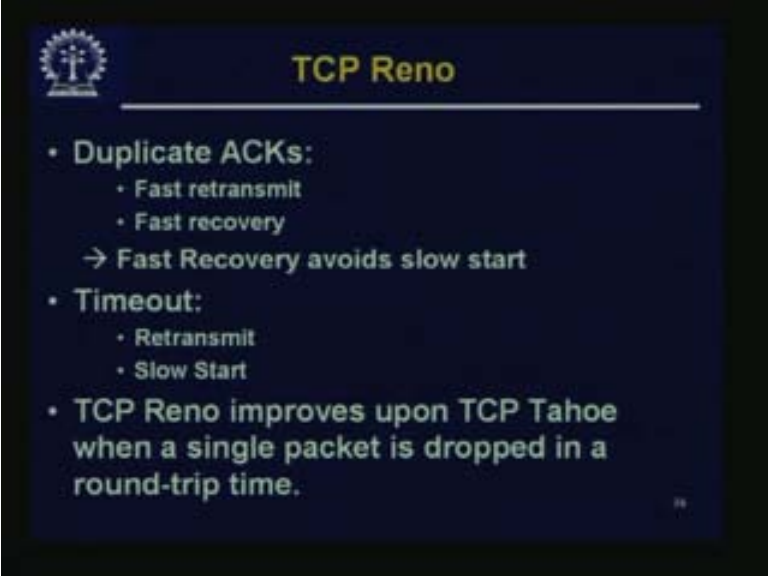
Fast Recovery

- Fast recovery avoids slow start after a fast retransmit
- Intuition: Duplicate ACKs indicate that data is getting through
- After three duplicate ACKs set:
 - Retransmit "lost packet"
 - $ssthresh = cwnd/2$
 - $cwnd = cwnd + 3$
 - Enter congestion avoidance
 - Increment $cwnd$ by one for each additional duplicate ACK
- When ACK arrives that acknowledges "new data"
 - $cwnd = ssthresh$
 - enter congestion avoidance

28

In Fast Recovery the slow start is avoided after a fast retransmit. That means after a fast retransmit intuition Duplicate ACKs indicate that data is still getting through or at least the duplicate ACKs are through. After three duplicate ACKs set retransmit lost packet, i.e. decrease $ssthresh$ to half so $ssthresh$ is equal to $cwnd/2$ but $cwnd$ is equal to $cwnd$ plus 3 and then you enter congestion avoidance. So increment $cwnd$ by one for each additional duplicate ack. This is a fast recovery but then after this you enter the congestion avoidance. That means, basically this is trying to tune the performance of TCP to get the maximum throughput without causing any congestion. When ACK arrives that acknowledges new data $cwnd$ is equal to $ssthresh$. After that we enter the congestion avoidance. So this is fast recovery.

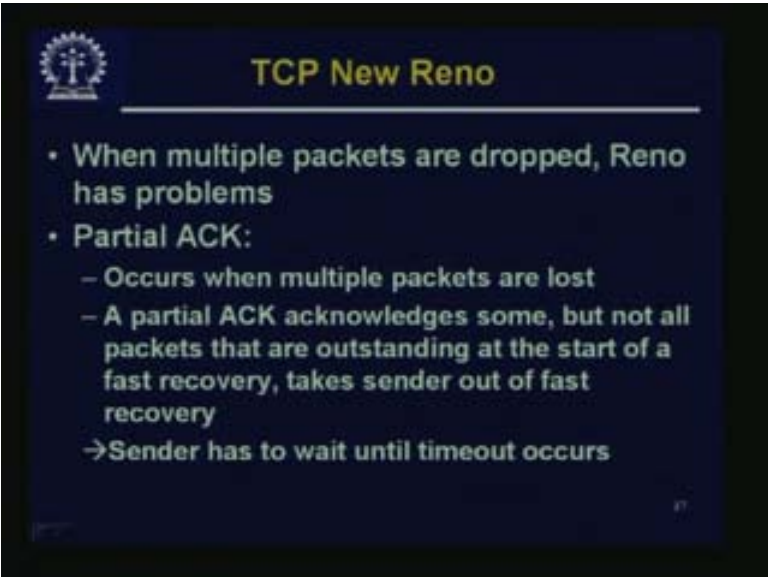
(Refer Slide Time: 29:51-30:18)

A presentation slide titled "TCP Reno" with a logo in the top left corner. The slide contains a bulleted list of features and behaviors of TCP Reno.

- Duplicate ACKs:
 - Fast retransmit
 - Fast recovery
 - Fast Recovery avoids slow start
- Timeout:
 - Retransmit
 - Slow Start
- TCP Reno improves upon TCP Tahoe when a single packet is dropped in a round-trip time.

TCP RENO: For Duplicate ACKs it does fast retransmit and fast recovery. Fast recovery avoids slow start. And if there is a time-out you retransmit and go to slow start. TCP RENO improves upon TCP Tahoe when a single packet is dropped in a round-trip time. But if multiple packets are dropped then of course the TCP RENO cannot handle that and for that we have a TCP NEW RENO.

(Refer Slide Time: 30:19-30:42)

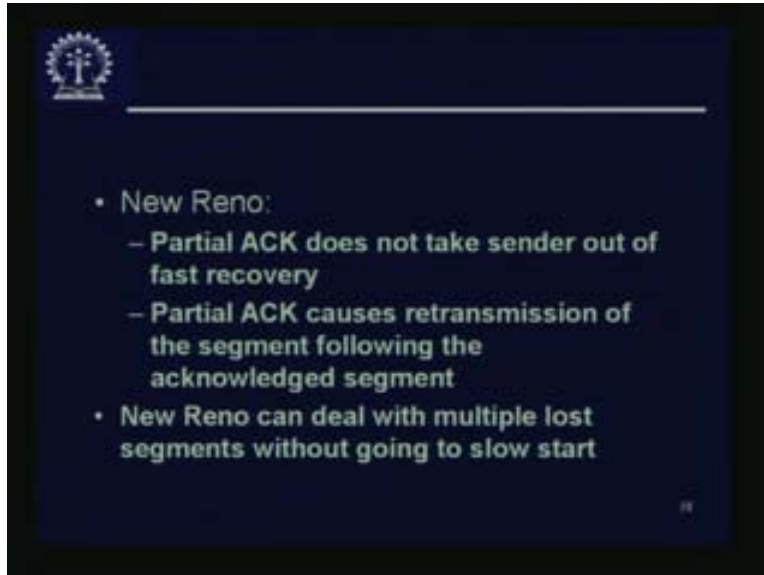
A presentation slide titled "TCP New Reno" with a logo in the top left corner. The slide contains a bulleted list of features and behaviors of TCP New Reno.

- When multiple packets are dropped, Reno has problems
- Partial ACK:
 - Occurs when multiple packets are lost
 - A partial ACK acknowledges some, but not all packets that are outstanding at the start of a fast recovery, takes sender out of fast recovery
 - Sender has to wait until timeout occurs

When multiple packets are dropped RENO has problems.

Partial ACK: Occurs when multiple packets are lost. A partial ACK acknowledges some but not all packets that are outstanding at the start of a fast recovery, takes sender out of fast recovery. The sender has to wait until time-out occurs.

(Refer Slide Time: 30:43-30:58)

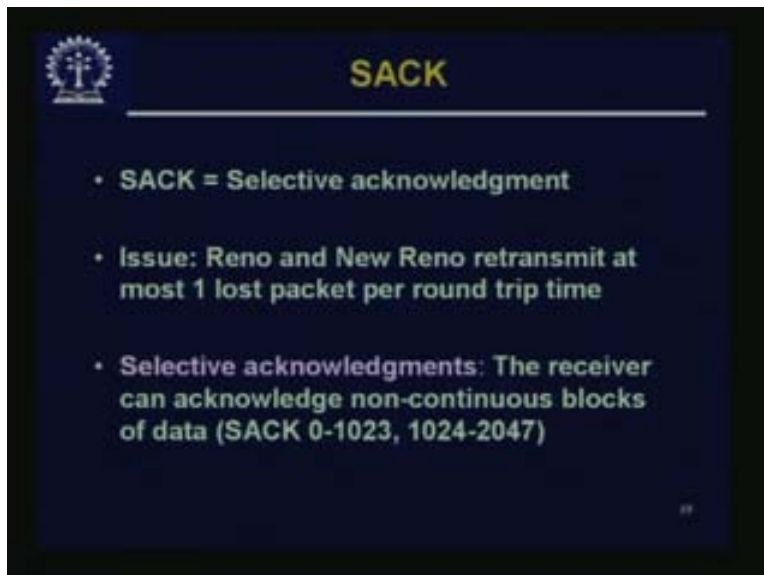


The slide features a dark blue background with a white logo in the top left corner. The title 'New Reno' is centered at the top. Below the title, there is a list of bullet points in white text.

