BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 4

DCA2201

COMPUTER NETWORKING

Unit 9

Transport Layer – Congestion Control

Table of Contents

1. INTRODUCTION

In the previous unit, we discussed transport layer protocols such as TCP and UDP. Also, we have studied how data can be transferred among processes across heterogeneous networks. In this unit, we will discuss different congestion control measures taken by transport layer while transferring data from source process to a process on a destination machine. Congestion occurs when a link is overloaded with data and which results in the data packet loss and blocking of new connection.

In this unit, first we will discuss various congestion control algorithms. In the next section, we will discuss different congestion avoidance mechanisms such as DECbit, random early detection and source-based congestion avoidance.

1.1 Objectives:

After studying this unit, you should be able to:

- *Describe congestion control algorithms*
- *Explain congestion avoidance using DECbit*
- *Describe random early detection*
- *Explain source-based congestion avoidance*

2. CONGESTION CONTROL ALGORITHMS

The presence of many packets in the network causes packet delay and loss that degrades performance. This situation is called congestion. The most effective way to control congestion is to reduce the load that the transport layer is placing on the network. Two solutions for congestion control are, increase the resources or decrease the load. Solutions are usually applied on different time scales as shown in figure 9.1.

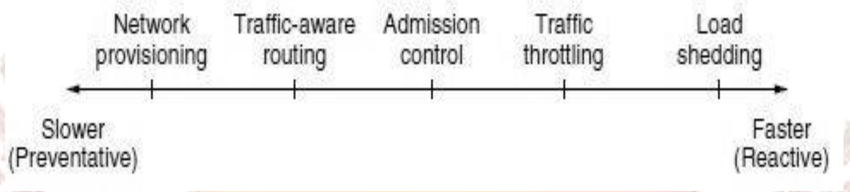


Fig 9.1: Timescales of approaches to congestion control.

The most basic way to avoid congestion is to build a network that is well matched to the traffic that it carries. Different approaches to congestion control are described below.

2.1 Network Provisioning

Congestion probably occurs if there is a low-bandwidth link on the path along which most traffic is directed. In case of a serious congestion, we can add resources such as spare routers or back up lines, dynamically. More often, links and routers that are regularly heavily utilized are upgraded at the earliest opportunity. This is called *provisioning* and happens on a time scale of months, driven by long-term traffic trends.

2.2 Traffic-aware Routing

Traffic-aware routing adapted to changes in topology, but not to changes in load. When computing new routes, we will consider load into account and try to shift the load away from hotspots. Hotspots are the first place in the network to experience congestion.

Consider the network shown in figure 9.2. Here, the network is divided into two parts, East and West, connected by two links, *CF* and *EI*. Suppose that most of the traffic between East and West is using link *CF*, and, as a result, this link is heavily loaded with long delays. Once CF is overloaded, most of the East-West traffic will now go over *EI*, loading this link. Consequently, in the next update, *CF* will appear to be the shortest path. As a result, the routing tables may oscillate wildly, leading to erratic routing and many potential problems.

If load is ignored and only bandwidth and propagation delay is considered, this problem does not occur. Two techniques can contribute to a successful solution. The first is multipath routing, in which there can be multiple paths from a source to a destination

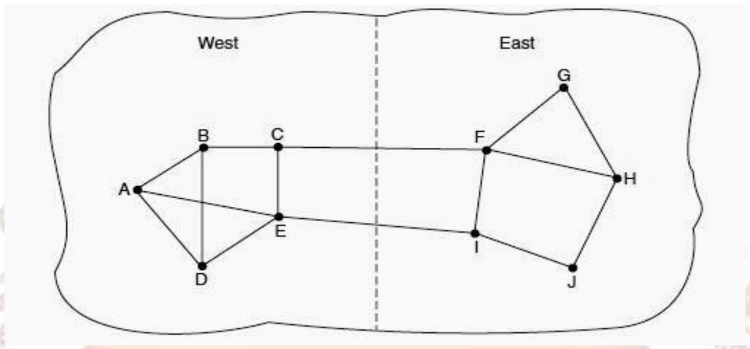


Fig 9.2: A network in which East & West parts are connected by two links

The second one is for the routing scheme to shift traffic across routes. The method of optimizing the performance of a telecommunications network by dynamically analyzing, predicting and regulating the behavior of data transmitted over that network is called *traffic engineering.*

2.3 Admission Control

In a virtual-circuit network, new connections can be refused if they would cause the network to become congested. That is, a check is performed before a connection is established to see if available resources are sufficient for the proposed connection. If there are no sufficient resources, new connections can be rejected. This is called *admission control.* The idea behind admission control is that, do not set up a new virtual circuit unless the network can carry the added traffic without becoming congested. Traffic is often described in terms of its rate and shape. One example of an admission control approach is *leaky bucket algorithm* also known as *token bucket.* The concept behind leaky bucket is that, irrespective of the incoming rate, outgoing flow rate will be constant.

