Lecture 6: Congestion collapse

Anirudh Sivaraman

2018/10/16

Last lecture, we discussed the sliding window protocol, and showed that its throughput is $\frac{W}{RTT}$, where W is the window size and RTT is the minimum round-trip time (i.e., the RTT when there are no queueing delays). We also showed that the "correct" value of the window W is $C * RTT$, where C is the bottleneck link's capacity along the path from the sender to the receiver. A window that is larger than $C * RTT$ (i.e., the bandwidth-delay product or BDP) will only contribute to queueing delay, without increasing the throughput.

1 Why is estimating the BDP hard?

Obviously, C and RTT can vary depending on the network conditions at the sender and the receiver and the path between them. They can also vary within the lifetime of a data transfer between the sender and the receiver. Henceforth, we will call a data transfer between a sender and a receiver a flow.[1]

Here are a few reasons why C and RTT can vary within the lifetime of a flow. If either the sender or the receiver is on a wireless link, C can vary depending on how far the user is from the WiFi AP or cellular base station. Similarly, RTT depends on whether the sender or a receiver is on a WiFi, Ethernet, or 4G link because each link technology has different propagation and transmission delays. Finally, RTT can change because of changes in the network path between the sender and the receiver (e.g., a router was taken down for maintenance). In summary, neither C nor RTT is known up front, and setting the sliding window protocol's window size to the BDP is impractical.

At the same time, setting the window to an incorrect value can cause serious performance problems. First, if the window size is larger than the BDP, the queueing delay, and hence latency, increases. For a flow that is interested in the latency of individual packets (e.g., the terminal application SSH running over TCP), this latency can hurt interactivity. Second, when queueing delay increases, the risk of a spurious and unnecessary retransmission timeout also increases. But the packet being retransmitted hasn't been lost yet; it's simply stuck in a long queue. The net result is a duplicate copy of the packet. If this happens enough, the effective transport-layer throughput can fall drastically because multiple packet copies are delivered to the receiver, which wastes link capacity.

So far we have focused on the problems with not setting the window size correctly for a single flow, assuming that there's only a single flow on the network at any time. This is obviously a bit unrealistic. Let's now look at the scenario of multiple flows sharing the same bottleneck link. The problems only get worse here. In this scenario, C for each flow is no longer the link's capacity, but some fraction of it so that the link is shared fairly among its flows. Typically, this fraction is $\frac{C}{n}$, where n is the number of flows.[2] The RTT for each flow could be different, but for simplicity, we'll assume the same RTT across all flows. Finally, when there are multiple flows with a constant window size for each flow, the effective window size seen by the network is the sum of the window sizes for each flow.

In this multi-flow scenario, especially when flows start and stop often, it is hard for any flow to know n precisely. This makes it more likely that each flow is using the wrong value of W. And as we have seen earlier, using a W that's too large can lead to queueing delay or retransmissions. Using a W that's too small can lead

[1] Networking literature uses the term flow because the stream of packets along a network path is similar to a fluid flowing from one point to another over a pipe of a certain length (RTT) and a certain minimum width (C).

[2] This is not the only way to share a link across competing flows.

to underutilization. We'll focus on the first case of wrong and large W in this lecture because of two reasons. One, it reflects human behavior: it is in a sender's selfish interest to set a larger W to get more throughput if all other senders are doing the same. Two, no matter how small an individual sender's W is, with enough senders sharing a network, the effective W will be too large.

2 Congestion collapse

Using a W that's too large (regardless of whether it's because of a single sender picking a large W or whether it's because there are too many senders) leads to a phenomenon called *congestion collapse*. Congestion collapse is a term for a situation in which the offered load, i.e., the demand for the network's services, is increasing, but the overall utility of the network to its users is decreasing. We'll make the terms offered load and utility precise in this lecture, and we'll also give a few simple examples of congestion collapse.

But first, it is useful to understand congestion collapse by way of analogy. Let's say there are 2 pairs of people in a crowded conference room conversing with each other at the same time. We can increase the number of pairs from 2 to say 5, 10, 20, 25, and still ensure that every person in the room can hear their partner in their pair. But, at some point, the overall noise level in the network will rise to a point where fewer and fewer people can hear their partner clearly. At this point, the overall utility of the network is decreasing (how many pairs can successfully communicate), while the offered load on the network is increasing (how many pairs wish to communicate).

If you have taken an economics class, congestion collapse is the networking equivalent of "The Tragedy of the Commons," where multiple stakeholders overexploiting a shared resource make the resource unusuable for everyone. Congestion collapse is also similar to the problem of thrashing in Operating Systems, where a process' working set cannot fit into main memory. Here the OS is still performing some work: swapping pages into and out of memory. But most of the CPU's cycles are not running user code; hence, the amount of useful work is quite low.