- New Reno:
 - Partial ACK does not take sender out of fast recovery
 - Partial ACK causes retransmission of the segment following the acknowledged segment
- New Reno can deal with multiple lost segments without going to slow start

In new RENO partial ACK does not take sender out of fast recovery. Partial ACK causes retransmission of the segment following the acknowledged segment. New RENO can deal with multiple lost segments without going to slow start.

(Refer Slide Time: 30:59-31:42)



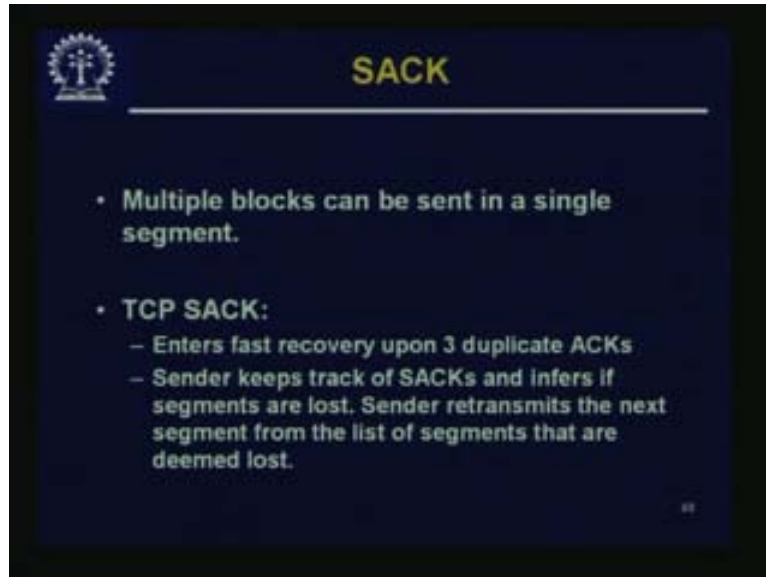
The slide features a dark blue background with a white logo in the top left corner. The title 'SACK' is centered at the top in yellow text. Below the title, there is a list of bullet points in white text.

- SACK = Selective acknowledgment
- Issue: Reno and New Reno retransmit at most 1 lost packet per round trip time
- Selective acknowledgments: The receiver can acknowledge non-continuous blocks of data (SACK 0-1023, 1024-2047)

There is a selective acknowledgement (SACK). Here you can selectively acknowledge. In an original TCP when you give an acknowledgement, that is the next segment you are expecting and all segments before that are acknowledged. Here you can give selective acknowledgement stating that you have got all these but not that particular one.

Issue: Reno and new Reno retransmit at most one lost packet per round-trip time. Selective acknowledgement: The receiver can acknowledge non continuous blocks of data. That means SACK selective acknowledgement of 0 to 1023, 1024-2047 and so on

(Refer Slide Time: 31:43-32:07)

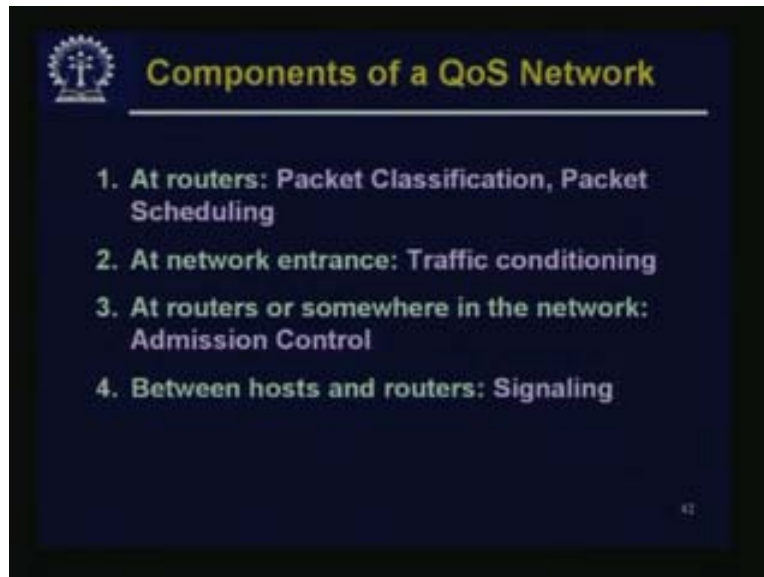


Multiple blocks can be sent in a single segment.

TCP SACK enters fast recovery upon three duplicate ACKs. Sender keeps track of SACKs and infers if segments are lost. Sender retransmits the next segment from the list of segment that is deemed to be lost like fast retransmit.

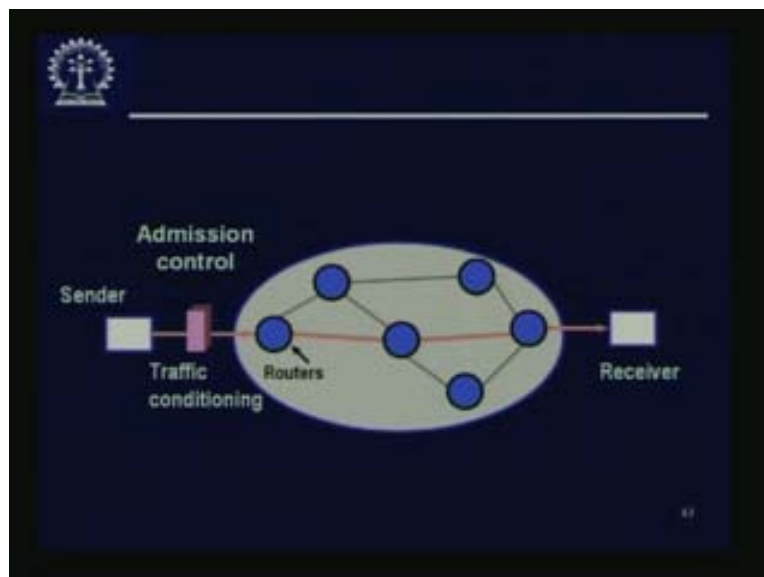
To improve the performance of TCP, there are two competing demands here. One is that we have to maximize the throughput and if you can maximize the throughput, naturally the overall delay, congestion etc will be small and at the same time you will get your job done faster. But in order to push this maximum throughput we should not get into congestion, a collapse so we try to guard against that. These are the versions or various flavors of TCP for doing that. So, we have looked at TCP. Now we are going to look at some other topic once again associated with congestion control and the other one is traffic engineering. That means, can you shape or can you handle your traffic in a particular way so that congestion is less likely to occur.

(Refer Slide Time: 33:13-33:43)



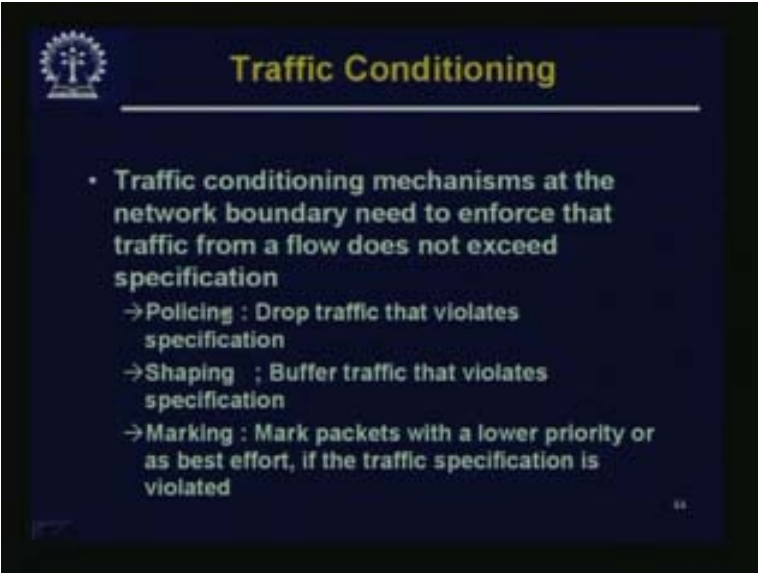
All these build to give a quality of service in a network. And at routers these may depend on Packet Classification and Packet Scheduling. At network entrance it may depend on traffic conditioning. At routers or somewhere in the network you may do admission control. Between hosts and routers you may do signaling. So these are the different components of QoS of a network.

(Refer Slide Time: 33:44-34:00)



So, let us say you have a sender and receiver here, these are the intermediate routers then you can do the traffic conditioning at the edge of the network. You can also do admission control here or somewhere else. So, these are the different components.

