ECE 333: Introduction to Communication Networks
Fall 2001

Lecture 28: Transport Layer III

- Congestion control (TCP)

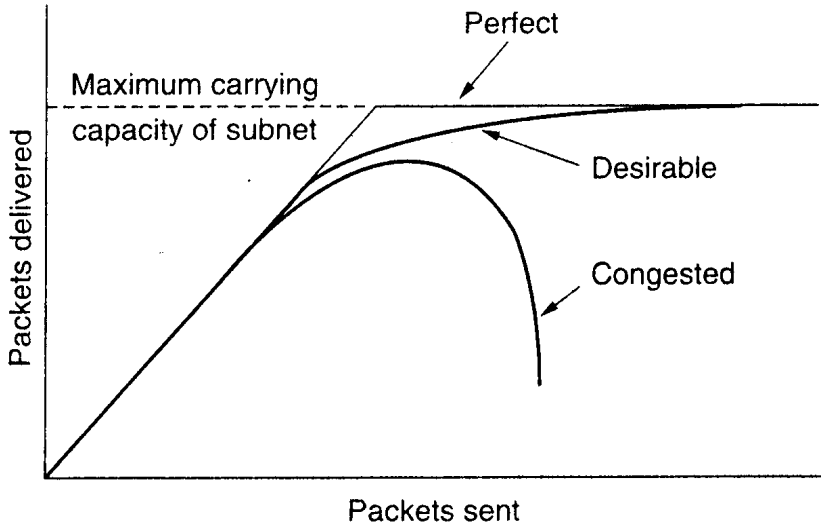
In the last lecture we introduced the topics of ***flow control*** and ***congestion control***. Both of these refer to methods that are used to regulate the rate at which a source can send data. Flow control often denotes techniques that regulate a source's rate based on the buffer space at the destination (independent of the network). Congestion control denotes techniques that regulate a source's rate based on congestion in the subnet.[1] We saw that in TCP, flow control is implemented by including a "receive window" in each TCP segment sent over the reverse link. This window is then used by the sliding window ARQ protocol to provide flow control.

Today we will discuss congestion control. First we look at the problem of congestion and some general approaches to congestion control. Then we describe how congestion control is implemented in TCP.

[1] This terminology is not universal and in many cases a distinction is not made between these two issues. Indeed, both flow and congestion control can be viewed as parts of the same problem.

Congestion

Congestion arises when the total load on the network becomes too large. This leads to queues building up and to long delays (recall the M/G/1 model). If sources retransmit messages, then this can lead to even more congestion and eventually to ***congestion collapse***. This is illustrated in the figure below. Notice, as the offered load increase, the number of packets delivered at first increases, but at high enough loads, this rapidly decreases.



For example, originally the Internet (TCP/IP) did not implement congestion control. This led to a series of congestion collapses starting in 1986. In Oct. of 1986, the data throughput between Lawrence Berkeley Laboratory and UC Berkeley dropped from 32 kbps to 40 bps[2] [V. Jacobson, "Congestion Avoidance and Control", *Proc. of SIGCOMM '88.*] To avoid this type of behavior, congestion control was incorporated into TCP.

The first order goal of congestion control is to avoid this type of congestion collapse. While avoiding congestion, it is still desirable to get the best performance possible. Congestion control can also be used to ensure that users get a desired QoS (i.e. throughput, delay). Another issue that congestion control needs to address is ensuring ***fairness*** between different sessions; in other words to sessions with the same requirements should receive equitable treatment from the network. (Exactly what is a fair treatment can be interpreted in several ways.)

[2] V. Jacobson, "Congestion Avoidance and Control", *Proc. of SIGCOMM '88*

Approaches to Congestion Control

Congestion control may be addressed at both the network level and the transport layer.

At the network layer possible approaches include

- Packet dropping - when a buffer becomes full a router can drop waiting packets - if not coupled with some other technique, this can lead to greater congestion through retransmissions.
- Packet scheduling - certain scheduling policies may help in avoiding congestion - in particular scheduling can help to isolate users that are transmitting at a high rate.
- Dynamic routing - when a link becomes congested, change the routing to avoid this link - this only helps up to a point (eventually all links become congested) and can lead to instabilities (see Lecture 23).
- Admission control/Traffic policing - Only allow connections in if the network can handle them and make sure that admitted sessions do not send at too high of a rate - only useful for connection-oriented networks.