2.4 Traffic Throttling

In the Internet and many other computer networks, senders adjust their transmissions to send as much traffic as the network can readily deliver. In this setting, the network aims to operate just before the attack of congestion. When congestion is nearby, it must tell the senders to throttle back their transmissions and slow down. To deliver feedback, the router must identify the appropriate senders. It must then warn them carefully, without sending many more packets into the already congested network. Different schemes use different feedback mechanisms.

Choke Packets

A choke packet is used in network maintenance and quality management to inform a specific node or transmitter that its transmitted traffic is creating congestion over the network. The most direct way to notify a sender of congestion is to tell it directly. In this approach, the router selects a congested packet and sends a *choke packet* back to the source host, giving it the destination found in the packet. The original packet may be tagged so that it will not generate any more choke packets farther along the path and then forwarded in the usual way. To avoid increasing load on the network during a time of congestion, the router may only send choke packets at a low rate.

Explicit Congestion Notification

Instead of generating additional packets to warn of congestion, a router can tag any packet it forwards (by setting a bit in the packet's header) to signal that it is experiencing congestion. When the network delivers the packet, the destination can note that there is congestion and inform the sender when it sends a reply packet. The sender can then throttle its transmissions as before. This design is called *ECN (Explicit Congestion Notification)* and is used in the Internet.

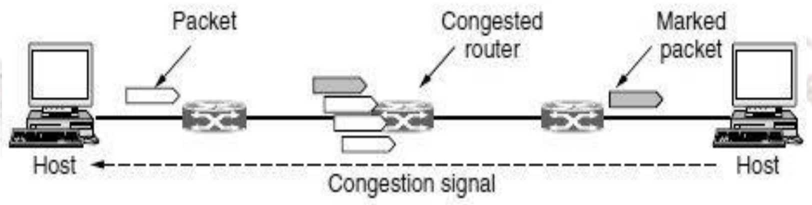


Fig 9.3: Explicit congestion notification

As illustrated in figure 9.3, if any of the routers packet pass through is congested, that router will then mark the packet as having experienced congestion as it is forwarded. The destination will then echo any marks back to the sender as an explicit congestion signal in its next reply packet. This is shown with a dashed line in the figure to indicate that it happens above the IP level. The sender must then throttle its transmissions, as in the case of choke packets.

Hop-by-Hop Backpressure

Sending backpressure to the sender when congestion occurs in the network is known as hop-by-hop backpressure. At high speeds or over long distances, many new packets may be transmitted after congestion has been signaled because of the delay before the signal takes effect. An ECN indication will take even longer because it is delivered via the destination. Choke packet propagation is illustrated as the second, third, and fourth steps in figure 9.4 (a).