(Refer Slide Time: 34:01-35:02)

A presentation slide titled "Traffic Conditioning" with a blue background and yellow text. The title is at the top right, and a small logo is at the top left. A horizontal line separates the title from the content. The content is a bulleted list of traffic conditioning mechanisms. The first bullet point is "Traffic conditioning mechanisms at the network boundary need to enforce that traffic from a flow does not exceed specification". It has three sub-points: "→ Policing : Drop traffic that violates specification", "→ Shaping : Buffer traffic that violates specification", and "→ Marking : Mark packets with a lower priority or as best effort, if the traffic specification is violated". A small number "44" is in the bottom right corner.

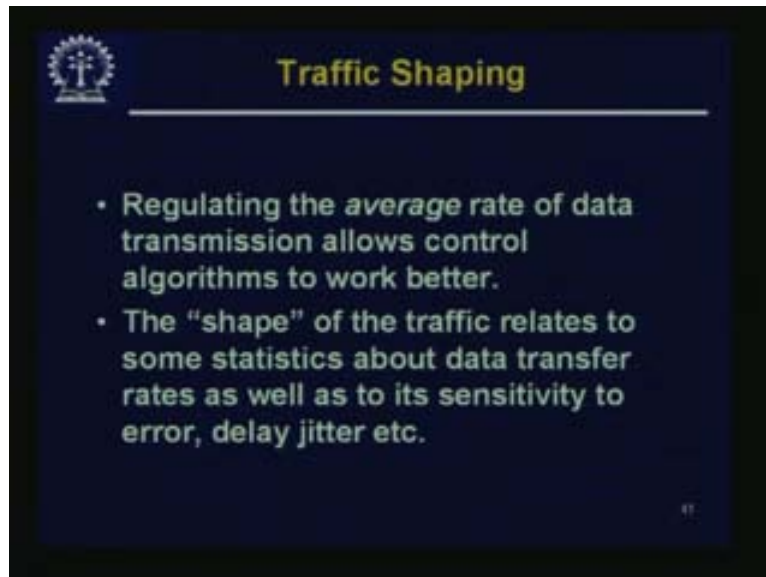
Traffic Conditioning

- Traffic conditioning mechanisms at the network boundary need to enforce that traffic from a flow does not exceed specification
 - Policing : Drop traffic that violates specification
 - Shaping : Buffer traffic that violates specification
 - Marking : Mark packets with a lower priority or as best effort, if the traffic specification is violated

44

Traffic conditioning mechanisms at the network boundary need to enforce that traffic from a flow does not exceed specification. So, we will look later at what kind of specifications we are talking about, what kinds of things people may agree on, or negotiate about that what are the parameters. But suppose from some source we had negotiated certain parameters and we find that the source is not sticking to that parameters and it is going out of that then we have to do some policing. So, policing is a drop traffic that violates the specifications. The specification as was agreed between the service provider and the sender. Shaping means the buffer traffic that violates specifications. Marking means mark packets with a lower priority or as best effort, if the traffic specification is violated.

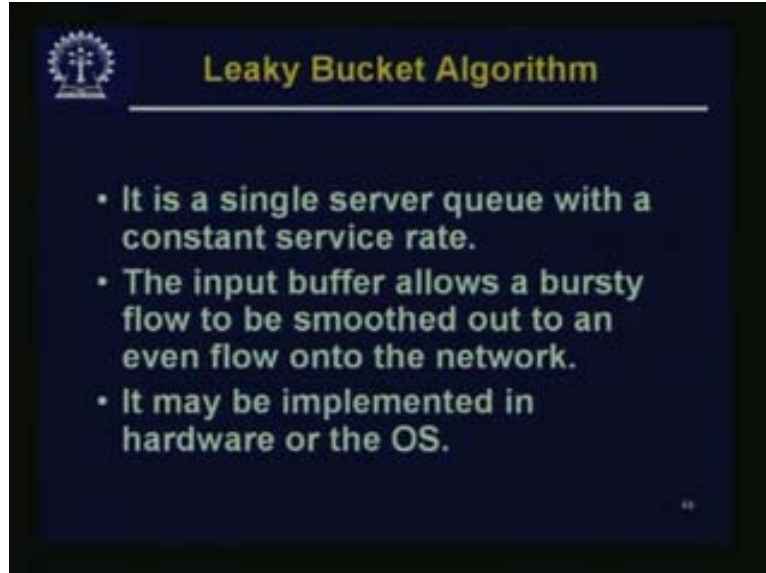
(Refer Slide Time: 35:03-36:56)



Let us look at Traffic shaping first. Regulating the average rate of data transmission allows control algorithms to work better. So this is to be understood. As I mentioned earlier, in computer network specifically data networks or internet traffic etc they are inherently very bursty in nature. When it comes it comes in one big bunch and then for long periods there may be no traffic.

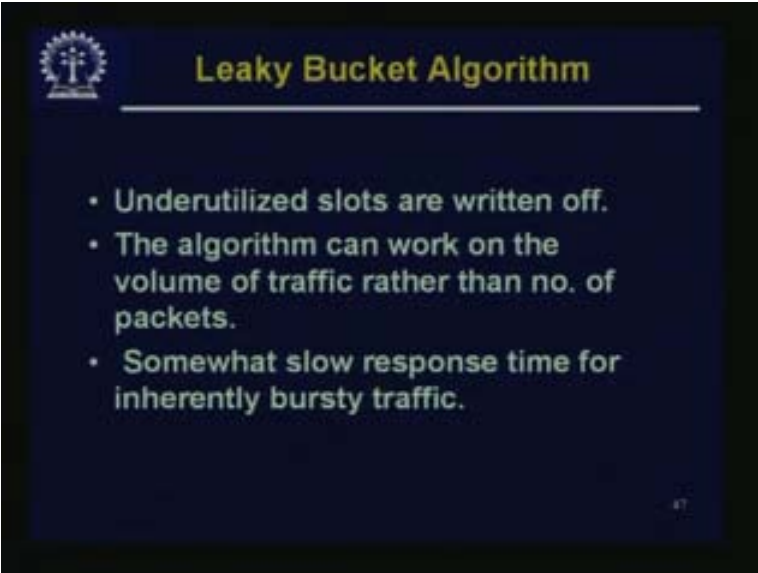
Now the trouble is, if the burst peak to the average ratio may be as much as 1:1000, we have to accordingly design your buffer and other network provisioning. So, we have to decide upon whether we are doing it for the peak or doing it for the average or may be doing something in between as designing for the peak. If you design it for the peak everything works fine but that becomes very expensive and not practical in many cases. You cannot do it for the average also and that may be too lower so it may be somewhere in between. Therefore one inherent problem is the burstiness of the traffic. Now, if you could somehow make the burstiness smooth, then all your system will work much better. One way of doing is to buffer it somewhere. The shape of the traffic is related to some statistics about data transfer rates as well as its sensitivity to error, delay jitter etc.

(Refer Slide Time: 36:57-38:10)



One famous algorithm is the Leaky Bucket Algorithm. It is a single server queue with a constant service rate. If you have a bucket which is leaking drop by drop that means water will come out at a constant rate. Therefore the same thing happens here. If you have a single queue and then you service it at a constant rate this is the rate at which you are pumping the data into the network. So, if there is a burst then it will get absorbed in your buffer at the edge so that in the core of the network the burst will not come and it will be more of a steady kind of a flow. A steady average kind of flow is also something beyond the capacity of the intermediate nodes. Then of course the capacity of the intermediate nodes has to be increased. That is the leaky bucket algorithm in short. So the input buffer allows a bursty flow to be smoothed out to an even flow onto the network. It may be implemented in hardware or the OS Operating System. It may be implemented either in hardware or software.

(Refer Slide Time: 38:11)



The slide features a dark blue background with a yellow title 'Leaky Bucket Algorithm' and a list of three bullet points. A small logo is in the top left corner, and the number '47' is in the bottom right corner.

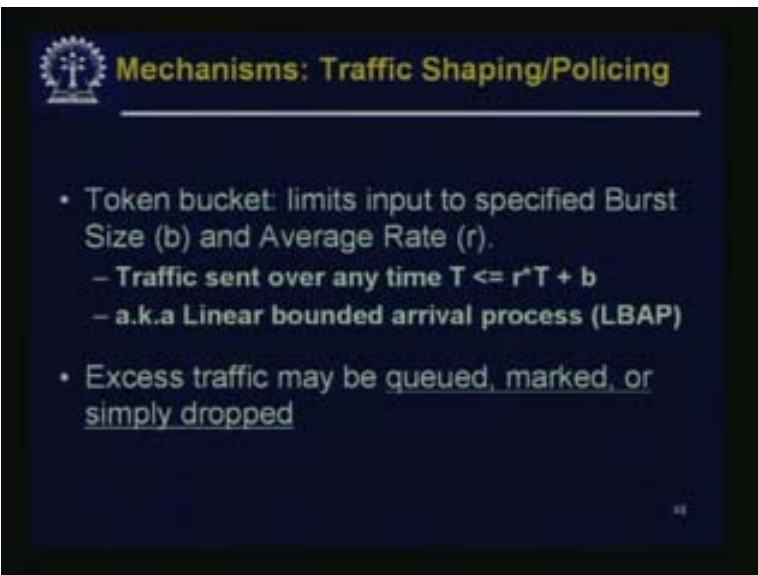
Leaky Bucket Algorithm

- Underutilized slots are written off.
- The algorithm can work on the volume of traffic rather than no. of packets.
- Somewhat slow response time for inherently bursty traffic.