An approach that can be used at either the network or transport layers is

- Rate control - this refers to techniques where the source rate is explicitly controlled based on feedback from either the network and/or the receiver.

For example, routers in the network may send a source a "choke packet" upon becoming congested. When receiving such a packet, the source should lower it rate.

These approaches can be classified as either "congestion avoidance" approaches, if they try to prevent congestion from ever occurring, or as "congestion recovery" approaches, if they wait until congestion occurs and then react to it. In general, "an ounce of prevention is worth a pound of cure."

Different networks have used various combinations of all these approaches. Traditionally, rate control at the transport layer has been used in the Internet, but new approaches are beginning to be used that incorporate some of the network layer techniques discussed above.

Congestion control in TCP

TCP implements end-to-end congestion control. V. Jacobson proposed the basic TCP congestion control algorithm in 1988. IP does not provide any explicit congestion control information; TCP detects congestion via the ACK's from the sliding-window ARQ algorithm used for providing reliable service.

Specifically, assume that the RTT for a session is known, so that the proper time-out time can be used. When the source times out before receiving an ACK, the most likely reason is because a link became congested. TCP uses this as an indication of congestion. In this case, TCP will slow down the transmission rate.

TCP controls the transmission rate of a source by varying the window size used in the sliding window protocol (this is the same technique used for flow control). Recall, (see lecture 10) that for a window of size of W packets, the effective rate under a sliding window protocol (ignoring errors) is given by

$$R_{eff} = \frac{Wd}{d + IR} \text{ (assuming that } W < 1 + (IR)/d)$$

Here I is the round-trip time, and d is the number of bits per packet. Thus a smaller value of W will result in a smaller effective rate.

Timeout time

Before discussing how TCP adjusts the rate when a timeout occurs. We first look at how TCP chooses a value for the timeout time. The timeout time should be set approximately equal to the round trip time, i.e. the time from when a segment is transmitted until an ACK for the segment is received. Choosing to small of a time-out time will result in unneeded retransmissions, and choosing too large a value will result in long delays before a packet is retransmitted. However, at the transport layer, the round trip time will vary with each pair of host (or more precisely with each route between any pair of hosts). The round-trip time will also depend on the queueing delays within the network; these will vary with each segment sent.

In TCP the time-out value is based on estimates of the round trip time (RTT). Specifically, for each packet sent the RTT is measured. Let Est_RTT be the current estimate of the RTT and let Sample_RTT be a new measured value. The new estimate of the RTT is then formed as follows:

$$\text{Est\_RTT} = \alpha(\text{Est\_RTT}) + (1-\alpha)\,\text{Sample\_RTT}$$

Here α is a parameter between 0 and 1; a typical value is 0.125. Thus the transmitter's estimate of the RTT is a moving average of the sample RTT's.

The timeout value used by TCP is equal to the Est_RTT plus a margin to account for variations due to changes in queueing delays. This margin is related to the variance of the RTT. In other words, a connection with a large variance in the RTT will have a larger margin. Specifically, in TCP

$$\text{Timeout} = \text{Est\_RTT} + 4\,(\text{Dev})$$

where Dev is also recursively calculated for each new sample using

$$\text{Dev} = (1-\alpha)\,\text{Dev} + \alpha\,|\text{Sample\_RTT} - \text{Est\_RTT}|$$

(This quantity does not give the true variance or standard deviation, but instead a related quantity called the ***mean deviation***. This appears to have been chosen because it is easier to calculate.)

When $|\text{Sample\_RTT} - \text{Est\_RTT}|$ is large, the Dev will increase, leading to a larger margin for the timeout as desired.

Next we look at how TCP changes the rate at which a source can send data. This is done by varying the window size used by the sliding-window algorithm. The window size is the largest number of unacknowledged data bytes that a source can send at any time. Recall, in TCP data is viewed as a byte stream, and all sequence numbers refer to byte number. The largest segment that a TCP implementation can send is referred to as the Maximum Segment Size (MSS), the value for the MSS can vary with implementation. For congestion control, TCP requires each source to keep track of a ***congestion window***, W_C, which indicates the maximum number of MSS-sized segments that the source can send, based on congestion.