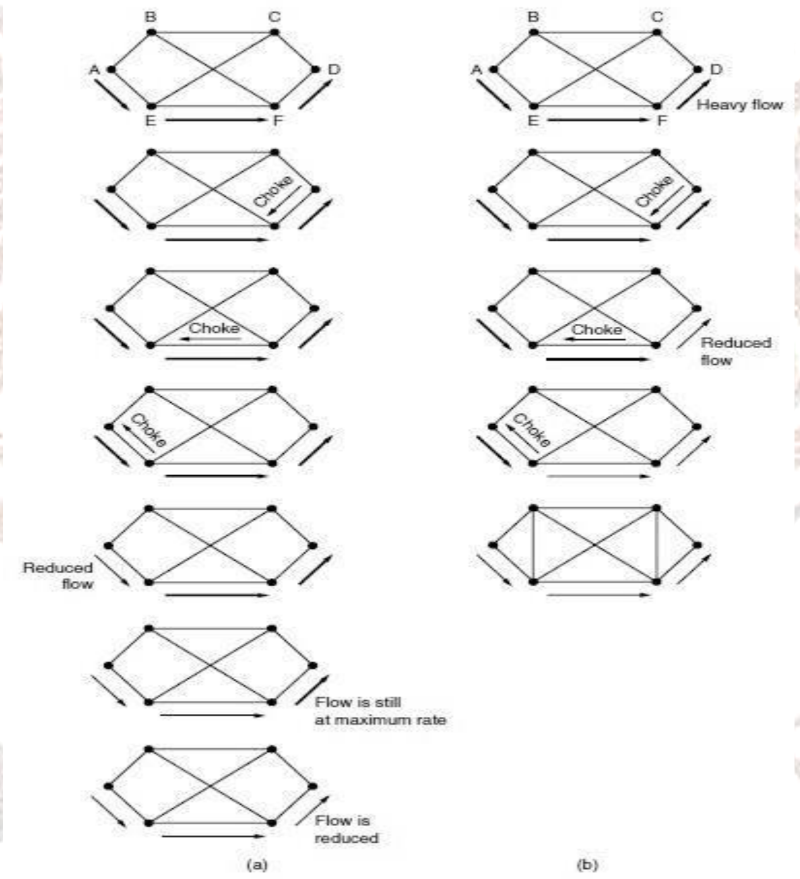


Fig 9.4 (a) A choke packet that affects only the source. (b) A choke packet that affects each hop it passes through.

An alternative approach is to have the choke packet take effect at every hop it passes through, as shown in the sequence of Figure 9.4(b). Here, as soon as the choke packet reaches *F*, *F* is required to reduce the flow to *D*. Doing so will require *F* to devote more buffers to the connection, since the source is still sending away at full blast, but it gives *D* immediate relief. In the next step, the choke packet reaches *E*, which tells *E* to reduce the flow to *F*. This action puts a greater demand on *E*'s buffers but gives F immediate relief. Finally, the choke packet reaches *A* and the flow genuinely slows down. The net effect of this hop-by-hop scheme is to provide quick relief at the point of congestion.

2.5 Load Shedding

When none of the above methods make the congestion disappear, routers can use a heavy weapon known as load shedding. The idea behind Load shedding is that when routers are being flooded by packets that they cannot handle, they just throw them away. The key question for a router drowning in packets is which packets to drop. The preferred choice may depend on the type of applications that use the network. For a file transfer, an old packet is worth more than a new one. In contrast, for real-time media, a new packet is worth more than an old one. This is because packets become useless if they are delayed and miss the time at which they must be played out to the user.

The former policy (old is better than new) is often called ***wine*** and the latter (new is better than old) is often called ***milk*** because most people would rather drink new milk and old wine than the alternative. To implement an intelligent discard policy, applications must mark their packets to indicate to the network how important they are. Then, when packets have to be discarded, routers can first drop packets from the least important class, then the next most important class, and so on.

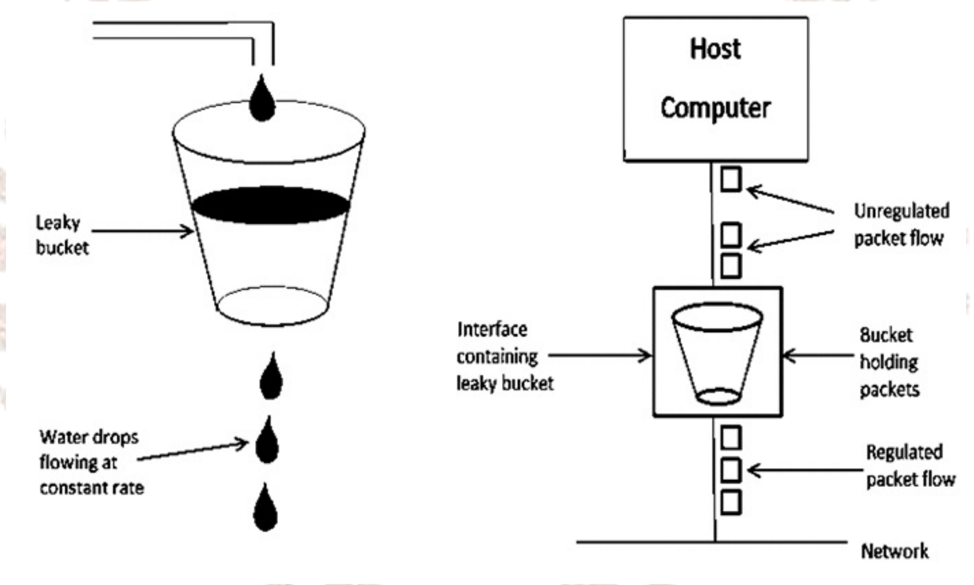
Open Loop Congestion Control: In this control policies are applied to prevent congestion before it happens. It is handled either by the source or the destination.