47

Underutilized slots are written off. By this what we mean is, the packets are being serviced by this network at a particular rate, let us say once every T unit of time. Now after another T unit of time it will try to service and finds that the buffer is empty so it will not send anything. Again after T unit of time and if something has arrived by that time it will send one packet. The algorithm can work on the volume of the traffic rather than number of packets. Only problem here is, a somewhat slow response time for inherently bursty traffic which is quite often in the node.

(Refer Slide Time: 39:04-39:58)



The slide features a dark blue background with a yellow title 'Mechanisms: Traffic Shaping/Policing' and a list of two bullet points. A small logo is in the top left corner, and the number '48' is in the bottom right corner.

Mechanisms: Traffic Shaping/Policing

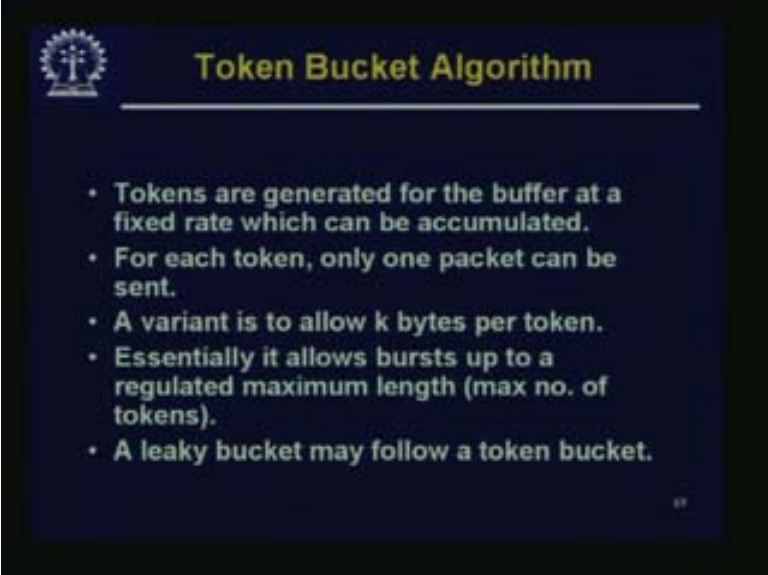
- Token bucket: limits input to specified Burst Size (b) and Average Rate (r).
 - Traffic sent over any time $T \leq r \cdot T + b$
 - a.k.a Linear bounded arrival process (LBAP)
- Excess traffic may be queued, marked, or simply dropped

48

One way to handle a little bit of burstiness is by a token bucket. This again improves the throughput a little bit and it can accommodate burstiness to a certain degree. We cannot allow all kinds of burstiness because then the burstiness will flow into the core of the network where it will be more difficult to handle. So this is the token bucket, it limits the input to specified burst size (b) and average rate (r).

So traffic sent over any time T is equal to $r \cdot T$ plus b, also known as linear bounded arrival process (LBAP). So there is bound on an arrival process. Excess traffic may be queued, marked or simply dropped.

(Refer Slide Time: 39:59-41:19)



The slide features a dark blue background with a light blue header area. In the top left corner of the header is a small circular logo containing a stylized 'T'. The title 'Token Bucket Algorithm' is centered in the header in a yellow font. Below the title, a list of five bullet points is displayed in white text. The slide is numbered '18' in the bottom right corner.

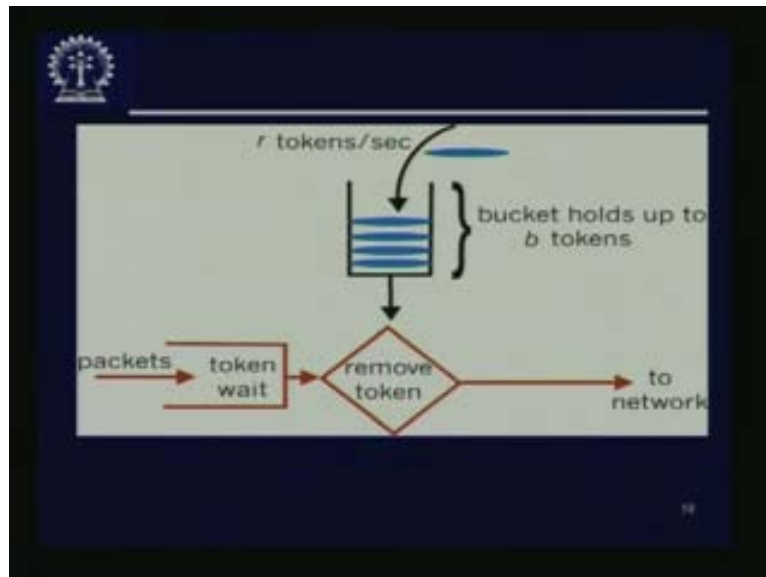
Token Bucket Algorithm

- Tokens are generated for the buffer at a fixed rate which can be accumulated.
- For each token, only one packet can be sent.
- A variant is to allow k bytes per token.
- Essentially it allows bursts up to a regulated maximum length (max no. of tokens).
- A leaky bucket may follow a token bucket.

18

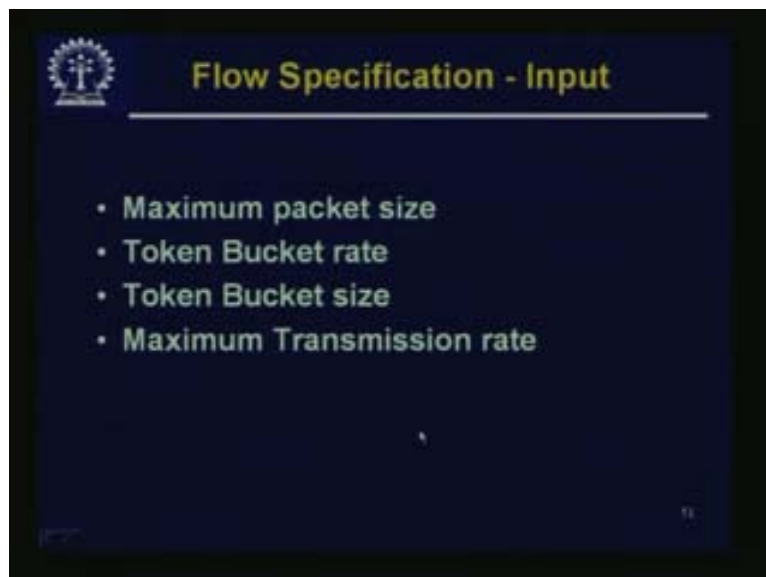
So, Tokens are generated for the buffer at a fixed rate which can be accumulated. So this is the main point where the token differs from the leaky bucket. In the leaky bucket the underutilized slots were written off. But here if your time comes you can get a token and you can collect and accumulate so many tokens. And then when a burst comes up to that many tokens can be sent. The longer time average is helpful because there is a limit to the number of tokens you can really accumulate because after that you cannot accumulate tokens anymore. And at the same time a little bit of burstiness is allowed if your source is inherently bursty, and if you can allow some amount of burstiness that will improve the throughput. So, for each token only one packet can be sent but tokens can be accumulated up to a certain maximum. A variant is to allow k bytes per token. Essentially it allows bursts up to a regulated maximum length that is maximum number of tokens. A leaky bucket may follow a token bucket also in order to make it absolutely smooth.

(Refer Slide Time: 41:20-41:44)



So this is the diagram, the bucket holds up to b tokens and there are so many tokens per second accumulating there. And when a packet burst comes then it waits for the tokens. If the tokens are not there then it cannot send. But if the tokens are there depending upon as many tokens that are available the tokens are removed and the packets are sent into the network.

