March 17 Lecture Transcript – (TCP Congestion)

Slide 2:
As discussed in earlier lectures, the (core) Internet is composed of a system of interconnected routers, and each of those routers has its own limitations in terms of (e.g.) the number of packets that they can receive and process as well as the bit-per-second speeds on their incoming and outgoing connections/links. If the load on individual Internet routers exceeds their actual capacity, the packets passed through these routers will be delayed and/or dropped. Such network conditions/states are referred to as 'network congestion'.

TCP protocol is designed in a way that:
1) it enables the sending machines to recognize whether/when the network/Internet is experiencing congestion, and
2) allows the sending machine to adequately reduce its own rate of sent/transmitted packets so as to help the network recover from the on-going congestion …

Slide 3:
The two graphs/figures provided in this slide illustrate the following general phenomena:

1) As the amount of traffic (load) in a network approaches the theoretical capacity/limit of this network, the delay experienced by individual packets starts to exponentially increase (and in the worst case scenario may become 'infinity' – i.e., the packets are dropped and they never arrive to the intended destination).
2) As the amount of traffic (load) in a network increases, the throughput (i.e., the number of successfully routed/delivered) packets will increase BUT only as long as the overall load is smaller than the theoretical capacity of the network. Once the load increases the theoretical capacity, the throughput will start to decrease …

Slide 4:
Theoretically, they 2 main ways how a network can cope with congestion (one approach illustrated in this and the other approach in the subsequent slide).

The first approach – the so-called Network Assisted Congestion Control – assumes that, in the time of congestion, the network routers will/could send explicit messages/packets to the network's end-devices warning them about congestion and demanding that the end-devices reduce the number of transmitted packets. Although this approach may seem reasonable, one of its main disadvantages is that it would impose extra workload on routers (e.g., requires them to send 'warning' packets to all end-devices) at the time when their resources are already heavily used/burdened.

ICMP Source Quench packets would be an example of this approach, though as we know the use of these packets – for the reasons already explained – is not mandatory for Internet routers …

Slide 5:
Another approach – the so-called End-to-End Congestion Control – assumes that there are no explicit messages sent from the routers to the end-devices. Instead, the end devices (on their own) perform

continuous monitoring of sent and received traffic, and based on those observations make their own independent conclusions about whether the network is congested or not. And this mechanism is exactly how TCP attempts to deal with and help the network relieve potential congestions …

Slide 6:

Assume a TCP connection between 2 machines – sending machine A and receiving machine B. A TCP packet sent from A to B could be lost because of either of the following:

1) Packet is dropped at the receiving machine B, due to B being overwhelmed with the overall amount of received data (from A and/or other machines);
2) Packet is dropped at the intermediate routers, due to the routers being overwhelmed with the overall amount of packets that they need to route/forward …

The same way that TCP deploys ReceiveWindow as a way of informing A that B is overwhelmed and A should 'slow down', TCP also deploys the so-called CongestionWindow as a way of trying to inform A about potential network congestion and make it reduce the amount of transmitted packets …

Ultimately, the amount of traffic that A sends will correspond to the smaller of the two windows.

Slide 7:

This slide shows that in a real-world TCP connection, in addition to being superimposed, ReceiveWindow and CongestionWindow are also continuously recalculated/re-evaluated to best reflect the current state/capabilities of the receiving machine as well as the network …

Slide 8:

In this slide we discuss the actual mechanism that are used to control/adjust the size of CongestionWindow.

Clearly, the key factor that determines how big/small the CongestionWindow of a sending machine should be is the amount of successfully (or unsuccessfully) transmitted packets. For example, if for every transmitted packet the sending machine receives an acknowledgement within a predefined amount of time, then it is reasonably to assume that there is no congestion in the network, and CongestionWindow can/should grow. On the other hand, if for some transmitted packets the respective acknowledgments never arrive, then it is reasonable to assume that there is (likely) congestion in the network, and the size of CongestionWindow should be reduced.