- Retransmission Policy – This type of policy is sometimes unavoidable. If the sender feels that a packet is lost or corrupted, then it thinks to retransmit. So, a retransmission policy can prevent congestion.
- Window Policy – In this type of window the sender may also affect congestion. The selective repeat window is better than the Go-Back-N window for congestion control. In the Go-Back-N window, when the timer for a packet time out, a number of packets may be resent, although some may have arrived safe and sound at the receiver.
- Acknowledgement Policy – This policy imposed by the receiver may also affect congestion. If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion. A receiver may send an

acknowledgment only if it has a packet to be sent or a special timer expires. A receiver may decide to acknowledge only N packets at a time.

- Discarding Policy – A discarding policy by the routers prevents congestion and at the same time may not harm the integrity of the transmission.
- Admission Policy – An admission policy is a quality-of-service mechanism, which prevents congestion in virtual circuit networks.

Leaky Bucket Algorithm:



Leaky Bucket Algorithm mainly controls the total amount and the rate of the traffic sent to the network.

Step 1 – Let us imagine a bucket with a small hole at the bottom where the rate at which water is poured into the bucket is not constant and can vary but it leaks from the bucket at a constant rate.

Step 2 – So (up to water is present in the bucket), the rate at which the water leaks does not depend on the rate at which the water is input to the bucket.

Step 3 – If the bucket is full, additional water that enters into the bucket spills over the sides and is lost.

Step 4 – Thus the same concept applied to packets in the network. Consider that data is coming from the source at variable speeds. Suppose that a source sends data at 10 Mbps for

4 seconds. Then there is no data for 3 seconds. The source again transmits data at a rate of 8 Mbps for 2 seconds. Thus, in a time span of 8 seconds, 68 Mb data has been transmitted.

That's why if a leaky bucket algorithm is used, the data flow would be 8 Mbps for 9 seconds. Thus, the constant flow is maintained.

Token Bucket Algorithm

The leaky bucket algorithm enforces output patterns at the average rate, no matter how busy the traffic is. So, to deal with the more traffic, we need a flexible algorithm so that the data is not lost. One such approach is the token bucket algorithm.

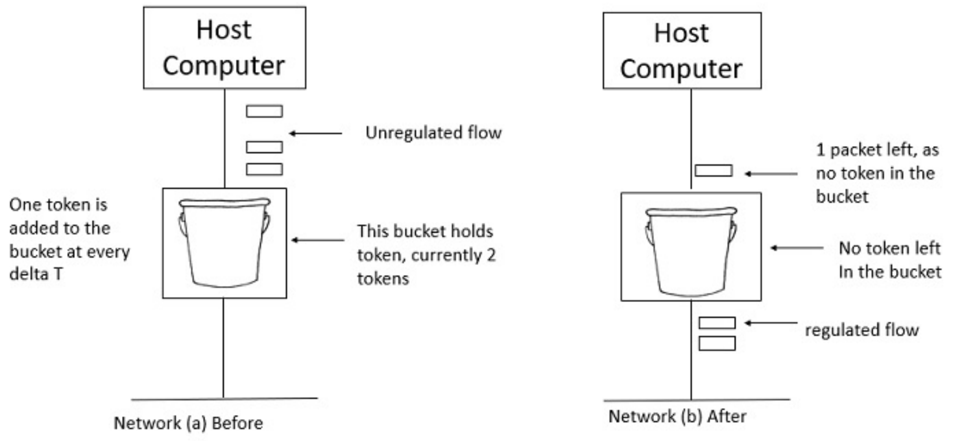
Let us understand this algorithm step wise as given below –

Step 1 – In regular intervals tokens are thrown into the bucket f.

Step 2 – The bucket has a maximum capacity f.

Step 3 – If the packet is ready, then a token is removed from the bucket, and the packet is sent.

Step 4 – Suppose, if there is no token in the bucket, the packet cannot be sent.


Host Computer
Unregulated flow
One token is added to the bucket at every delta T
This bucket holds token, currently 2 tokens
Network (a) Before
Host Computer
1 packet left, as no token in the bucket
No token left In the bucket
regulated flow
Network (b) After

In figure (a) the bucket holds two tokens, and three packets are waiting to be sent out of the interface.