(Refer Slide Time: 41:45-42:18)



Now having talked about this, let us just mention what are the kinds of traffic parameters that are important or that may be negotiated between the sender and the network service provider. One could be maximum packet size that defines how big the packet is. The

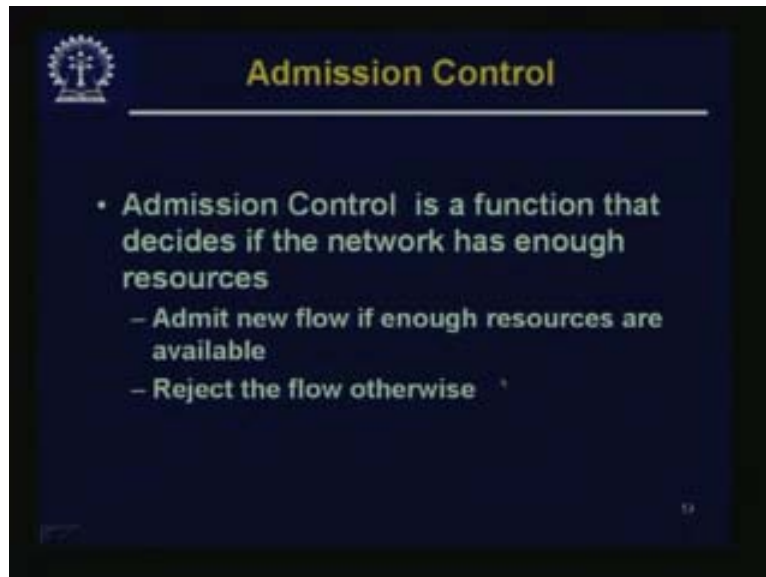
token bucket rate: Defines what the average rate is. Token bucket size defines how burst it will be. Maximum transmission rate tells us the exact maximum transmission rate.

(Refer Slide Time: 42:19-43:08)



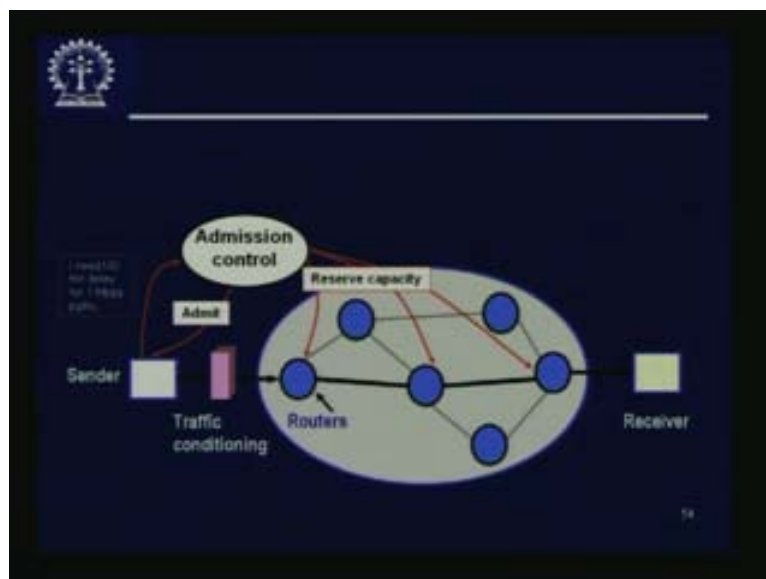
Loss sensitivity: Is this flow very sensitive to losses. If you are just doing some file transfer it will be sensitive to losses, but if you are sending some voice it may not be that sensitive. Loss Interval: At what interval it is a loss, if the loss is very bursty or if the loss has to be averaged out, etc. Burst loss sensitivity (packets): in terms of the number of packets. Minimum delay noticed, and maximum delay variation which are allowed. These are again very important for multimedia traffic. Quality of the guarantee: Is it just a best effort or better than the best effort is what it tells about. These are the flow specifications of services.

(Refer Slide Time: 43:09-43:39)



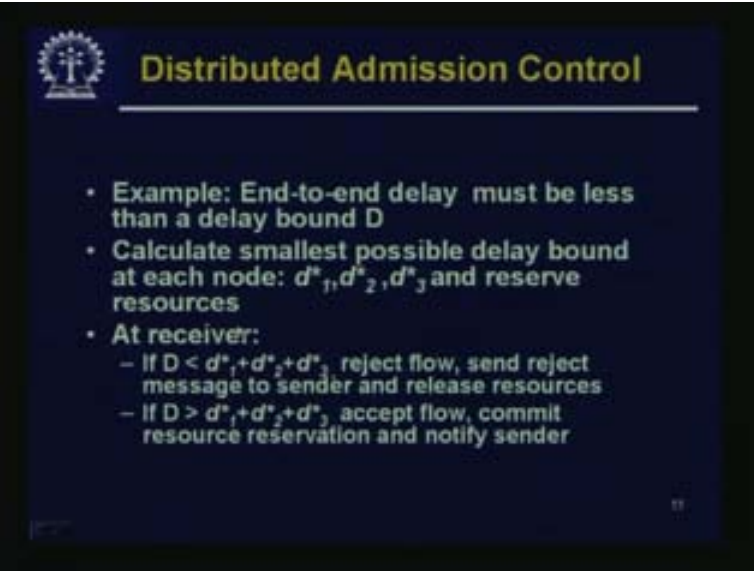
We will come to admission control and signaling in more detail when we discuss RSVP in the next lecture when we discuss QoS and Multimedia traffic. But just to mention it here, Admission Control is a function that decides if the network has enough resources. Admit new flow if enough resources are available. Reject the flow otherwise.

(Refer Slide Time: 43:40-44:06)



You do some reservation of capacity through some protocol like RSVP which we will discuss later. And if you find that you can reserve the capacity for this kind of flow that is the flow with these kinds of parameters then you admit it but otherwise you do not admit it. This assumes that we have some kind of a virtual circuit.

(Refer Slide Time: 44:07-44:20)

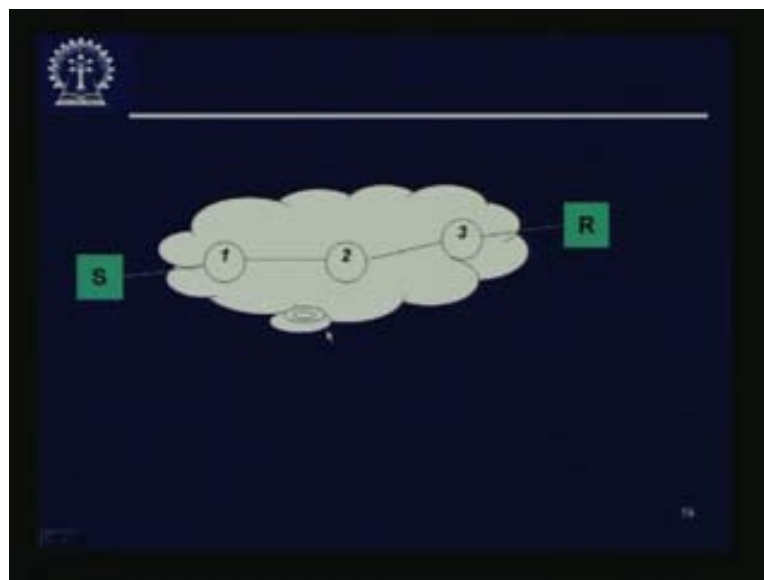


Distributed Admission Control

- Example: End-to-end delay must be less than a delay bound D
- Calculate smallest possible delay bound at each node: d_1^*, d_2^*, d_3^* and reserve resources
- At receiver:
 - If $D < d_1^* + d_2^* + d_3^*$, reject flow, send reject message to sender and release resources
 - If $D > d_1^* + d_2^* + d_3^*$, accept flow, commit resource reservation and notify sender

There may be Distributed Admission Control instead of central admission control at the beginning. For example, it may be end to end delay which must be less than a delay bound D . So calculate d_1, d_2 , etc and you reserve resources.

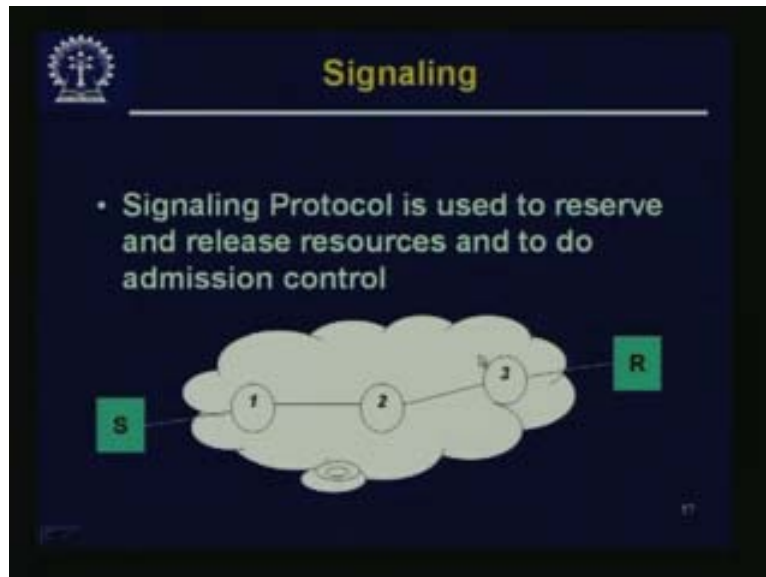
(Refer Slide Time: 44:21-44:48)



And what would you do is, the D is specified by the source and as it travels some reservation signal it calculates the delay d_1, d_2, d_3 etc and if $D < d_1$ plus d_2 plus d_3 then you reject the flow and if it is greater then you accept it. Send reject message to sender and release resources.

