بسمه تعالی

دانشگاه علم و صنعت ایران

School of Computer Engineering

Advanced Computer Networks

Transport Layer and Congestion Control

Part 8

Seyed Hamed Rastegar

Fall 1401

Lecture: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Connection-oriented transport: TCP
- Principles of congestion control
- Congestion Control Techniques
  - Basic Related Schemes
  - TCP Congestion Control
  - Advanced Schemes

Advanced Schemes

- In this part we explore congestion control more deeply.
- In doing so, it is important to understand that the standard TCP’s strategy is to control congestion once it happens, as opposed to trying to avoid congestion in the first place.
- In fact, TCP repeatedly increases the load it imposes on the network in an effort to find the point at which congestion occurs, and then it backs off from this point.

Advanced Schemes

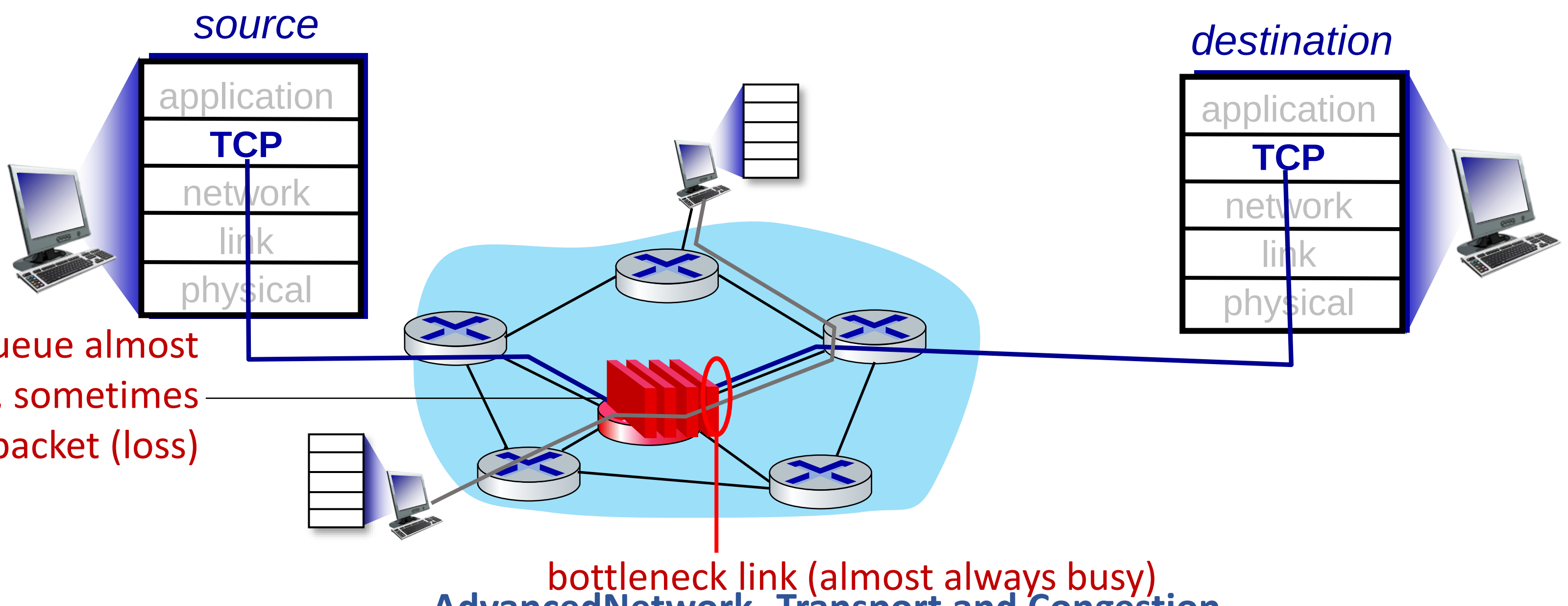
- Said another way, TCP needs to create losses to find the available bandwidth of the connection.

- An appealing 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.

- We call such a strategy congestion avoidance to distinguish it from congestion control, but it is probably most accurate to think of “avoidance” as a subset of “control.”

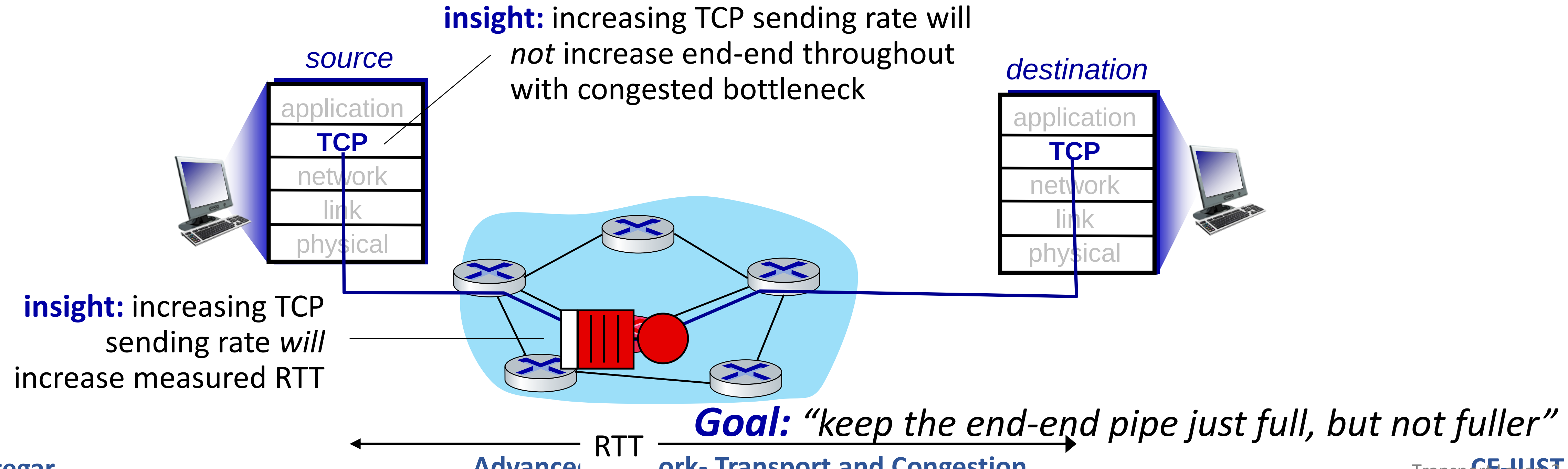
TCP and the congested "bottleneck link"

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*


source
application
TCP
network
link
physical
destination
application
TCP
network
link
physical
packet queue almost never empty, sometimes overflows packet (loss)
bottleneck link (almost always busy)