In Figure (b) two packets have been sent out by consuming two tokens, and 1 packet is still left.

When compared to Leaky bucket the token bucket algorithm is less restrictive that means it allows more traffic. The limit of busyness is restricted by the number of tokens available in the bucket at a particular instant of time.

The implementation of the token bucket algorithm is easy – a variable is used to count the tokens. For every t-second the counter is incremented and then it is decremented whenever a packet is sent. When the counter reaches zero, no further packet is sent out.

Factors Affecting Congestion:

- Transport Layer
- Retransmission Policy
- Out-of-Order Receiving
- Acknowledgement factor
- Flow control determination
- Timeout determination

Network Layer

- Virtual Circuit vs. Datagram usage
- Packet discarding policy
- Packet queuing and servicing policy
- Routing algorithm
- Packet lifetime management

Data-link Layer

- Retransmission policy
- Out-of-Order Receiving
- Acknowledgement and flow control policy

SELF-ASSESSMENT QUESTIONS – 1

1. The presence of many packets in the network causes packet delay and loss that degrades performance, this situation is called ________________________.
2. Two solutions for congestion control are, __________________and _____________
3. "Traffic-aware routing adapted to changes in topology, but not to changes in load". (True/False)
4. Which among the following is NOT a congestion control algorithm?
   a) Traffic aware routing
   b) admission control
   c) DEC bit
   d) Load shedding
5. One example of an admission control approach is ________________.
6. Which method is more suitable for congestion control in a high speed or over long-distance transmission?
   a) Choke packet
   b) hop-by-hop backpressure
   c) ECN
   d) Admission control
7. The policy in which the old packet is better than the new one is often called _________________and the new packet is better than the old one is called _________________

.

3. CONGESTION AVOIDANCE MECHANISMS

In the case of TCP, the strategy is congestion control rather than congestion avoidance. TCP repeatedly increases the load it imposes on the network in an effort to find the point at which congestion occurs, and thus finds the available bandwidth of the connection. So, in effect, it creates losses to find the bandwidth. An alternative is to predict when congestion is about to happen and then to reduce the rate at which hosts send data just before packets start being discarded. Such a strategy is called congestion avoidance. This section describes three different congestion-avoidance mechanisms. They are: DECbit, Random early detection and source-based congestion avoidance. In the first two mechanisms, they put some additional functionality in to the router to assist the end node in the prediction of congestion. But in the third mechanism, it attempts to avoid congestion purely from the end nodes.

3.1 DECbit

DECbit mechanism was developed for use on the Digital Network Architecture (DNA), a connectionless network with a connection-oriented transport protocol. This mechanism could, therefore, also be applied to TCP and IP. The idea here is to more evenly split the responsibility for congestion control between the routers and the end nodes. Each router monitors the load it is experiencing and explicitly notifies the end nodes when congestion is about to occur. This notification is implemented by setting a binary congestion bit in the packets that flow through the router, hence the name *DECbit.* The destination host then copies this congestion bit into the ACK it sends back to the source. Finally, the source adjusts its sending rate so as to avoid congestion. Below discussion describes the procedure in more detail.

A single congestion bit is added to the packet header. A router sets this bit in a packet if its average queue length is greater than or equal to 1 at the time the packet arrives. This average queue length is measured over a time interval that spans the last busy+idle cycle, plus the current busy cycle. Figure 9.5 shows the queue length at a router as a function of time. Essentially, the router calculates the area under the curve and divides this value by the time interval to compute the average queue length. Using a queue length of 1 as the trigger for setting the congestion bit is a trade-off between significant queuing and increased idle time.

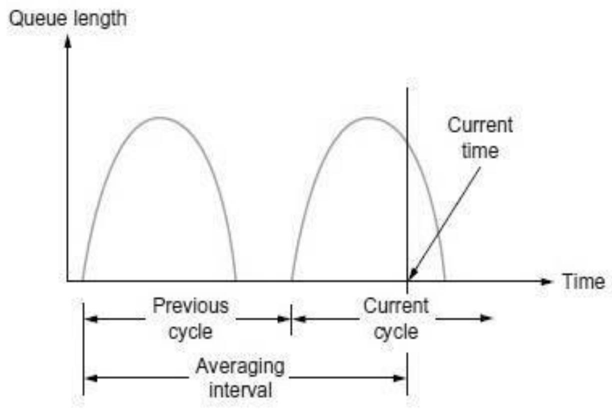


Fig 9.5: Computing average queue length at a router

Let's see in the source records how many of its packets resulted in some router setting the congestion bit. In particular, the source maintains a congestion window, just as in TCP, and watches to see what fraction of the last window's worth of packets resulted in the bit being set. If less than 50% of the packets had the bit set, then the source increases its congestion window by one packet. If 50% or more of the last window's worth of packets had the congestion bit set, then the source decreases its congestion window to 0.875 times the previous value. The value 50% was chosen as the threshold based on analysis that showed it to correspond to the peak of the power curve. The "increase by 1, decrease by 0.875" rule was selected because additive increase/multiplicative decrease makes the mechanism stable.