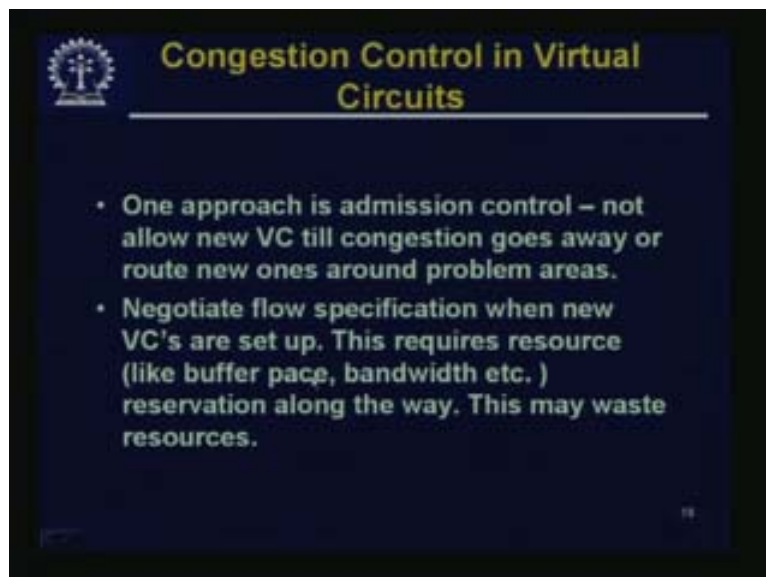
Therefore if $D > d_1 + d_2 + d_3$ accept flow, commit resource reservation and notify sender.

(Refer Slide Time: 44:49-45:12)



Some signaling protocol is used to reserve and release resources and to do admission control. So you reserve one mbps that the request goes through.

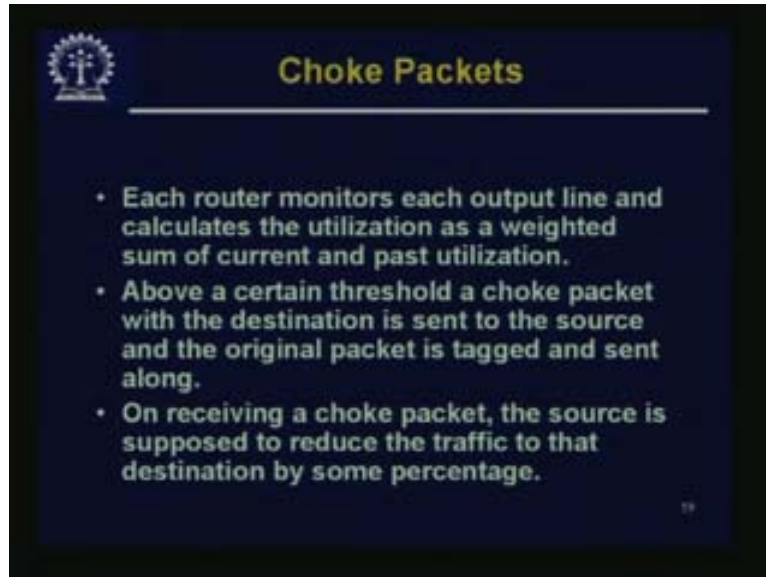
(Refer Slide Time: 45:13-45:39)



So, this is a Congestion Control in virtual circuits. One approach is admission control, not allow new VC till congestion goes away or route new ones around problem areas. Other

is, negotiate flow specification when new VCs are set up. This requires resource like buffer space, bandwidth etc, and reservation along the way. This may waste resources.

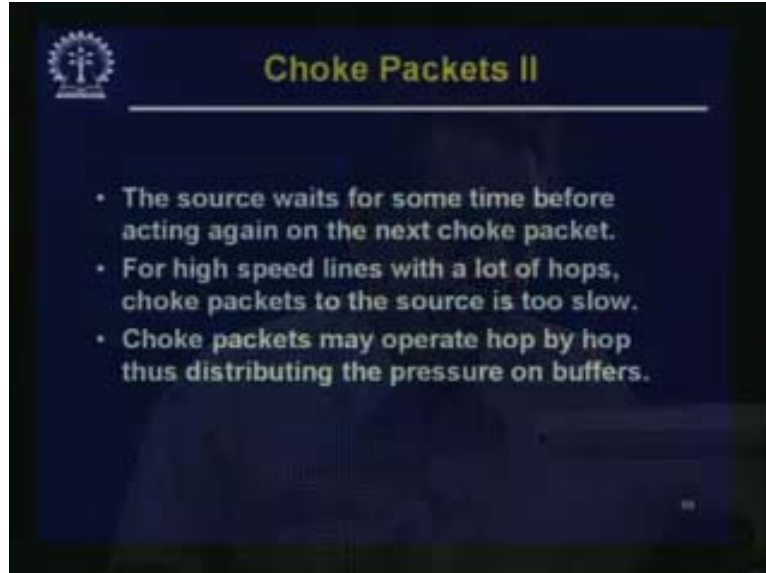
(Refer Slide Time: 45:40-47:01)



One topic we mentioned earlier is the TCP IP source quench or sometimes called as choke packets. This may be used as a crude mechanism for handling congestion. Each router monitors each output line and calculates the utilization as a weighted sum of current and past utilization. Above a certain threshold a choke packet with the destination is sent to the source and the original packet is tagged and sent along.

On receiving a source choke packet the source is supposed to reduce the traffic to that destination by some percentage. If that happens and if it works then that is very fine. When congestion is detected the source is sort of distributed and they are remote to each other. So, if you could send this feedback instantaneously then you could control the congestion much better but that is not possible. You have a distributed algorithm where you work only with some local information and something that might come along with some particular packet.

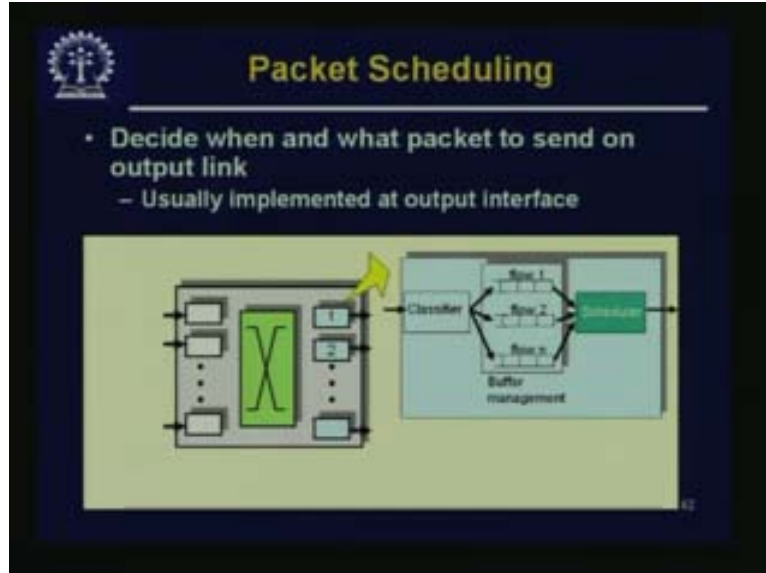
(Refer Slide Time: 47:02-47:59)



The source waits for some time before acting again on the next choke packet because there may be multiple choke packets coming. Therefore for the same burst it has created ripples of congestion along the way and all the routers sending choke packets so multiple choke packets does not necessarily mean these are independent but they may have come because of the same source so it waits for sometime before acting on the next choke packet. For high speed lines with a lot of hops, choke packets to the source is too slow. So, choke packets may operate hop by hop thus by distributing the pressure on buffers. These choke packets add to the network traffic and it operates hop by hop thus distributing the pressure on buffers.

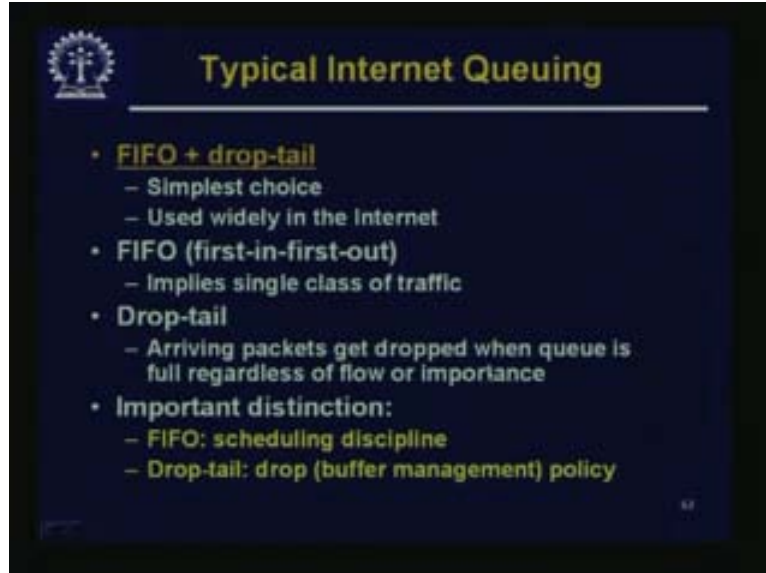
Scheduling: This is another way to handle congestion.