TCP and the congested "bottleneck link"

- TCP (classic, CUBIC) increase TCP's sending rate until packet loss occurs at some router's output: the *bottleneck link*
- understanding congestion: useful to focus on congested bottleneck link



insight: increasing TCP sending rate will *not* increase end-end throughout with congested bottleneck
source
application
TCP
network
link
physical
destination
application
TCP
network
link
physical
insight: increasing TCP sending rate *will* increase measured RTT
RTT

Goal: *"keep the end-end pipe just full, but not fuller"*

Advanced Schemes

- We describe two different approaches to congestion avoidance.
- The **first** puts a small amount of additional functionality into the router to assist the end node in the anticipation of congestion.
  - This approach is often referred to as **Active Queue Management (AQM)** which lies in the category of **network-assisted schemes**.
- The **second** approach attempts to avoid congestion purely from the end hosts.
  - This approach is implemented in TCP, making it a variant of the congestion control mechanisms described in the previous section.
  - This approach is often referred to as **source-based/delay-based scheme**.

Advanced Schemes

AQM

- The first approach requires changes to routers
  - It has never been the Internet’s preferred way of introducing new features
  - But nonetheless it has been a constant source of consternation over the last 20 years.
- The problem is that while it is generally agreed that routers are in an ideal position to detect the onset of congestion,
  - ➢ **There has not been a consensus on exactly what the best algorithm is.**
- In the following, we describe two of the classic mechanisms.

Advanced Schemes

AQM: DECbit >> Overview

- The first 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.
- As noted above, the idea here is to more evenly split the responsibility for congestion control between the routers and the end nodes.

Advanced Schemes

AQM: DECbit >> General Overview

- 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.

Advanced Schemes

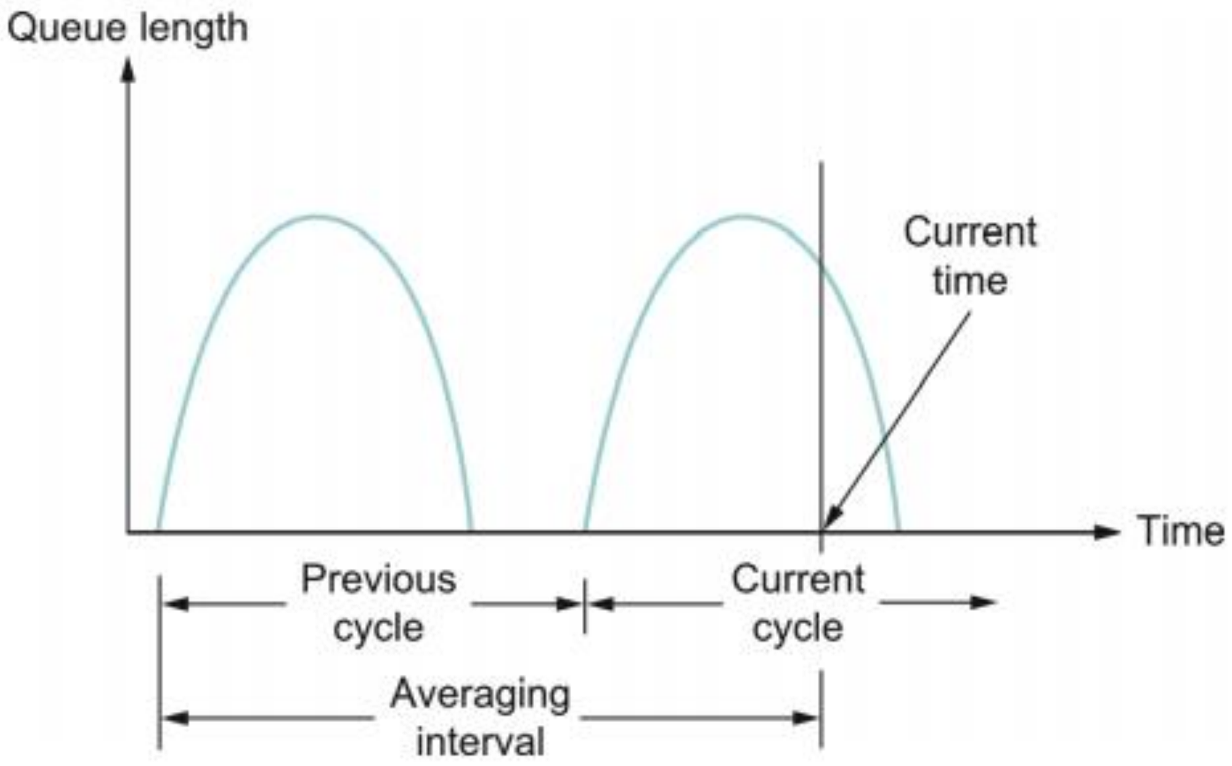
AQM: DECbit

- 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.
  - The router is busy when it is transmitting and idle when it is not.

Advanced Schemes

AQM: DECbit

- The Figure shows the queue length at a router as a function of time.



Advanced Schemes

AQM: DECbit

- 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 hence higher throughput) and increased idle time (and hence lower delay).
- In other words, a queue length of 1 seems to optimize the power function.

Advanced Schemes

AQM: DECbit

- Now turning our attention to the host half of the mechanism, 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.

Advanced Schemes

AQM: DECbit

- 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 (AIMD) makes the mechanism stable.

Advanced Schemes

AQM: RED

- A second mechanism, called random early detection (RED), is similar to the DECbit scheme in that
  - each router is programmed to monitor its own queue length and, when it detects that congestion is imminent, to notify the source to adjust its congestion window.

- RED, invented by Sally Floyd and Van Jacobson in the early 1990s, differs from the DECbit scheme in two major ways.

Advanced Schemes

AQM: RED

- 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.
- In case you have not already guessed, RED is designed to be used in conjunction with TCP, which currently detects congestion by means of timeouts (or some other means of detecting packet loss such as duplicate ACKs).

Advanced Schemes

AQM: RED

- As the “early” part of the RED acronym suggests, the gateway drops the packet **earlier** than it would have to so as to notify the source that it should decrease its congestion window sooner than it would normally have.

- In other words, 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.

Advanced Schemes

AQM: RED

- 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 (the tail drop policy of the previous section), 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.