3.2 Random Early Detection (RED)

RED, invented by Sally Floyd and Van Jacobson in the early 1990s, differs from the DECbit scheme in two major ways. The first is that rather than explicitly sending a congestion notification message to the source, RED is most commonly implemented such that it *implicitly* notifies the source of congestion by dropping one of its packets. The source is, therefore, effectively notified by the subsequent timeout or duplicate ACK. As the "early" part of the RED acronym suggests, the router drops a few packets before it has exhausted its buffer space completely, so as to cause the source to slow down, with the hope that this will mean it does not have to drop lots of packets later on.

The second difference between RED and DECbit is in the details of how RED decides when to drop a packet and what packet it decides to drop. To understand the basic idea, consider a simple FIFO queue. Rather than wait for the queue to become completely full and then be forced to drop each arriving packet, we could decide to drop each arriving packet with some *drop probability* whenever the queue length exceeds some *drop level.* This idea is called *early random drop*. The RED algorithm defines the details of how to monitor the queue length and when to drop a packet.

By having routers drop packets early, before the situation has become hopeless, there is time for the source to take action before it is too late. A popular algorithm for doing this is called *RED (Random Early Detection).* To determine when to start discarding, routers maintain a running average of their queue lengths. When the average queue length on some link exceeds a threshold, the link is said to be congested and a small fraction of the packets are dropped at random. Picking packets at random makes it more likely that the fastest senders will see a packet drop; this is the best option since the router cannot tell which source is causing the most trouble in a datagram network. The affected sender will notice the loss when there is no acknowledgement, and then the transport protocol will slow down. The lost packet is thus delivering the same message as a choke packet, but implicitly, without the router sending any explicit signal. RED routers improve performance compared to routers that drop packets only when their buffers are full, though they may require tuning to work well.

First, RED computes an average queue length using a weighted running average similar to the one used in the original TCP timeout computation. Queuing decision is taken based on the AvgLen (average queue length).

That is, AvgLen is computed as,
$\text{AvgLen} = (1 - \text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen}$
Where $0 < \text{Weight} < 1$ and SampleLen is the length of the queue when a sample measurement is made.

Second, RED has two queue length thresholds that trigger certain activity. They are MinThreshold and MaxThreshold. When a packet arrives at the gateway, RED compares the current AvgLen with these two thresholds, based on the following rules:

```
If AvgLen≤ MinThreshold
    Queue the packet
If MinThreshold < AvgLen < MaxThreshold
{
    Calculate probability P
    Drop the arriving packet with probability P
}
If MaxThreshold ≤ AvgLen
    Drop the arriving packet
```

If the average queue length is smaller than the lower threshold, no action is taken, and if the average queue length is larger than the upper threshold, then the packet is always dropped. If the average queue length is between the two thresholds, then the newly arriving packet is dropped with some probability P.

The random nature of RED confers an interesting property on the algorithm.

Because RED drops packets randomly, the probability that RED decides to drop a particular flow's packet(s) is roughly proportional to the share of the bandwidth that that flow is currently getting at that router. This is because a flow that is sending a relatively large number of packets is providing more candidates for random dropping. Thus, there is some sense of reasonable resource allocation built into RED.

Consider the setting of the two thresholds, *MinThreshold* and Max-Threshold. If the traffic is fairly bursty, then *MinThreshold* should be sufficiently large to allow the link utilization to be maintained at an acceptably high level. Also, the difference between the two thresholds should be larger than the typical increase in the calculated average queue length in one RTT (round trip time). Setting *MaxThreshold* to twice *MinThreshold* seems to be a reasonable rule of thumb given the traffic mix on today's Internet. In addition, since we expect the average queue length to hover between the two thresholds during periods of high load, there should be enough free buffer space above *MaxThreshold* to absorb the natural bursts that occur in Internet traffic without forcing the router to enter tail drop mode.