The window size used by TCP is then given by

$$\min(W_C(\text{MSS}), R_w),$$

where R_w is the receive window used for flow control (see Lecture 27).

In the following we will assume that R_w is very large and can be ignored - this way we can focus on the behavior of the congestion window.

We also assume that the transmitter always has data available to send, and that each segment sent contains MSS bytes. With these assumptions we describe the basic idea of TCP congestion control.

TCP Congestion control

TCP tries to adaptively find the "right" congestion window size for the network. If window is too large, network will become congested; if too small then inefficient. More over, the "right" window size may change over time, depending on network load.

When segments are successfully acknowledged, without the sender timing out, TCP increases the window size. Whenever TCP times-out while waiting for a segment it decreases the window size. In this manner TCP continually adapts the window size.

In addition to the congestion window, the sender also keeps track of a parameter called the ***threshold***; we denote this parameter by T_C.

Both W_C and T_C are adapted over time. Initially a connection sets W_C=1. Assume that T_C is some value greater than 1.

When segments are acknowledged before the transmitter times out TCP increases the window size using the following rules:

- If $W_C \leq T_C$ then set $W_C = W_C + 1$ for every successful ACK that is received.
- If $W_C > T_C$ then set $W_C = W_C + 1$ after W_C successful ACK's are received.

We see that starting at $W_C = 1$, the congestion window will increase by one for every ACK that is received until it becomes larger than the threshold, this phase is called ***slow start***. After the congestion window becomes larger than the threshold, the congestion window is increased by one for every congestion window worth of ACKs that are received; this phase is called the ***congestion avoidance*** phase.

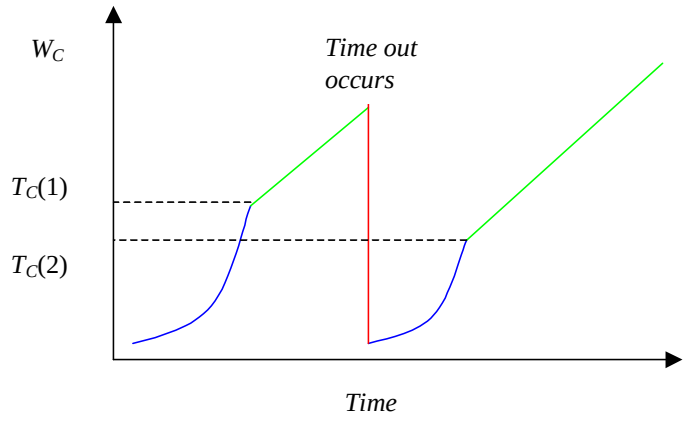
Assume that in approximately one RTT, a congestion window worth of data can be sent and acknowledged (this is reasonable if the time to send a windows worth of data is much less than the RTT). With this assumption, during the slow start phase, the congestion window will double during each RTT, i.e. it will grow exponentially fast. During the congestion avoidance phase, the window will increase by 1 during each RTT; in this case, it is growing at a linear rate.

When the transmitter times out, it does the following:

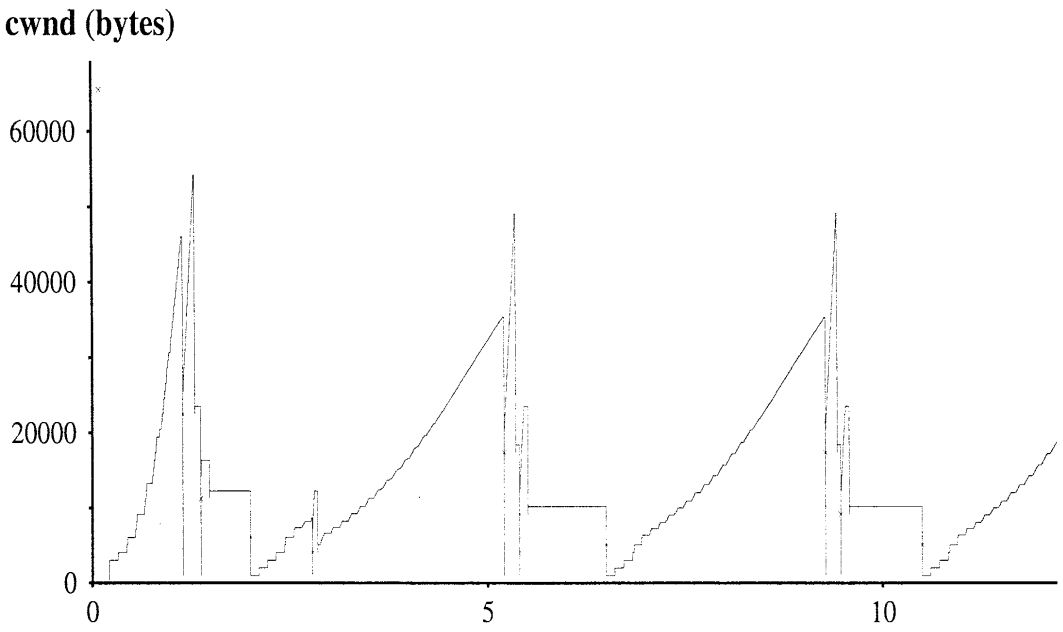
- Set $T_C = W_C/2$ and set $W_C = 1$.

In other words, the threshold is set half of the value of the congestion window before the time out occurred, and the congestion window is reduced to 1 (causing the algorithm to re-enter the slow start phase).

An example of this algorithm is shown below. The slow start phase is shown in blue and the congestion avoidance phase is shown in green. In this figure one time out occurred as indicated. Here $T_C(1)$ indicates the initial threshold and $T_C(2)$ indicates the threshold after the time out occurs.



An actual trace of this algorithm is shown next.

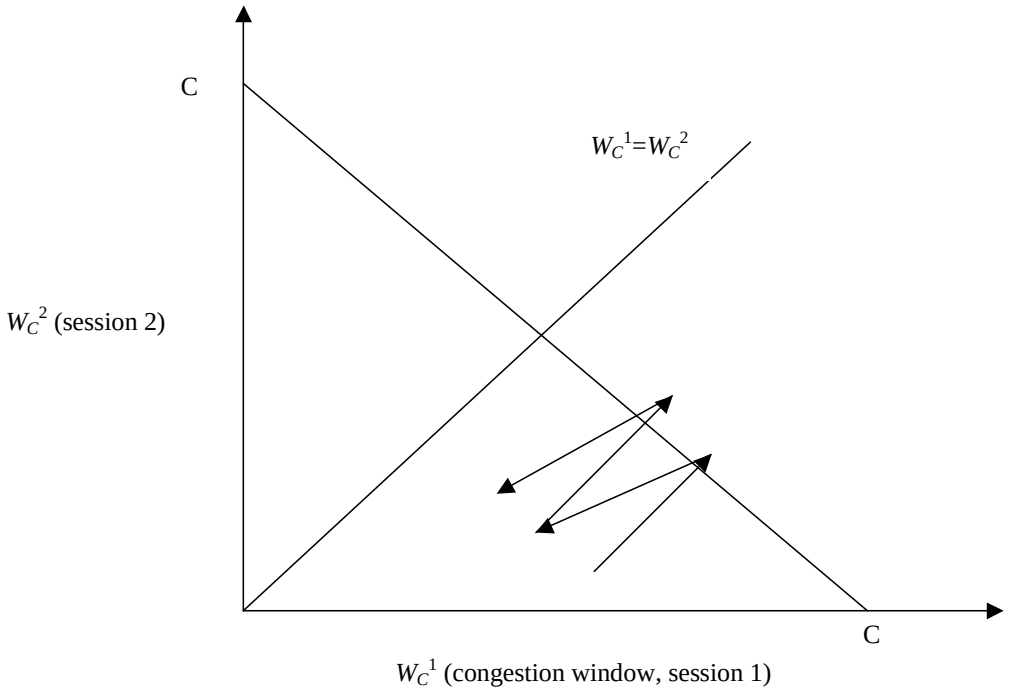


Again the vertical axis is the congestion window and the horizontal axis is time.