Advanced Schemes

AQM: RED

- First, RED computes an average queue length using a weighted running average similar to the one used in the original TCP timeout computation.
- That is, AvgLen is computed as

$$\text{AvgLen} = (1 - \text{Weight}) \times \text{AvgLen} + \text{Weight} \times \text{SampleLen}$$

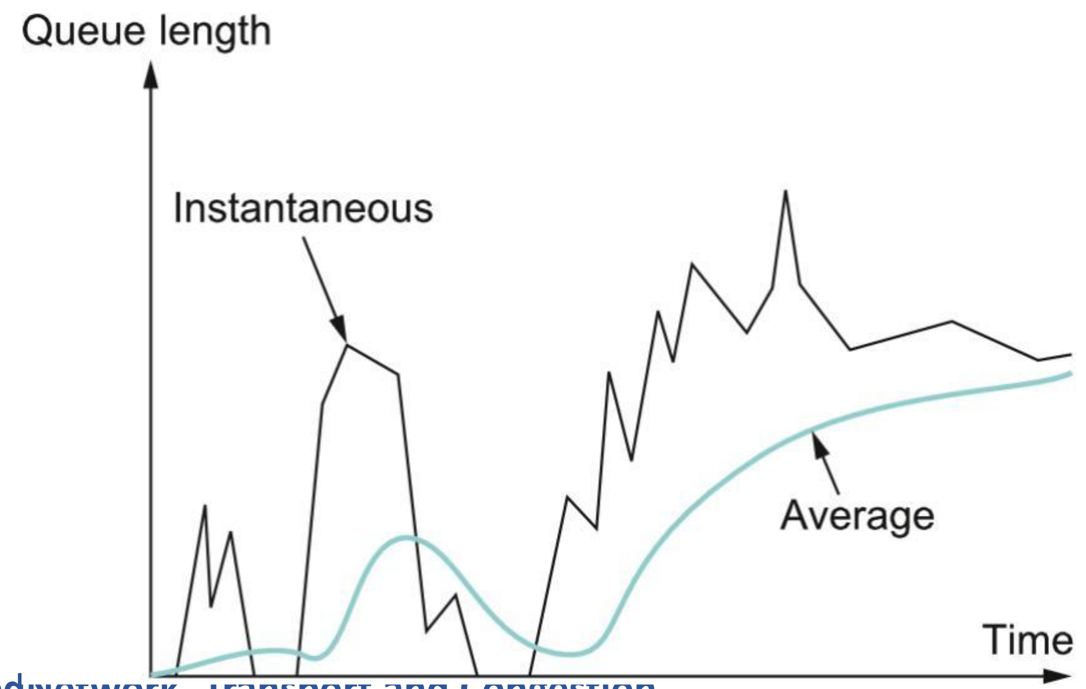
where 0 < Weight < 1 and SampleLen is the length of the queue when a sample measurement is made.

❖ In most software implementations, the queue length is measured every time a new packet arrives at the gateway. In hardware, it might be calculated at some fixed sampling interval.

Advanced Schemes

AQM: RED

- The weighted running average calculation tries to detect long-lived congestion, as indicated in the right-hand portion of the Figure, by filtering out short-term changes in the queue length.

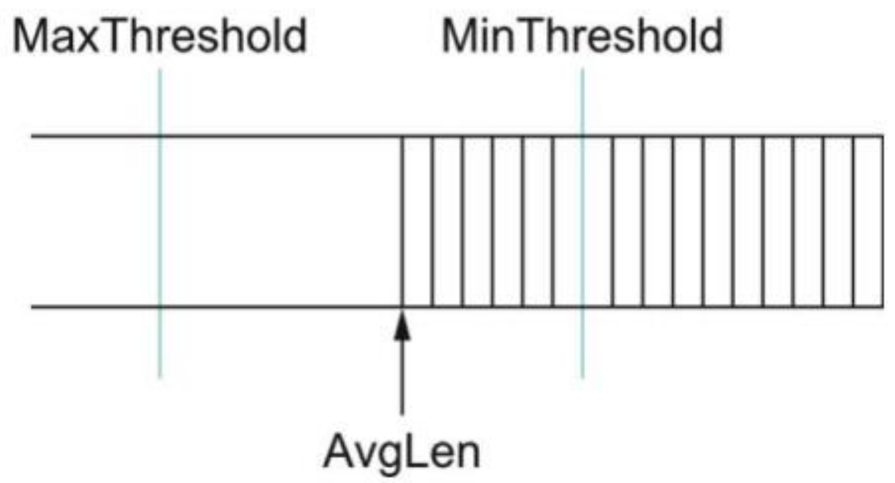


Advanced Schemes

AQM: RED

- The policy for dropping a packet in RED

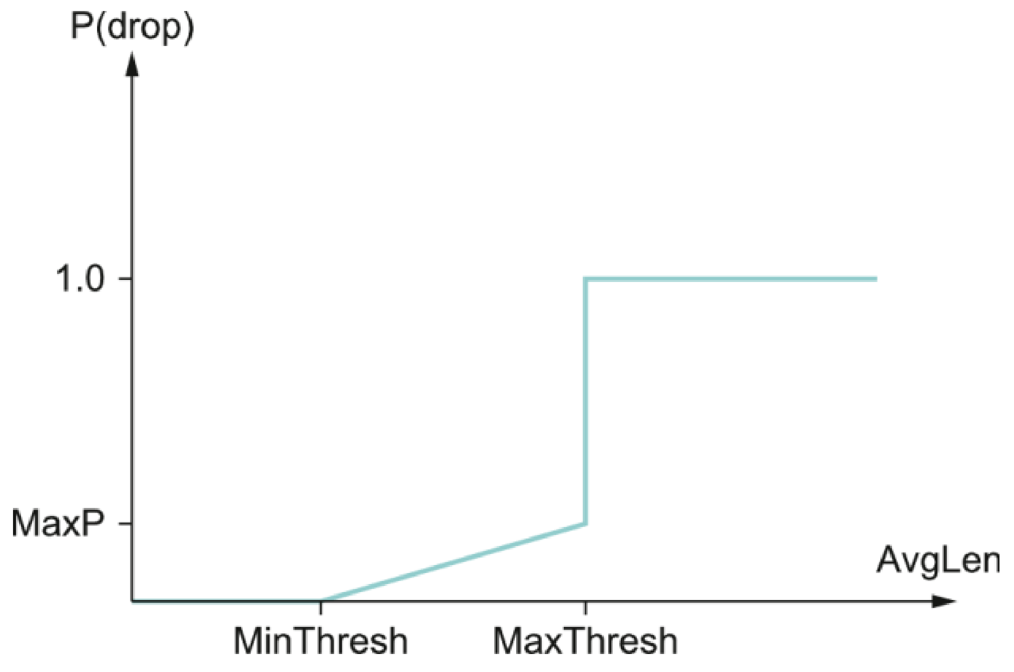
```
if AvgLen ≤ MinThreshold
    queue the packet
if MinThreshold < AvgLen < MaxThreshold
    calculate probability P
    drop the arriving packet with probability P
if MaxThreshold ≤ AvgLen
    drop the arriving packet
```



Advanced Schemes

AQM: RED

- Calculating P:
  - P is a function of AvgLen and Count
  - Count is the number of packets that have arrived since last reset
  - Reset happens when either a packet is dropped or AvgLen is above MaxThreshold



$$P = \frac{\text{TempP}}{1 - \text{count} \times \text{TempP}}$$

$$\text{TempP} = \text{MaxP} \times \frac{\text{AvgLen} - \text{MinThreshold}}{\text{MaxThreshold} - \text{MinThreshold}}$$

Advanced Schemes

AQM: ECN

- RED is the most extensively studied AQM mechanism, but it has not been widely deployed, due in part to the fact that it does not result in ideal behavior in all circumstances.