Now, the actual algorithm deployed by TCP to control the size of CongestionWindow, in fact, proceeds in 3 stages:

- Slow Start (or Exponential Increase) Stage
- Additive Increase (or Linear Increase) Stage
- Multiplicative Decrease Stage

Slide 9:

In the first – Slow Start – stage, CongestionWindow size is initially set to 1 packet, and only that 1 packet is transmitted into the network. (This way, the sending machine is 'carefully probing' the network for potential evidence of congestion.) If the acknowledgment for this packet arrives back successfully, the size of CongestionWindow is doubled (is now 2), and 2 new packets are sent into the network. If/when the acknowledgment for these 2 packets arrive back, the size of CongestionWindow is further doubled to 4. Etc. The doubling of CongestionWindow will continue until some predefined window **Threshold** (size) is reached.

Through the Slow Start mechanisms, the sending machine starts carefully, but (at the same time) if there is no evidence of congestion, the sending machine will keep increasing the size of CongestionWindow 'rather optimistically/aggressively' …

Slide 10:

The second – Additive Increase – stage is triggered (i.e., it starts) once CongestionWindow size reaches the predefined **Threshold**. During this stage, every successfully received acknowledgment will result in the increase of CongestionWindow only by one, which (in fact) implies a linear increase of CongestionWindow size.
As such, Additive Increase stage is clearly far more conservative than the Slow Start, as it very 'cautiously' keeps increasing the number of transmitted packets. (During this stage, there is only 1 more packet being sent in every subsequent RTT amount of time.)

The third stage – Congestion Avoidance – starts at the first detection of a lost packet, which is considered to be an indication of network congestion. In this stage (once a packet loss is detected) the value of Threshold is set to ½ of the actual CongestionWindow at the time of packet loss and the size of CongestionWindow itself is reduced back to 1.

Slide 11:

This slide provides a visual illustration of the three CongestionWindow control stages.

Slide 12:

The TCP congestion control algorithm described in the previous slides is, in fact, an early/original version of TCP – the so-called TCP Tahoe Algorithm.

Over time, some improvements of the original TCP congestion control algorithm were proposed, and one of them is TCP Reno Algorithm. In TCP Reno, the provision is made for cases in which a lost packet is not a sign of serious congestion. In particular, if upon detecting a lost packet the sending machine receives 3 duplicate acknowledgment (for the last successfully transmitted packet), the sending machine can justifiably assume that the network conditions are reasonably good – i.e., the congestion is not severe. And, as a result, the sending machine will (still) adjust Threshold value to ½ of the current CongestionWindow, but then instead of readjusting/reducing the size of CongestionWindow to 1 it will reduce it just to Threshold. In other words, in order to deal with this (mild) congestion, the sending machine will move straight to Additive Increase stage, instead of going back to Slow Start stage.

Slide 13:

This slide summarizes the characteristic of three stages of TCP Reno Algorithm.

Slide 14:

This slide illustrates the operation of TCP Reno through a numerical example.

Slide 15:
This slide provides a comparative summary of TCP vs/against UDP.

Slide 16:
In short, this numerical questions assumes a TCP connection with Reno algorithm for congestion control. The algorithm asks about the worst-case scenario of a packet loss. That is, if we assume that one out of 7 transmitted packet could potentially get lost, which of those 7 would have the worst consequences in terms of the overall transmission time.

Clearly, if a packet loss occurs during the stage of such a connection when there are lass than 4 packets in transit, such a packet loss would not be followed by 3 consecutive successfully received acknowledgments and (as a consequence) would result in the more dramatic form of congestion control – i.e., it would trigger 'Slow Start' and cause the reduction of CongestionWindow all the way back to 1.
On the other hand, if a packet loss occurs when there are 4 or more packets in transit, (ultimately) the sending machine would quickly receive 3 acknowledgments for those other successfully transmitted packets. And, as a result, the less dramatic form of congestion control would take place (Linear Increase), which would result in an overall shorter time to transmit all packets …