Notice, if congestion is detected, then at the end of the next slow start phase, the congestion window will be 0.5 times it value before congestion occurred. Thus, if we ignore the slow start phase then TCP simply increases the congestion window additively each time no congestion is detected and decreases the congestion window multiplicatively each time congestion is detected. For this reason, TCP is described as an ***Additive-Increase, Multiplicative-Decrease*** (***AIMD***) algorithm.

This type of algorithm has been shown to have some nice fairness properties. We illustrate this with an example of two TCP connections sharing a single link, we assume that both sessions have the same MSS and RTT, and both always have data to send. Assume that the link has a capacity of *C* MSS-segments, (i.e. if the sum of rates (window sizes) of the two users exceed *C* then congestion sets in).

The figure below illustrates how an AIMD algorithm drives the two users to a fair (equal) allocation of the link. The axes are the congestion windows of each session. Both increase additively until exceeding the capacity then decrease multiplicatively towards the origin. This behavior drives the congestion windows toward the fair (equal) rate line over time.



Versions of TCP

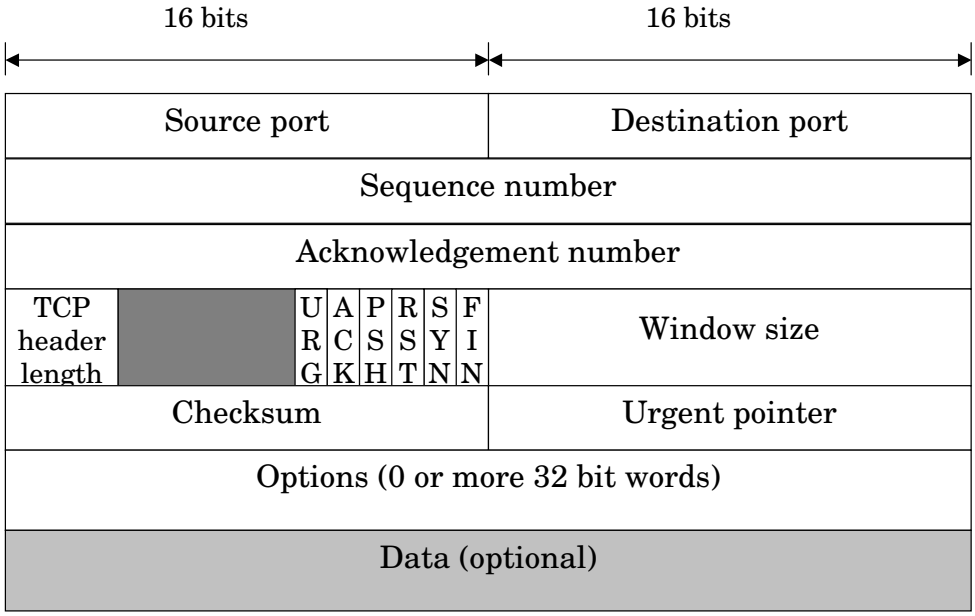
The preceding description of TCP congestion control roughly corresponds to the version of TCP called "Tahoe". Several other versions exist.

For example in TCP Reno, the algorithm is modified to support ***fast-retransmission***, where a packet can be retransmitted after three duplicate ACKs are received (before timing out). A fast retransmission is followed by a ***fast-recovery***, where TCP basically skips the slow start period.

Another version of TCP is called "Vegas." In Vegas the size of the congestion window is based on the history of RTT's observed by the sender. For example, when the change in the RTT becomes too large the sender decreases the congestion window. The idea here is to detect the onset of congestion before a time out occurs and achieve higher throughput.

TCP header

The header fields for TCP are shown below.



Header length is measured in 32 bit words. The URG,ACK, PSH, RST, SYN, and FIN fields are one bit flags. The ACK bit indicates that the Acknowledgement field is a valid acknowledgement. The RST bit is used to reset a connection. The SYN bit is used during connection establishment. A connection request packet has SYN=1, ACK =0, a connection accept packet has SYN=1, ACK=1, and the sequence number of the connection request packet is in the Acknowledgement field. The FIN bit is used to release connection. The PSH and URG fields are rarely used. As in UDP, the TCP checksum is calculated using the TCP header, data and part of the IP header.