- It is, however, the benchmark for understanding AQM behavior.

- The other thing that came out of RED is the recognition that TCP could do a better job *if routers were to send a more explicit congestion signal.*

Advanced Schemes

AQM: ECN

- That is, instead of dropping a packet and assuming TCP will eventually notice (e.g., due to the arrival of a duplicate ACK), RED (or any AQM algorithm for that matter) can do a better job if it instead marks the packet and continues to send it along its way to the destination.

- This idea was codified in changes to the IP and TCP headers known as Explicit Congestion Notification (ECN).

Advanced Schemes

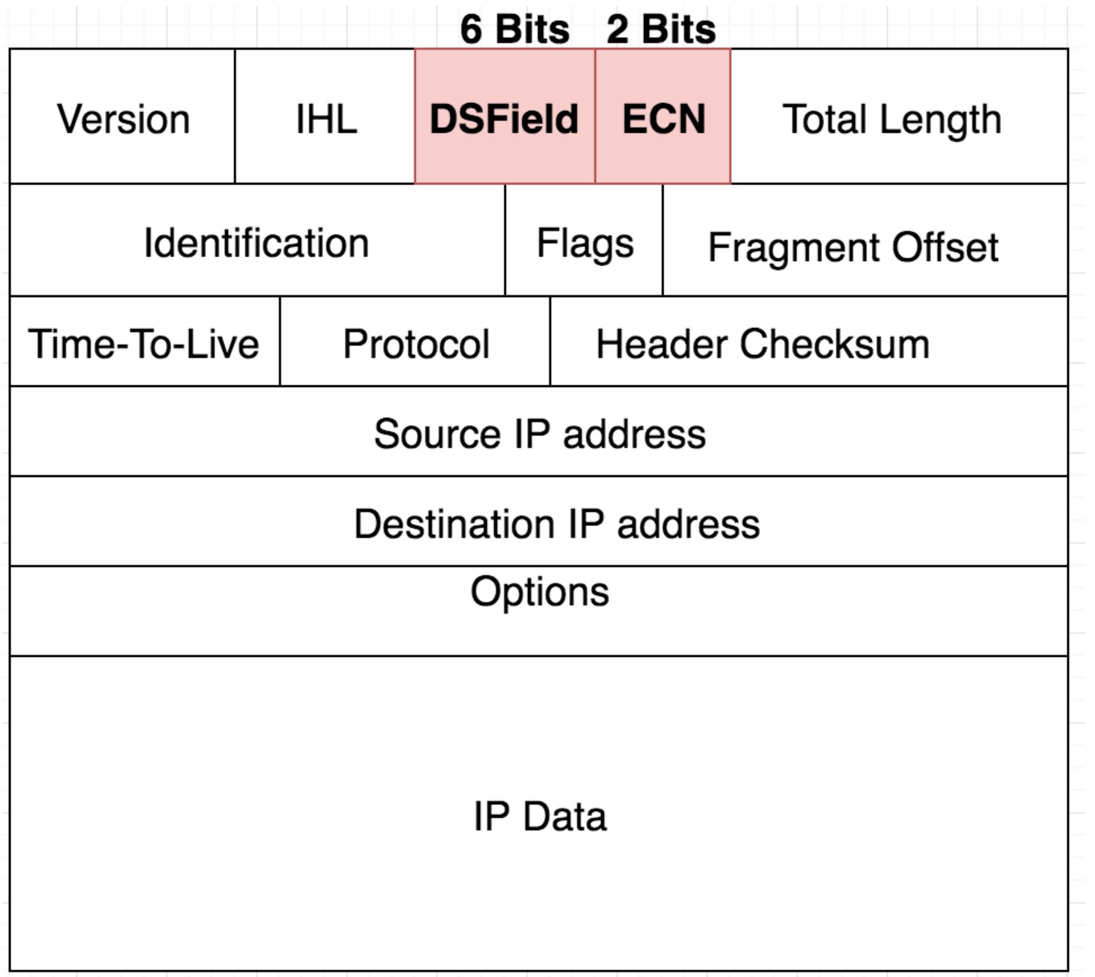
AQM: ECN

- Specifically, this feedback is implemented by treating **2 bits in the IP TOS field** as ECN bits.
- **One bit** is set by the source to indicate that it is ECN-capable, that is, able to react to a congestion notification.
  - **This is called the ECT bit (ECN-Capable Transport).**
- **The other bit** is set by routers along the end-to-end path when congestion is encountered, as computed by whatever AQM algorithm it is running.
  - **This is called the CE bit (Congestion Encountered).**

Advanced Schemes

AQM: ECN

- **2 bits in the IP TOS field** as ECN bits.



Advanced Schemes

AQM: ECN

- **In addition to these 2 bits** in the IP header (which are transport-agnostic), ECN also includes the addition of **two optional flags to the TCP header.**

