Lecture-27: TCP Reno

1 Adaptive window flow control and TCP protocols

Congestion control algorithms deployed in the Internet use a concept called window flow control. Each user maintains a number, called a window size, which is the number of unacknowledged packets that are allowed to be sent into the network. Any new packet can be sent only when an acknowledgement for one of the previous sent packets is received by the sender, as shown in Figure 7.1. Suppose that the link speeds in the network are so large that the amount of time it takes to process a packet at a link is much smaller than the round-trip time (RTT), which is the amount of time that elapses between the time that a packet is released by the source and the time when it receives the acknowledgement (ack) for the packet from the destination. Then, roughly speaking, the source will send W packets into the network, receive acknowledgements for all of them after one RTT, then send W more packets into the network, and so on. Thus, the source sends W packets once every RTT, which means that the average transmission rate x of the source can approximated by the formula $x = \frac{W}{T}$, where T is the RTT.

A significant problem with the simple window flow control scheme as described above is that the optimal choice of W depends on the number of other users in the network: clearly, the transmission rate of a user should be small if it shares a common link with many other users, but it can be large (but smaller than the capacity of its route) if there are few other users sharing the links on its route. Thus, what is required is an algorithm to adjust adaptively the window size in response to congestion information. This task is performed by the congestion control part of the Transmission Control Protocol (TCP) in today's Internet. The general idea is as follows: a sender increases its window size if it determines that there is no congestion, and decreases the window size if it detects congestion. Packet losses or excessive delays in the route from the source to the destination are used as indicators of congestion. Excessive delays in receiving an ack indicate large delays, while missing ACKs signal lost packets. With some exceptions, congestion detection is performed by the source at time instants when it receives an ACK for a packet. This means that the congestion control algorithm makes decision at ACK reception events, and does not explicitly rely on any clocks for timing information. In other words, TCP is a self-clocking protocol. Different versions of TCP use different algorithms to detect congestion and to increase/decrease the window sizes. We discuss two versions of TCP in the following sections.

1.1 TCP-Reno: a loss-based algorithm

The most commonly used TCP flavors used for congestion control in the Internet today are Reno and NewReno. Both of them are updates of the TCP-Tahoe, which was introduced in 1988. Although they vary significantly in many regards, the basic approach to congestion control is similar. The idea is to use successful reception packets as an indication of available capacity and dropped packets as an indication of congestion.

We consider a simplified model for the purpose of exposition. Each time the destination receives a packet, it sends and acknowledgement (also called ACK) asking for the next packet in sequence. For example, when packet 1 is received, the acknowledgement takes the form of a request for packet 2. If, instead of the expected packet 2, the destination receives packet 3, the acknowledgement still requests packet 2. Reception of three duplicated acknowledgments or dupAcks (i.e., four successive identical ACKs) is taken as an indication that packet 2 has been lost due to congestion. The source then proceeds to cut down the window size and also to re-transmit lost packets. In case the source does not receive any acknowledgements for a finite time, it assumes that all its packets have been lost and times out. When a non-duplicate acknowledgment is received, the protocol increases its window size. The amount by which the window size is increases depends upon the TCP transmission phase. TCP operates in two distinct phases.

Slow start. When file transfer begins, the window size is 1, but the source rapidly increases its transmission window size so as to reach the available capacity quickly. Let us denote the window size W. The algorithm increases the window size by 1 each time an acknowledgement indicating success is received, i.e., $W \leftarrow W + 1$. This is called the slow-start phase. Since one would receive acknowledgements corresponding to one window's worth of packets in an `RTT`, and we increase the window size by one for each successful packet transmission, this also means that (if all transmissions are successful) the window would double in each `RTT`, so we have an exponential increase in rate as time proceeds. Slow-start refers to the fact that the window size is still small in this phase, but the rate at which the window is increases is quite rapid.

Congestion avoidance. When the window size either hits a threshold, called the slow-start threshold or `ssthresh` or the transmission suffers a loss (immediately leading to a halving of window size), the algorithm shifts to a more conservative approach called the congestion avoidance phase. When in the congestion-avoidance phase, the algorithm increases the window size by $\frac{1}{W}$ every time feedback of a successful packet transmission is received, so we now have $W \leftarrow W + \frac{1}{W}$.

Loss event. When a packet loss is detected by the receipt of three `dupAck`, the slow-start threshold `ssthresh` is set to W and TCP Reno cuts its window size by half, i.e., $W \leftarrow \frac{W}{2}$.