3 A simple example of congestion collapse

We'll illustrate congestion collapse with a very simple example and then move on to slightly more involved examples of congestion collapse. We'll assume several things for this example.

1. We'll assume there are n flows, each of which runs a sliding window protocol. This n is the offered load or the demand presented to the network.
2. Each flow has the same window size W. Hence, the effective window is nW.
3. Each flow has the same minimum RTT RTT.
4. All flows share the same bottleneck link.
5. The bottleneck link's capacity is C.
6. The link has infinite buffers. In other words, the link never drops a packet and queueing delays can go up indefinitely.
7. Because packets are never dropped in this system, we'll turn off the retransmission logic in the sliding window protocol for this example alone.
8. We'll assume each flow has a *utility function* that quantitatively captures how the flow values high throughput and low latency. For simplicity, we'll assume the following utility function for all flows in this example: $\frac{throughput}{latency}$, where $throughput$ and $latency$ are the average throughput and latency in steady state for each flow. We'll also assume we can add up these per-flow utilities to determine the aggregate utility of the network.[3]

[3] Adding utilities this way is problematic in general because it is akin to adding two people's happiness, and it's hard to compare the happiness of two different people. But we'll add utilities in today's lecture for ease of illustration.

Now, let's see what happens as we increase n. As long as $n \leq \frac{C.RTT}{W}$, the effective window size seen by the network is under the BDP, and hence there is no queueing delay. Hence, until this point, the utility of each flow is $\frac{\frac{W}{RTT}}{RTT}$, because each flow's average throughput is $\frac{W}{RTT}$ and each flow's average latency is RTT, because there is no queueing delay. Hence, the aggregate utility is

$$\begin{aligned} & n.\frac{\frac{W}{RTT}}{RTT} \\ =& \frac{n.W}{RTT^2} \end{aligned}$$

Now, what happens if $n > \frac{C.RTT}{W}$. At this point, the throughput of each flow is $\frac{C}{n}$, by symmetry because (1) each flow is carrying out the exact same protocol under the same conditions and (2) the total link capacity is C. The queueing delay is non-zero and is given by $\frac{W_{total}}{C} - RTT$. Here W_{total} is the effective window size presented by all senders to the network and is the sum of the individual windows W, i.e., $W_{total} = nW$. The rationale for this expression for queueing delay is because *each flow* experiences the queueing delay caused by packets from *all flows* in the network with first-in first-out queues. Hence, each individual flow's utility is

$$\begin{aligned} & \frac{\frac{C}{n}}{RTT + \frac{W_{total}}{C} - RTT} \\ =& \frac{\frac{C}{n}}{\frac{W_{total}}{C}} \\ =& \frac{\frac{C}{n}}{\frac{nW}{C}} \end{aligned}$$

Hence, the aggregate utility is

$$\begin{aligned} & n.\frac{\frac{C}{n}}{\frac{nW}{C}} \\ =& \frac{C^2}{nW} \end{aligned}$$

If we plot the aggregate utility as a function of n, we see a graph like the one in Figure 1. The distinguishing feature of congestion collapse is how the utility goes up until a point and then goes down or "collapses" sharply when the offered load is increased further. In the regime in which there is congestion collapse, the network is still doing work. For instance, in the example above, the bottleneck link is still 100% utilized and delivering packets at every instant. But, not all of this work is *useful*. Specifically, in this example, there is diminishing user utility to delivering packets late because the utility is inversely proportional to the average packet latency.

4 A different utility function that is more sensitive to latency

We can modify the utility to be even more sensitive to the packet latency. For instance, we could declare that if the average packet latency is greater than a certain threshold th, the utility to the user is zero, regardless of what the throughput is. Such a utility function captures the preference of browser users, who will occasionally give up on a web search if it takes longer than a few milliseconds. In such cases, there is an even sharper form of congestion collapse (Figure 2), where the utility falls to zero once the number of senders n reaches a number n', such that $\frac{n'W}{C} > th$, i.e., the average latency exceeds th.

5 An example of congestion collapse with retransmissions

We'll demonstrate another example of congestion collapse here. In this example, the congestion collapse will occur because packets are retransmitted multiple times and multiple packet copies are received at the receiver,

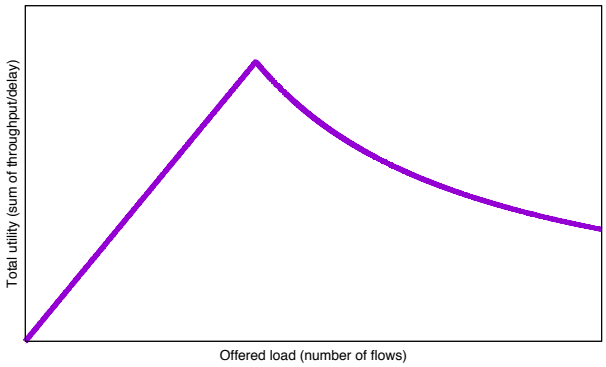

Figure 1: Simple example of congestion collapse where each flow's utility is $\frac{throughput}{latency}$ and links have infinite buffers.