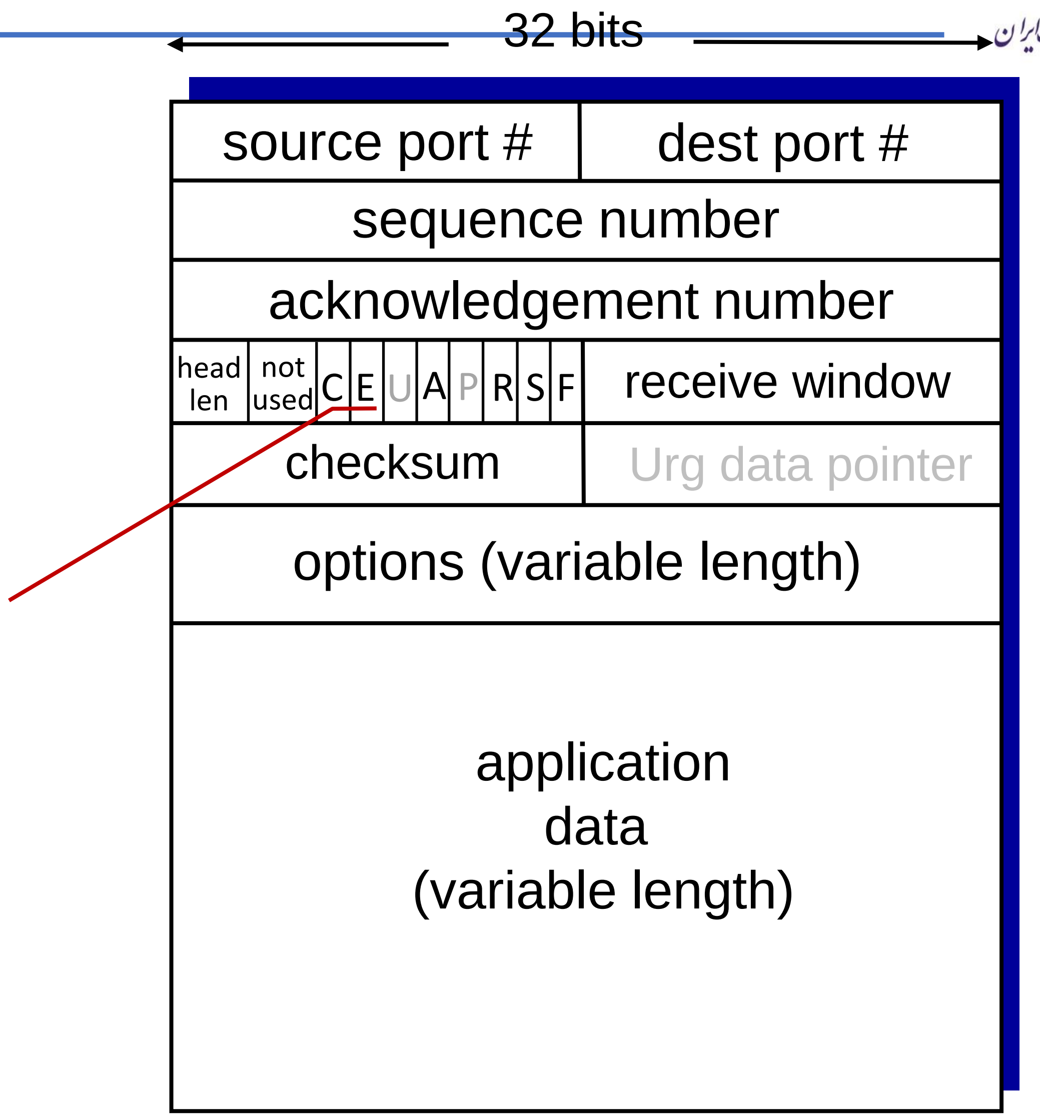
- **The first**, ECE (ECN-Echo), communicates from the receiver to the sender that it has received a packet with the CE bit set.

- **The second**, CWR (Congestion Window Reduced) communicates from the sender to the receiver that it has reduced the congestion window.

Advanced Schemes

AQM: ECN

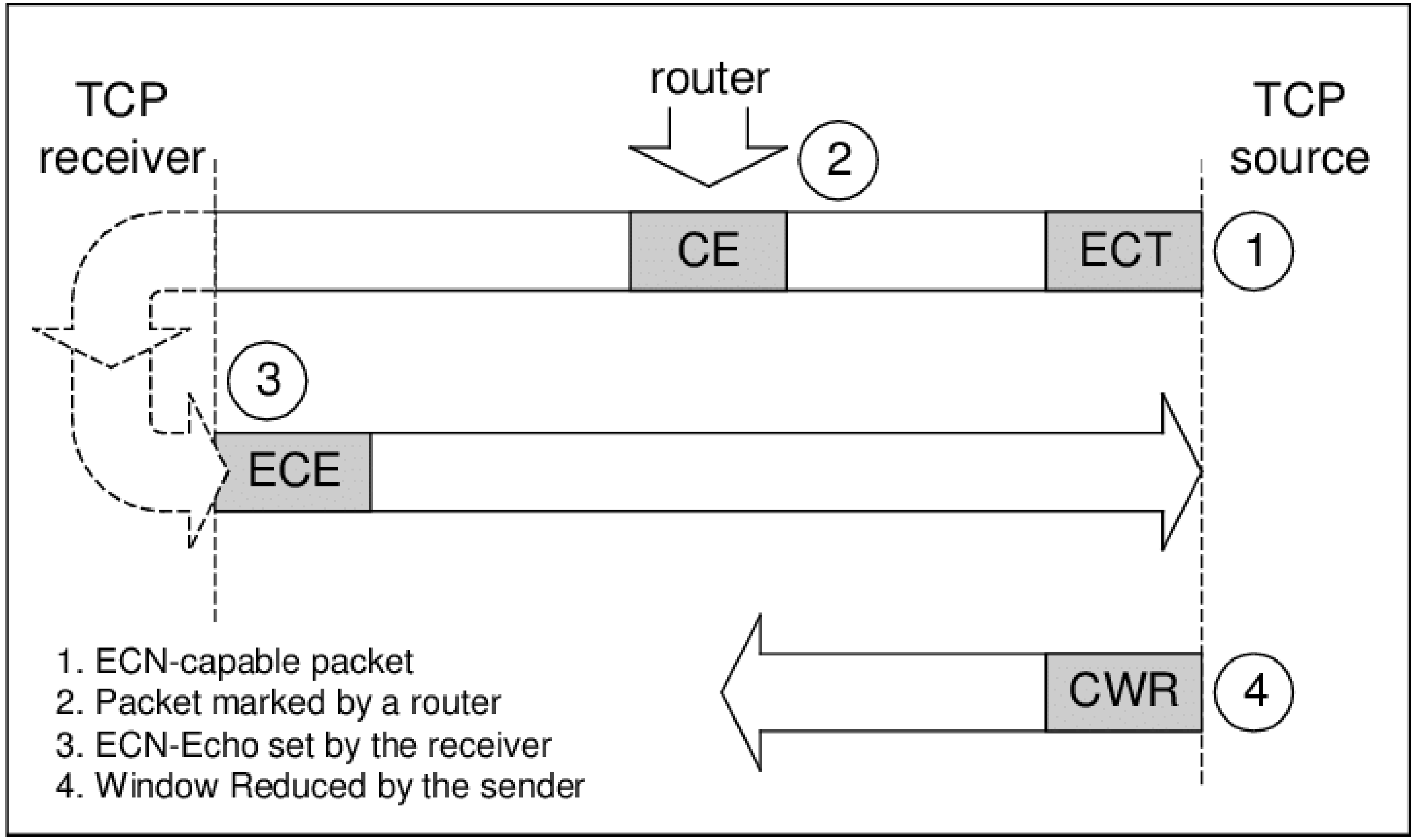
- two flags in the TCP header
- C: CWR flag
- E: ECN-echo (ECE) flag