Thus, in each `RTT`, the window increases by one packet—a linear increase in rate. Protocols of this sort where increment is by a constant amount, but the decrement is by a multiplicative factor are called *additive-increase multiplicative-decrease* (AIMD) protocols. When packet loss is detected by a time-out, the window size is reset to 1 and TCP enters the slow-start phase. We illustrate the operation of TCP-Reno in terms of rate of transmission in Figure 4.2. The slow-start phase of a flow is relatively insignificant if the flow consists of a large number of packets. So we will consider only the congestion-avoidance phase.

1.2 TCP-Reno: time evolution of congestion window

Let us call the congestion window at time t as $W(t)$, which is the number of packets in-flight. The time taken by each of these packets to reach the destination, and for the corresponding acknowledgement to be received is T. The `RTT` is a combination of propagation delay and queueing delay. In our modeling, we assume that the `RTT` is constant, equal to the propagation delay plus the maximum queueing delay. If one observes the medium between the source and destination for an interval $[t, t+T]$, and there are no dropped packets, then the number of packets seen is $W(t)$ since the window size changes rather slowly in one `RTT`. Thus, the average rate of transmission $x(t)$ is just the window size divided by T, i.e., $x(t) = \frac{W(t)}{T}$. We assume that packet $n-1$ sent at time T_{n-1} is successfully received, and an `ACK` is received at time $T_{n-1} + T$. Packet n is sent at time T_n and if it is successfully received then we receive an `ACK` for it at time $T_n + T$, and ξ_n is the indicator that packet n is successfully received. If a duplicate `ACK` for packet $n-1$ is received at time $T_{n+1} + T$, then it indicates packet n was dropped. Further, we observe that $W(T_n)$ is the number of packets to be sent in the duration $[T_n, T_n + T)$, and therefore $T_{n+1} = T_n + \frac{1}{x(T_n)}$ To be precise, we have

$$\frac{W(T_{n+1}+T) - W(T_n+T)}{T_{n+1} - T_n} = \xi_{n+1} x(T_n) \frac{1}{W(T_n+T)} - \bar{\xi}_n x(T_n) \frac{W(T_n+T)}{2}.$$

We now write down what we have just seen about TCP Reno's behavior in terms of the differential equation models that we have become familiar with. Consider a flow r. As defined above, let $W_r(t)$ denote the window size and T_r its `RTT`. Earlier we had the concept of the price of a route r being $q_r(t)$. We now use the same notation to denote the probability that a packet will be lost at time t. Notice that the loss of packets is the price paid by flow r for using the links that constitute the route it uses. Replacing indicator ξ_n by its mean $q_r(T_n)$, we can model the congestion avoidance phase of TCP-Reno as

$$\dot{W}_r(t) = x_r(t - T_r)(1 - q_r(t - T_r)) \frac{1}{W_r(t)} - x_r(t - T_r) q_r(t - T_r) \beta W_r(t). \tag{1}$$

The above equation can be derived as follows.

Congestion avoidance. The rate at which the source obtains acknowledgements is $x_r(t-T_r)(1-q_r(t-T_r))$. Since each acknowledgement leads to an increase by $\frac{1}{W(t)}$, the rate at which the transmission rate increases is given by the first term on the right side.

Loss event. The rate at which packets are lost is $x_r(t-T_r)q_r(t-T_r)$. Such events would cause the rate of transmission to be decreased by a factor that we call β. This is the second term on the right side. Considering the fact that there is a halving of window size due to loss of packets, β would naturally taken to be $\frac{1}{2}$. However, studies show that a more precise value of β when making a continuous-time approximation of TCP's behavior is close to $\frac{2}{3}$.

To compare the TCP formulation above to the resource allocation framework, we write $W_r(t)$ in terms of $x_r(t)$ which yields

$$\dot{x}_r = \frac{x_r(t-T_r)(1-q_r(t-T_r))}{T_r^2 x_r} - \beta x_r(t-T_r)q_r(t-T_r)x_r(t). \tag{2}$$

The equilibrium value of x_r is found by setting $\dot{x}_r = 0$ and denoted by $\hat{x}_r$. Denoting the equilibrium loss probability as $\hat{q}$, we observe that

$$\hat{x}_r = \frac{1}{T_r}\sqrt{\frac{1-\hat{q}_r}{\beta\hat{q}_r}}.$$

For small values of $\hat{q}_r$ (which is what one desires in the Internet), $\hat{x}_r \propto \frac{1}{T_r\sqrt{\hat{q}_r}}$. This result is well-known and widely used in the performance analysis of TCP Reno.