3.3 Source-Based Congestion Avoidance

Unlike the two previous congestion-avoidance schemes, which depended on new mechanisms in the routers, this new technique describes a strategy for detecting the nascent stages of congestion before loss of data occurs from the end hosts. We first give a brief overview of a collection of related mechanisms that use different information to detect the early stages of congestion, and then we describe a specific mechanism in detail.

The general idea of these techniques is to watch for some sign from the network that some router's queue is building up and that congestion will happen soon if no action takes place. For instance, the source might notice that as packet queues build up in the network's routers, there is a measurable increase in the RTT for each successive packet it sends.

One particular algorithm exploits this observation as follows: The congestion window normally increases as in TCP, but every two round-trip delays the algorithm checks to see if the current RTT is greater than the average of the minimum and maximum RTTs seen so far. If it is, then the algorithm decreases the congestion window by one-eighth.

A second algorithm does something similar. The decision as to whether or not to change the current window size is based on changes to both the RTT and the window size. The window is adjusted once every two roundtrip delays based on the product.

$$(CurrentWindow - OldWindow) \times (CurrentRTT - OldRTT)$$

Where CurrentWindow is the size of the window while calculating current round-trip time and OldWindow is the size of the window while calculating previous round trip time (OldRTT).

If the result is positive, the source decreases the window size by one eighth; if the result is negative or 0, the source increases the window by one maximum packet size. The window changes during every adjustment.

Another change seen as the network approaches congestion is the dropping of the sending rate. A third scheme takes advantage of this fact. Every RTT, it increases the window size by one packet and compares the throughput achieved to the throughput when the window was one packet smaller. If the difference is less than one-half the throughput achieved when only one packet was in transit as was the case at the beginning of the connection the algorithm decreases the window by one packet. This scheme calculates the throughput by dividing the number of bytes outstanding in the network by the RTT.

A fourth mechanism, the one we are going to describe in more detail, is similar to the last algorithm in that it looks at changes in the throughput rate or, more specifically, changes in the sending rate. However, it differs from the third algorithm in the way it calculates throughput, and instead of looking for a change in the slope of the throughput it compares the measured throughput rate with an expected throughput rate. The algorithm is known as TCP Vegas which uses the strategy of maintaining the "right" amount of extra data in the network. Obviously, if a source is sending too much extra data, it will cause long delays and possibly lead to congestion. Less obviously, if a connection is sending too little extra data, it cannot respond rapidly enough to transient increases in the available network bandwidth. TCP Vegas's congestion-avoidance actions are based on changes in the estimated amount of extra data in the network.

We now describe the algorithm in detail. First, define a given flow's BaseRTT (initial round trip time) to be the RTT of a packet when the flow is not congested. TCP Vegas sets BaseRTT to the minimum of all measured round-trip times; it is commonly the RTT of the first packet sent by the connection, before the router queues increase due to traffic generated by this flow. If we assume that we are not overflowing the connection, then the expected throughput is given by,

$$ExpectedRate = CongestionWindow/BaseRTT$$

Where CongestionWindow is the TCP congestion window, which we assume to be equal to the number of bytes in transit.

Second, TCP Vegas calculates the current sending rate, Actual Rate. This is done by recording the sending time for a distinguished packet, recording how many bytes are transmitted between the time that packet is sent and when its acknowledgment is received, computing the sample RTT for the distinguished packet when its acknowledgment arrives, and dividing the number of bytes transmitted by the sample RTT. This calculation is done once per round-trip time.

Third, TCP Vegas compares ActualRate of throughput to ExpectedRate and adjusts the window accordingly. Let, Diff = *ExpectedRate−ActualRate.* Diff is positive or 0 by definition. Since ActualRate > ExpectedRate implies that we need to change BaseRTT to the latest sampled RTT. We also define two thresholds, say $\alpha < \beta$, roughly corresponding to having too little and too much extra data in the network, respectively. When *Diff*$<\alpha$, TCP Vegas increases the congestion window linearly during the next RTT, and when *Diff*$>\beta$, TCP Vegas decreases the congestion window linearly during the next RTT. TCP Vegas leaves the congestion window unchanged when $\alpha <$ Diff $< \beta$.

Finally, you will notice that TCP Vegas decreases the congestion window linearly, seemingly in conflict with the rule that multiplicative decrease is needed to ensure stability. The explanation is that TCP Vegas does use multiplicative decrease when a timeout occurs; the linear decrease just described is an early decrease in the congestion window that should happen before congestion occurs and packets start being dropped.