Advanced Schemes

AQM: ECN

- After negotiation the transmission(IP and TCP bits) is as follows:

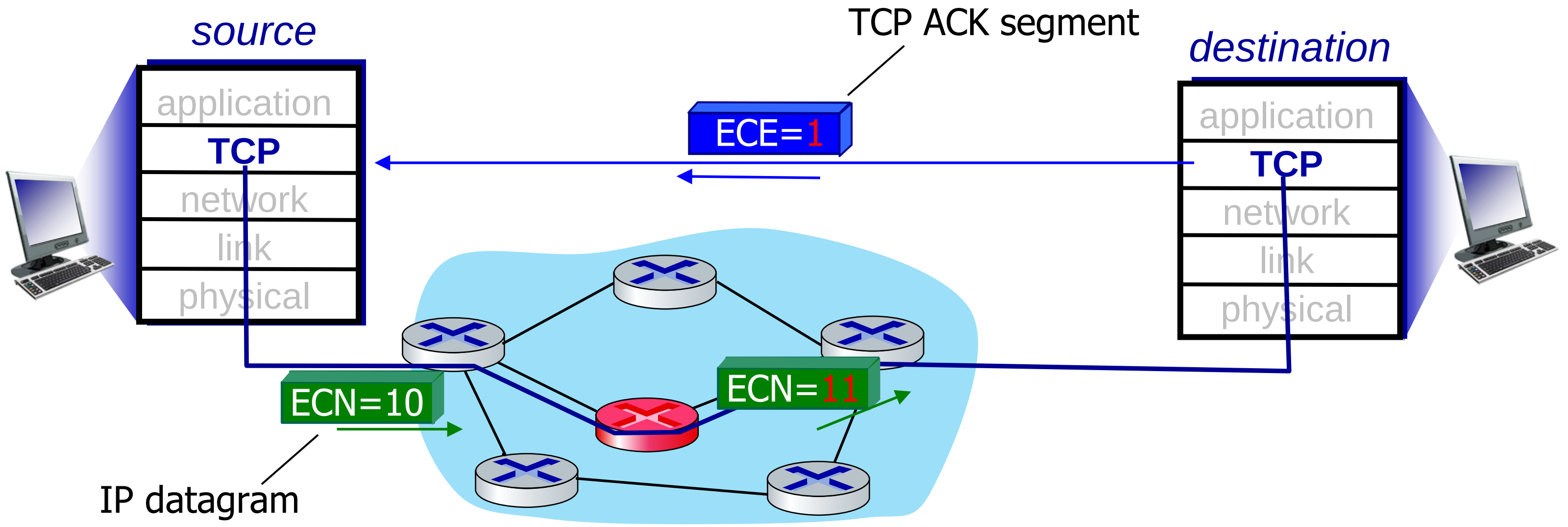


1. ECN-capable packet
2. Packet marked by a router
3. ECN-Echo set by the receiver
4. Window Reduced by the sender

Advanced Schemes

TCP deployments often implement *network-assisted* congestion control:

- two bits in IP header (ToS field) marked *by network router* to indicate congestion
  - *policy* to determine marking chosen by network operator
- congestion indication carried to destination
- destination sets ECE bit on ACK segment to notify sender of congestion
- involves both IP (IP header ECN bit marking) and TCP (TCP header C,E bit marking)



Advanced Schemes

AQM: ECN >> Remarks

- While ECN is now the standard interpretation of 2 of the 8 bits in the TOS field of the IP header, and support for **ECN is highly recommended, it is not required.**
- Moreover, **there is no single recommended AQM algorithm**, but instead, there is a list of requirements a good AQM algorithm should meet.
- Like TCP congestion control algorithms, every AQM algorithm has its advantages and disadvantages, and so we need a lot of them.
- There is one particular scenario, however, where the TCP congestion control algorithm and AQM algorithm are designed to work in concert: **the datacenter**.
  - We return to this use case at the end of this lecture.

Advanced Schemes

AQM: ECN >> Remarks

- ECN is **not** a congestion control algorithm like slow start restart and AIMD.
- This algorithm has only one responsibility to inform the sender about congestion building at the routers.
- Hence, ECN is a **congestion notification** or **congestion signaling** algorithm.
- It informs the sender about congestion so that respective measures can be taken to avoid that.
- ECN is **Congestion Signaling Mechanism** defined in RFC 3168. It came around in 1999 and went on to become finalized around in 2001.

Advanced Schemes

AQM: ECN >> Remarks

- ECN is similar to an earlier scheme known as the DECbit scheme; however, there are quite a few differences.

- In the DECbit scheme, the algorithm for generating the notification is coupled with the actual mechanics of how to carry notification. However, ECN just outlines how the congestion notification is carried to the destination and back to the source in IP networks. ECN is not tied to any queue management algorithm and is typically used in conjunction with RED.

- Further, the DECbit scheme uses only a single bit to indicate congestion as opposed to two bits in ECN.

Advanced Schemes

Source-Based (or Delay- Based) Approaches (Vegas, BBR, DCTCP)

- **Unlike the previous** congestion avoidance schemes, which depended on **cooperation from routers**, we now describe a strategy for detecting the incipient stages of congestion—before losses occur—**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.
- **Then** we describe **two specific mechanisms** in more detail.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP)

- The **general idea** of these techniques:

To watch for a sign from the network that some router's queue is building up and that congestion will happen soon if nothing is done about it.

- **For example**, 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.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP)

- **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.**

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP)

- **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 round-trip delays based on the product

$$(\text{CurrentWindow} - \text{OldWindow}) \times (\text{CurrentRTT} - \text{OldRTT})$$

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP)

(CurrentWindow - OldWindow) x (CurrentRTT - OldRTT)

So how to decide?

- **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**.

❖ Note that the window changes during every adjustment;
  - ➢ that is, it oscillates around its optimal point.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP)

- **Another change** seen as the network approaches congestion is the **flattening 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 half of 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.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- The mechanism 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.