(Refer Slide Time: 48:17-49:42)



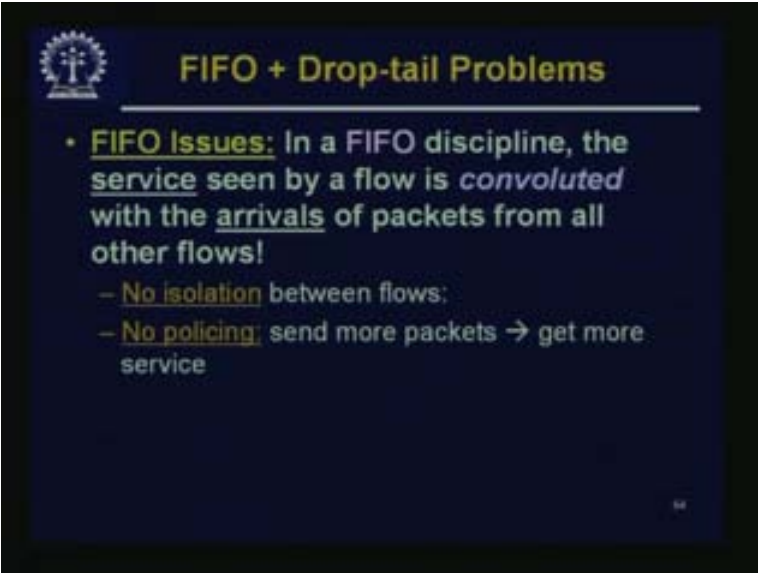
Packet scheduling has to be done by deciding when and what packet to send on output link, usually implemented at the output interface. Suppose you have some switch or some router or network node and a number of packets are coming out so what you might want to do is to classify these packets. In this context this could be the worst effort. There are premium services and other kind of services etc and the rest are the best effort for with the rest. What you may have is that you may have various classifications for these flows which may go into different queues and then there is a scheduler which schedules as to which queue to be serviced next. A scheduler has a vital role to play on the kind of services on each of the packets at the micro level and each of the flows in general at a higher level they get.

(Refer Slide Time: 49:43-51:17)



Typical Internet Queuing: In Internet queuing what we do is, we use FIFO plus drop tail. What is FIFO? FIFO is, First In First Out. It is a simplest choice and is used widely in internet. This First In First Out implies single class of traffic which is essentially means that we have a single queue. So whoever comes in first, he is the one who would be attended first for servicing. This FIFO has to do with scheduling. And there is a drop tail which means the arriving packets get dropped when queue is full regardless of which flow it belongs or regardless of its importance. So, if the buffer is full whoever comes next will be dropped. So FIFO has to do with the scheduling discipline and drop tail that is the drop policy has to do with the buffer management. This means how much of your buffer is kept empty or whether you allow the buffer to get full or whom you drop out when the buffer gets full etc are some of the buffer management policies. Now let us look at scheduling. Actually the scheduling policy and the buffer management policy always go hand with hand and the buffer management policy always come in pair.

(Refer Slide Time: 51:18-52:33)



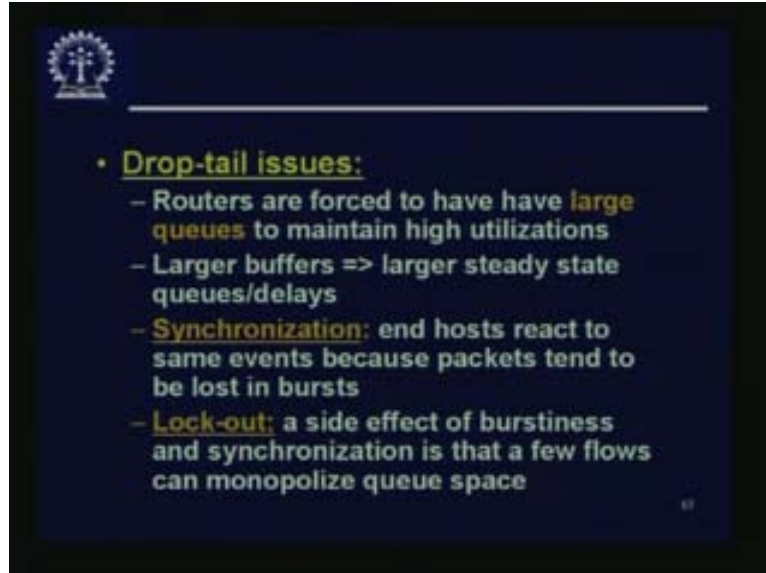
The slide features a dark blue background with a small gear icon in the top left corner. The title 'FIFO + Drop-tail Problems' is written in yellow at the top. Below the title, a bulleted list in yellow text describes FIFO issues. The first bullet point states that service is convoluted with arrivals from all flows. Two sub-points follow: 'No isolation between flows' and 'No policing', both explaining that higher packet rates lead to more service.

FIFO + Drop-tail Problems

- **FIFO Issues:** In a FIFO discipline, the service seen by a flow is *convoluted* with the arrivals of packets from all other flows!
 - No isolation between flows:
 - No policing: send more packets → get more service

FIFO Issues: In a FIFO discipline, the service seen by a flow is convoluted with the arrivals of packets from all other flows. So there is no isolation between flows and no policing. Send more packets and get more services. We have one single queue so whoever is pumping in more packets into it he is more likely to be serviced. Of course, you will lose some packets also but other people will also lose some packets but in some sense it favors somebody who is pumping data at a higher rate. So he gets more service or may be he requires it or it is just some kind of a row node. You do not differentiate between different flows at all. There is no isolation between the flows. If there is a flow which is very important but sends less number of packets he will get much lesser service compared to the one pumping lots of packets. So that is the Issue with FIFO.

(Refer Slide Time: 52:34-53:35)

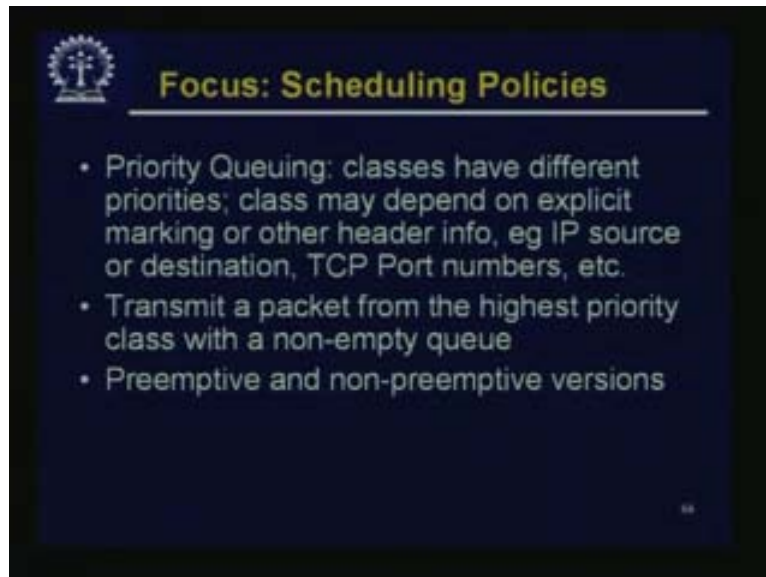


Drop-tail issues:

Routers are forced to have large queues to maintain high utilizations, that is a problem. Larger buffer implies larger steady state queues or delays so the delay is more. Synchronization: End hosts react to same events because packets tend to be lost in bursts. So what happens is, when the buffer gets full, different packets coming from different sources would be dropped. So, all the sources would know that the packets may be timed-out or something so they would act again in **unition** and this acting in **unition** is always bad, then you take another step which again is wrong in some way, this makes it become more bursty

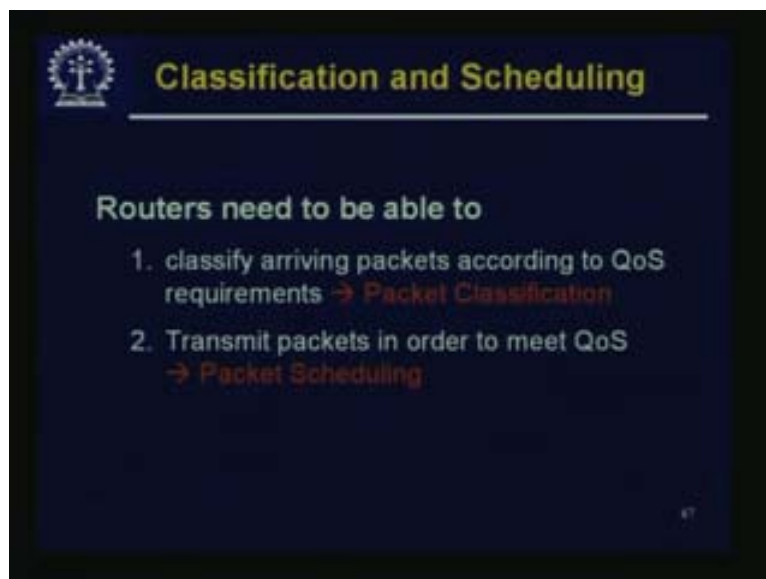
Lock out: A side effect of burstiness and synchronization is that a few flows can monopolize the queue space. So these are the drop-tail issues.