SELF-ASSESSMENT QUESTIONS – 2

8. The strategy in which prediction is done when congestion is about to happen and then to reduce the rate at which hosts send data just before packets start being discarded, is called ________________.
9. Mechanism was developed for use on the Digital Network Architecture.
10. "Digital Network Architecture (DNA) is a connection-oriented network with a connection-oriented transport protocol". (True/False)
11. RED, invented by ________________and ____________in the early 1990s.
12. Describe a strategy for detecting the nascent stages of congestion before losses occur from the end hosts.

4. SUMMARY

Let us recapitulate the important concepts discussed in this unit:

- The presence of many packets in the network causes packet delay and loss that degrades performance. This situation is called congestion.
- Two solutions for congestion control are, increase the resources or decrease the load.
- Sometimes resources can be added dynamically when there is serious congestion. This is known as network provisioning.
- Traffic-aware routing adapted to changes in topology, but not to changes in load.
- In a virtual-circuit network, new connections can be refused if they would cause the network to become congested. This is called admission control.
- The idea behind Load shedding is that when routers are being flooded by packets that they cannot handle, they just throw them away.
- Three different congestion-avoidance mechanisms are DECbit, Random early detection and source-based congestion avoidance.
- In DECbit and random early detection, some additional functionality has been implemented in to the router to assist the end node in the prediction of congestion
- In source-based congestion avoidance, an attempt is made to avoid congestion purely from the end nodes.

5. TERMINAL QUESTIONS

1. Describe Traffic-aware routing.
2. Explain Traffic throttling.
3. Compare choke packets and Hop-by-Hop Backpressure techniques.
4. Explain Load shedding.
5. Write short notes on DECbit and Random early detection.
6. Describe source-based congestion avoidance.

6. ANSWERS

Self-Assessment Questions

1. Congestion
2. Increase the resources, decrease the load
3. True
4. (c) DECbit
5. Leaky bucket algorithm
6. (b) hop-by-hop backpressure
7. Wine, Milk
8. Congestion avoidance
9. DECbit
10. (b) false
11. Sally Floyd, Van Jacobson
12. Source-based congestion avoidance

Terminal Questions

1. Traffic-aware routing adapted to changes in topology, but not to changes in load. The goal in taking load into account when computing routes is to shift traffic away from hotspots that will be the first places in the network to experience congestion. (Refer section 2.2 for more details).
2. When congestion is nearby, it must tell the senders to throttle back their transmissions and slow down. To deliver feedback, the router must identify the appropriate senders. It must then warn them carefully, without sending many more packets into the already congested network. (Refer section 2.4 for more details).
3. The most direct way to notify a sender of congestion is to tell it directly. In this approach, the router selects a congested packet and sends a *choke packet* back to the source host, giving it the destination found in the packet. An alternative approach is to have the choke packet take effect at every hop it passes through. (Refer section 2.4 for more details).
4. The idea behind Load shedding is that when routers are being flooded by packets that they cannot handle, they just throw them away. (Refer section 2.5 for more details).

5. The DECbit mechanism was developed for use on the Digital Network Architecture (DNA), a connectionless network with a connection-oriented transport protocol. This mechanism could, therefore, also be applied to TCP and IP. RED, invented by Sally Floyd and Van Jacobson in the early 1990s, differs from the DECbit scheme in two major ways. The first is that rather than explicitly sending a congestion notification message to the source, RED is most commonly implemented such that it implicitly notifies the source of congestion by dropping one of its packets. (Refer section 3.1 and 3.2 for more details).
6. Source-Based Congestion Avoidance technique describes a strategy for detecting the nascent stages of congestion before losses occur from the end hosts. The general idea of these techniques is to watch for some sign from the network that some router's queue is building up and that congestion will happen soon if no action takes place. (Refer section 3.3 for more details).

References:

- Andrew S Tanenbaum, David J. Wetherall, "*Computer Networks,*" Fifth edition.
- Larry L. Peterson, Bruce S. Davie, "Computer *Networks – a Systems Approach*," Fifth edition.
- James F. Kurose, Keith W. Ross, "*Computer Networking – A top-down approach,*" Sixth edition.
- Behrouz A. Forouzan, Sophia Chung Fegan, *"Data Communication and Networking,"* Fourth edition.
- William Stallings, *"Computer Networking with Internet Protocols and Technology,"* Third edition.