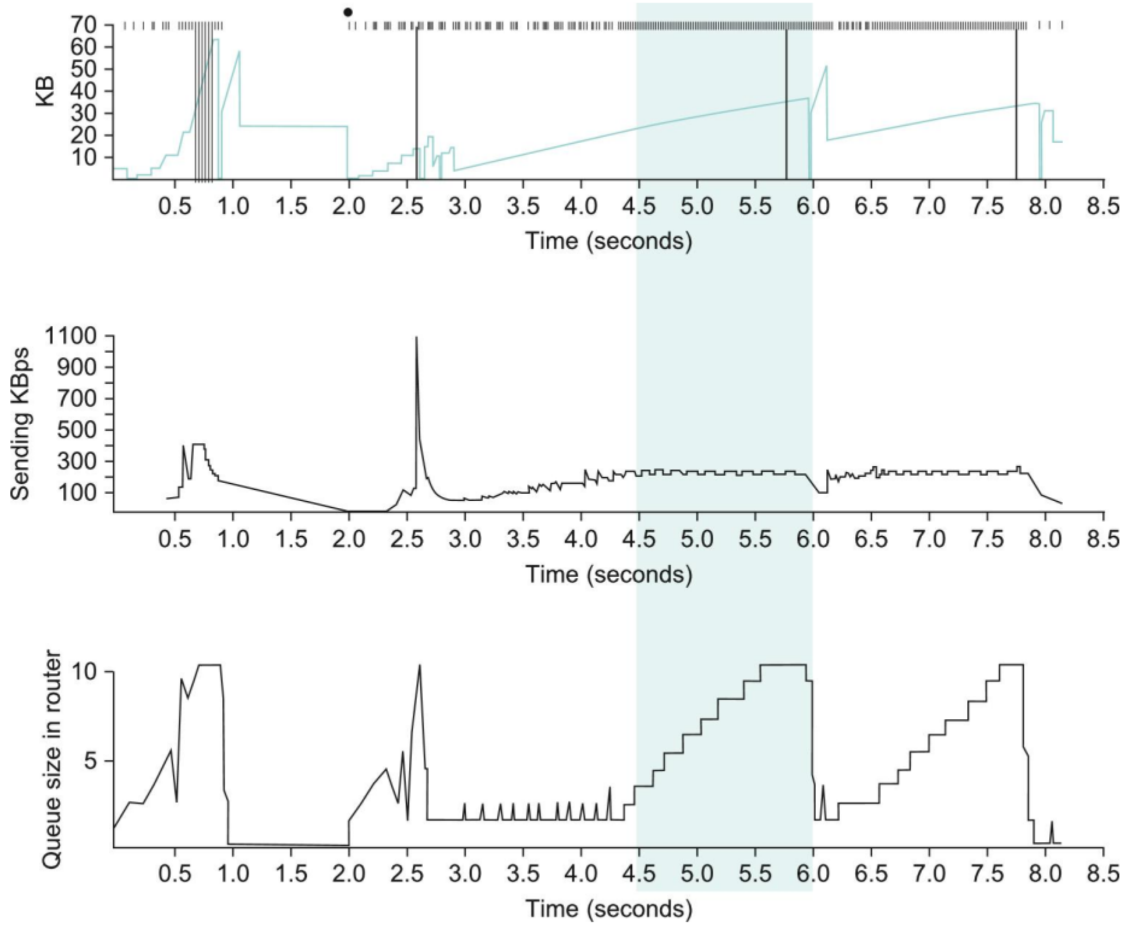
- However, **it differs from the previous algorithm** in the way it calculates throughput, and **instead of looking for a change in the slope of the throughput**, it compares the measured throughput rate with an expected throughput rate.

❖ The algorithm, **TCP Vegas, is not widely deployed in the Internet today**, but the strategy it uses has been **adopted by other implementations** that are now **being deployed**.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- Motivating scenario:



Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- The intuition behind the Vegas algorithm can be seen in the trace of standard TCP given in the Figure (previous slide).
- The **top graph** shown in Figure traces the connection’s congestion window.
- The **middle graph** shows the average sending rate as measured at the source.
- The **bottom graph** shows the average queue length as measured at the bottleneck router.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- All three graphs are synchronized in time.

- In the period between 4.5 and 6.0 seconds (shaded region), the congestion window increases (top graph).
  - **We expect** the observed throughput to also increase, but instead, it stays flat (middle graph).

- **This is because** the throughput cannot increase beyond the available bandwidth.
  - Beyond this point, any increase in the window size only results in packets taking up buffer space at the bottleneck router (bottom graph).

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- TCP Vegas uses this idea to **measure and control** the amount of **extra data** this connection has in transit.
- By “extra data” we mean data that the source wants to transmit to match exactly the available bandwidth of the network.
- The goal of TCP Vegas is to maintain the “right” amount of extra data in the network.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- We will have:
  - **too much extra data** >> long delays and possibly lead to congestion.
  - **too little extra data** >> 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, not only on dropped packets.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

➢ We now describe the algorithm in detail.

- **First**, define a given flow’s **BaseRTT**
  - the RTT of a packet when the flow is not congested.
  - In practice, 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

$$\text{ExpectedRate} = \text{CongestionWindow} / \text{BaseRTT}$$

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- **Second**, TCP Vegas calculates the current sending rate, **ActualRate**.
- 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.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- **Third**, TCP Vegas compares ActualRate to ExpectedRate and adjusts the window accordingly.

- We let

$$\text{Diff} = \text{ExpectedRate} - \text{ActualRate}.$$

- Note that Diff is positive or 0 by definition,
  - since ActualRate >ExpectedRate implies that we need to change BaseRTT to the latest sampled RTT.

Advanced Schemes

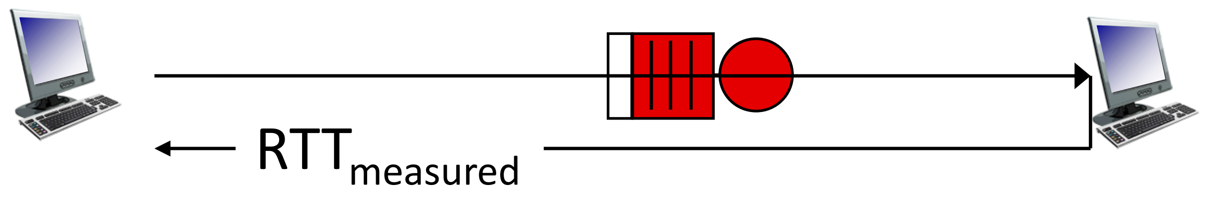
Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- We also define two thresholds, α < β, roughly corresponding to
  - α: having too little extra data in the network
  - β : having too much extra data in the network

- **Strategy:**
  - When Diff < α, TCP Vegas increases the congestion window linearly during the next RTT,
  - when Diff > β, TCP Vegas decreases the congestion window linearly during the next RTT.
  - TCP Vegas leaves the congestion window unchanged when α < Diff < β.