1.3 Relationship with primal congestion control algorithm

Now, consider the controller (2) again. Suppose that there were no feedback delay, but the equation is otherwise unchanged. So T_r^2 that appears in (2) is just some constant now. Also, let $q_r(t)$ be small, i.e., the probability of losing a packet is not too large. Then the controller reduces to

$$\dot{x}_r = \frac{1}{T_r^2} - \beta x_r^2 q_r = \beta x_r^2\Big(\frac{1}{\beta T_r^2 x_r^2} - q_r\Big).$$

Comparing with primal congestion controller $\dot{x}_r = k_r(x_r)(U_r'(x_r) - q_r)$, we observe that the the controller (2) is a primal congestion controller for scaling function $k_r(x_r) = \beta x_r^2$ and the utility function $U_r : \mathbb{R}_+ \to \mathbb{R}$ of the source r that satisfies

$$U_r'(x_r) = \frac{1}{\beta T_r^2 x_r^2}.$$

We can find the source utility up to an additive constant by integrating the above, which yields $U_r(x_r) = -\frac{1}{\beta T_r^2 x_r}$. Thus, TCP can be approximately viewed as a control algorithm that attempts to achieve weighted minimum potential delay fairness.

If we do not assume that q_r is small, the delay-free differential equation is given by

$$\dot{x}_r = \frac{1-q_r}{T_r^2} - \beta x_r^2 q_r = (\beta x_r^2 + \frac{1}{T_r^2})\Big(\frac{1}{\beta T_r^2 x_r^2 + 1} - q_r\Big).$$

This is also a primal congestion controller for scaling function $k_r(x_r) = (\beta x_r^2 + \frac{1}{T_r^2})$ and the following derivative of the utility function and the utility function up to an additive constant,

$$U_r'(x_r) = \frac{1}{\beta T_r^2 x_r^2 + 1}, \qquad U_r(x_r) = -\frac{1}{T_r\sqrt{\beta}}\tan^{-1} x_r T_r\sqrt{\beta}.$$

1.4 A generalization of TCP-Reno

Instead of increasing the window size by $\frac{1}{W}$ for each ack and decreasing the window by $\frac{1}{2}$ upon detecting a loss, one could consider other increase-decrease choices as well. Consider a protocol where $W \leftarrow W + aW^n$ when an acknowledgement is received, while a loss triggers a window decrease given by $W \leftarrow W - bW^m$.

Setting $a = 1, n = -1, b = 0.5$, and $m = 1$ would yield TCP-Reno type behavior. The equivalent rate-based equation describing the dynamics of such a protocol would be

$$\dot{x}_r = \frac{x_r(t-T_r)(1-q_r(t-T_r))}{T_r} a(x_r(t)T_r)^n - \frac{x_r(t-T_r)q_r(t-T_r)}{T_r} b(x_r(t)T_r)^m.$$

Ignoring the feedback delay in obtaining the congestion information, the above differential equation becomes

$$\dot{x}_r = a x_r^{n+1} T_r^{n-1} - (a x_r^{n+1} T_r^{n-1} + b x_r^{m+1} T_r^{m-1}) q_r = a x_r^{n+1} T_r^{n-1} (1 + \frac{b}{a}(x_r T_r)^{m-n}) \Big(\frac{1}{1 + \frac{b}{a}(x_r T_r)^{m-n}} - q_r \Big).$$

Note that $\frac{1}{1+\frac{b}{a}(x_r T_r)^{m-n}}$ is a decreasing function for $m > n$. Thus, its derivative will be negative and hence one can view the above differential equation as the primal congestion controller for a source r with scaling function k_r and a concave utility function U_r given by

$$k_r(x_r) \triangleq a x_r^{n+1} T_r^{n-1} (1 + \frac{b}{a}(x_r T_r)^{m-n}), \qquad U_r(x_r) \triangleq \int_0^{x_r} \frac{1}{1 + \frac{b}{a}(x T_r)^{m-n}} dx.$$

A special case of the above congestion control algorithm called Scalable-TCP which uses the parameters $a = 0.01, n = 0, b = 0.125$,and $m = 1$ has been proposed for high-bandwidth environments.