(Refer Slide Time: 53:36-54:02)



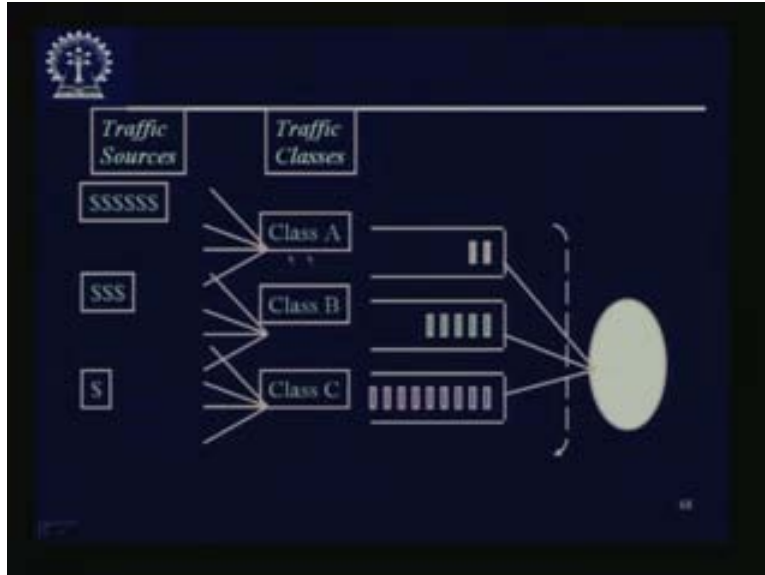
Priority Queuing: classes have different priorities, and class may depend on explicit marking or other header info, for example IP source or destination, TCP port numbers, etc. Transmit a packet from the highest priority class with a non empty queue. This has preemptive and non preemptive versions. This is the kind of scheduling policies we have.

(Refer Slide Time: 54:03-54:11)



So, Routers must be able to classify arriving packets according to QoS requirements. This is known as packet classification and packets are transmitted in order to meet the QoS requirements which are known as packet scheduling.

(Refer Slide Time: 54:12-54:27)



You have Class A service which is very premium. Class B services and Class C services are also there. You might attach different priorities to different queues and serve them that way.

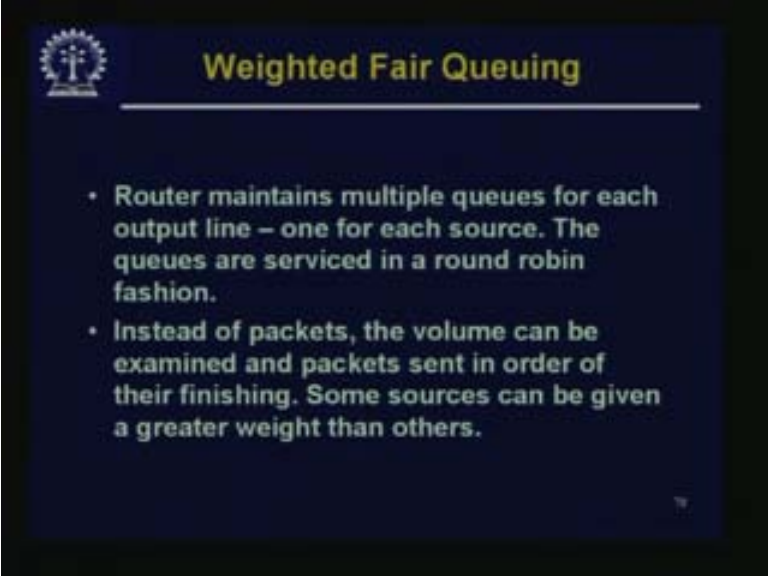
(Refer Slide Time: 54:28-54:41)


Queuing Disciplines

- Each router **must** implement some queuing discipline
- Queuing allocates **bandwidth and buffer space**:
 - Bandwidth: which packet to serve next (scheduling)
 - Buffer space: which packet to drop next (buff mgmt)
- Queuing also affects latency

So, each router must implement some queuing discipline. Queuing allocates bandwidth and buffer space, so bandwidth tells which packet to serve next (scheduling) and buffer space tells which packet to drop next (buff management). Queuing also affects latency.

(Refer Slide Time: 54:42-55:22)

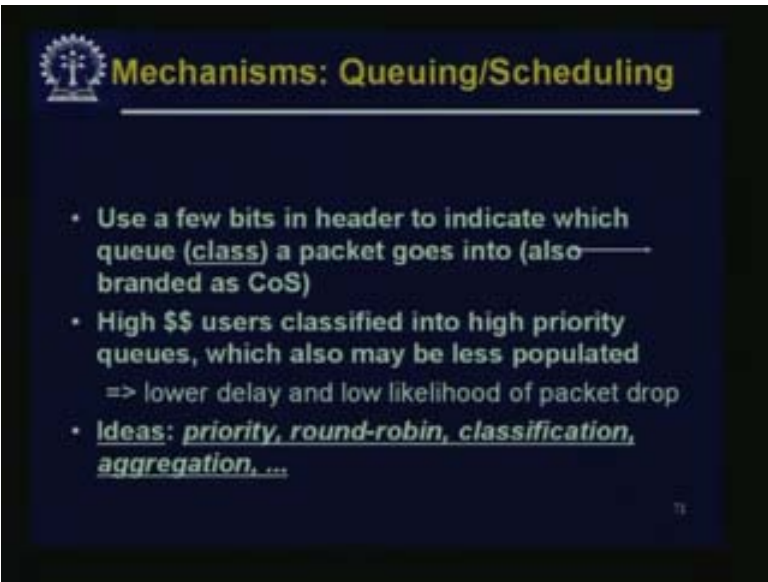



 **Weighted Fair Queuing**

- Router maintains multiple queues for each output line – one for each source. The queues are serviced in a round robin fashion.
- Instead of packets, the volume can be examined and packets sent in order of their finishing. Some sources can be given a greater weight than others.

One thing which is very widely used is Weighted Fair Queuing. Router maintains multiple queues for each output line one for each source. The queues are serviced in a round robin fashion. Instead of packets the volume can also be examined and packets sent in order of their finishing. Some sources can be given a greater weight than others. The point is, even if you do not give a greater weight, then the service which is premium, may be much less number of people might be there. So, since the round robin is between the queues, automatically those which have the premium class get a better service since the population is low.

(Refer Slide Time: 55:23-57:33)



 **Mechanisms: Queuing/Scheduling**

- Use a few bits in header to indicate which queue (class) a packet goes into (also branded as CoS)
- High \$\$ users classified into high priority queues, which also may be less populated
=> lower delay and low likelihood of packet drop
- Ideas: priority, round-robin, classification, aggregation, ...

Use a few bits in header to indicate which queue (class) a packet goes into (also branded as CoS. Lower delay and low likelihood of packet drop for high end users. Priority, round robin, classification, aggregation etc are the different mechanism which we use. With this we come to a sort of the end of short handling of this congestion control issue, it is not a very easy issue because as we know this is a global problem but you have to take some local action so that it works fine.

In the next lecture we will take up Quality of Service, Quality of service as we have already mentioned today, we have different kinds of quality requirements for different sets of people. As I mentioned that if you are transferring a file, you do not want any bit to be lost because it may be a very vital bit so it may be a binary or a source or something that the whole thing may become junk. If you lose one bit, it is very difficult and if it goes in a jerky fashion or takes longer time then you may not mind. So that is one kind of quality you require.

Another kind of quality you might require is, when you are doing some kind of multimedia transmission like audio, video, etc where I may sort of be insensitive to a few packets or a few bits being lost here and there but if the delay is too large or keeps on varying too much then I have a problem with the quality of reception. So that is a different kind of quality. So how to handle different kinds of quality and how multimedia transmission etc can take place in a network etc would be the content of our next lecture.

Thank you.