Advanced Schemes: TCP Vegas Overview

Keeping sender-to-receiver pipe "just full enough, but no fuller": keep bottleneck link busy transmitting, but avoid high delays/buffering



$$\text{measured throughput} = \frac{\text{\# bytes sent in last RTT interval}}{RTT_{measured}}$$

TCP-Vegas Delay-based approach:

- RTT_{min} - minimum observed RTT (uncongested path)
- uncongested throughput with congestion window `cwnd` is cwnd/RTT_{min}

```
if measured throughput "very close" to  uncongested throughput
    increase cwnd linearly              /* since path not congested */
else if measured throughput "far below" uncongested throughout
    decrease cwnd  linearly             /* since path is congested */
```

Advanced Schemes

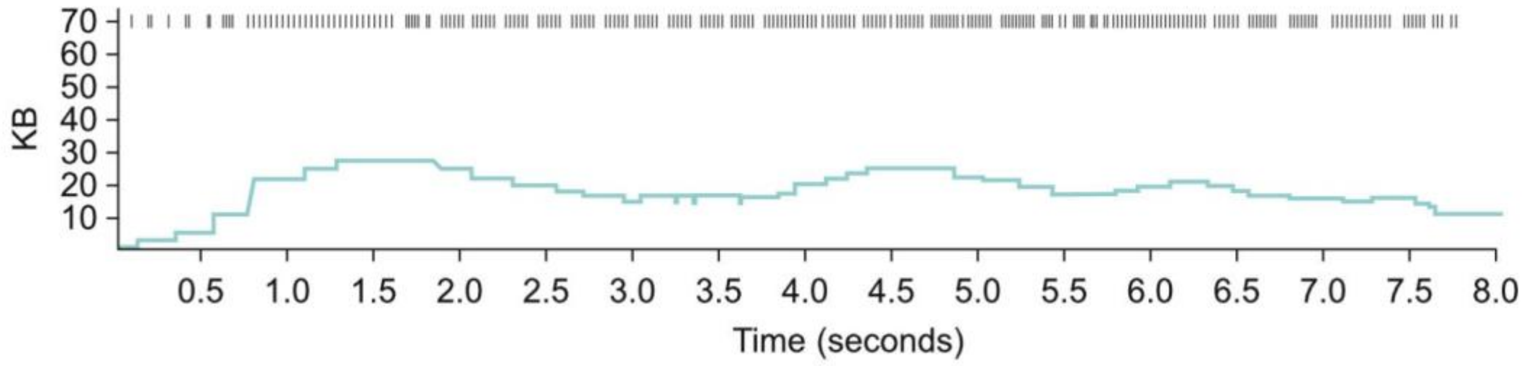
Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

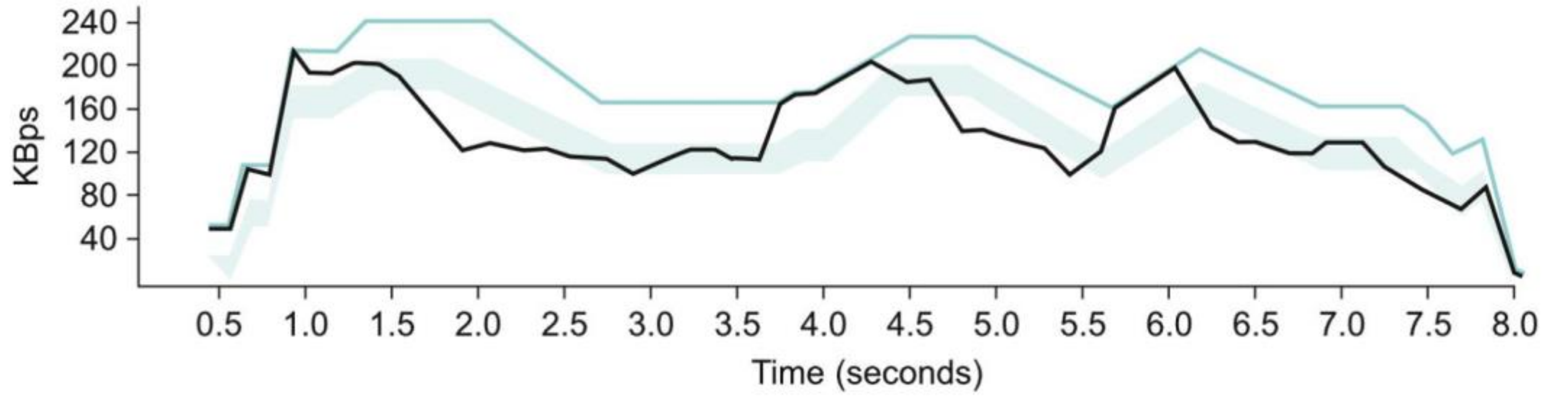
- **Intuitively**,
  - We can see that the farther away the actual throughput gets from the expected throughput, the more congestion there is in the network, which implies that the sending rate should be reduced. The β threshold triggers this decrease.
  - On the other hand, when the actual throughput rate gets too close to the expected throughput, the connection is in danger of not utilizing the available bandwidth. The α threshold triggers this increase.
  - The overall goal is to keep between α and β extra bytes in the network.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- **Top**: Congestion Window
- **Bottom**: Rates and Strategy





Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- The Figure traces the TCP Vegas congestion avoidance algorithm.
- **The top graph** traces the congestion window.
- **The bottom graph** traces the expected and actual throughput rates that govern how the congestion window is set.
- It is this bottom graph that best illustrates how the algorithm works.
  - The colored line tracks the ExpectedRate, while the black line tracks the ActualRate.
  - The wide shaded strip gives the region between the α and β thresholds; the top of the shaded strip is α kBps away from ExpectedRate, and the bottom of the shaded strip is β kBps away from ExpectedRate.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- **The goal** is to keep the ActualRate between these two thresholds, within the shaded region.
  - Whenever **ActualRate falls below the shaded region** (i.e., gets too far from ExpectedRate), TCP Vegas decreases the congestion window because it fears that too many packets are being buffered in the network.
  - Likewise, whenever **ActualRate goes above the shaded region** (i.e., gets too close to the ExpectedRate), TCP Vegas increases the congestion window because it fears that it is underutilizing the network.

Advanced Schemes

Source-Based Approaches (Vegas, BBR, DCTCP) >> Vegas

- Finally, you will notice that TCP Vegas decreases the congestion window linearly.

- So, it seems to be 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.