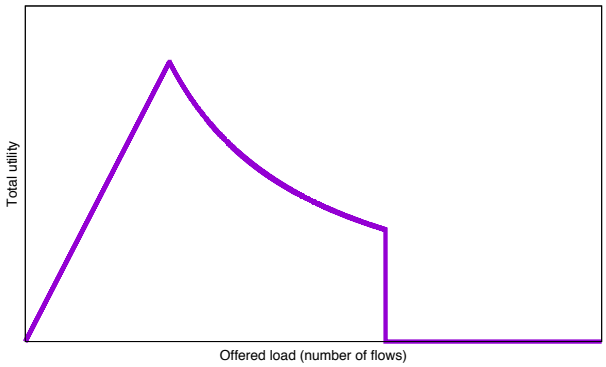

Figure 2: The same example as Figure 1 with the additional requirement that the utility is zero if the average latency exceeds a certain threshold latency.

wasting scarce link capacity in the process. To setup this example, we'll assume the same conditions as before, with a few small, but important, changes.

1. Instead of removing all retransmission logic from the sliding window protocol, we'll assume that packets are retransmitted after a *fixed timeout* of T, where $T > RTT$.

2. We'll now change the utility function to reflect the flow's transport-layer throughput alone, not latency. But because packets can be retransmitted by the sender, it is possible that multiple copies of the same packet are received by the receiver. In such cases, we'll count these copies only once. In other words, we are interested in the number of unique packets that are received in order per second (i.e., the transport-layer throughput), reflecting the abstraction that TCP provides us.

Again, we'll look at what happens as we increase the offered load: the number of flows n. Until $n < \frac{C.RTT}{W}$, there is no queueing delay, and hence there are no timeouts or retransmissions. The aggregate transport-layer throughput (and hence aggregate utility) in this regime increases linearly with n as with the previous two examples.

In the next regime, where $RTT < \frac{nW}{C} < T$, the aggregate transport-layer throughput remains fixed at C, because it cannot increase any further. Importantly, packets are not being retransmitted yet, because the packet latency, $\frac{nW}{C}$, is still under the timeout T, so every packet that is delivered by the link is still useful because packets have not been duplicated yet.

In the third regime, where $\frac{nw}{C} > T$, timeouts start to occur. This is because the average latency is greater than T, which is the timeout interval. Furthermore, none of these timeouts is useful because the packets for

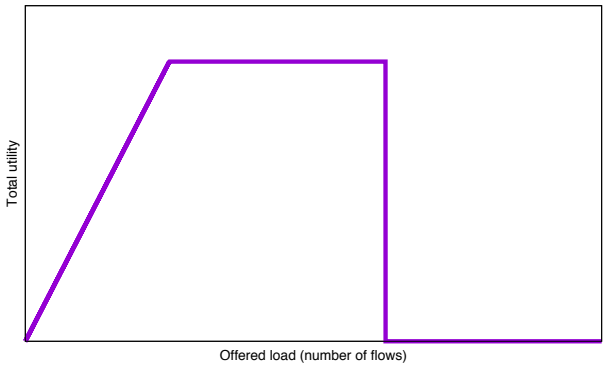

Figure 3: Congestion collapse when utility depends on the number of unique packets that are delivered in order, and there are retransmissions in the network.

which timeouts occurred are still in the queue and will eventually be delivered (recall the link does not drop any packets). At the same time, because the link's buffers don't drop any packets the retransmitted packets will be delivered as well. As a result, the packets delivered by the link are a mix of the original packets and their duplicate retransmissions, implying that the aggregate transport-layer throughput will be strictly less than C.

How much less? The degradation in aggregate transport-layer throughput depends on how large the queueing latency gets, and hence how many retransmissions occur. This, in turn, depends on the offered load n. As n gets larger, the aggregate transport-layer throughput drops. Unlike the previous two examples, it's harder to do a pen-and-paper calculation of throughput for this example because it requires us to keep track of when a packet was first transmitted, when it needs to be retransmitted, and so on, which can quickly get overwhelming.

In such cases, one way to quantify the throughput is to run a *simulation*, where we imitate the operations of the link, the sender, and the receiver over time using a computer program known as a *simulator*. We then measure the transport-layer throughput in this make-believe simulator world. For this particular example, a simulation shows that the steady-state aggregate transport-layer throughput falls to zero once $\frac{nw}{C}$ exceeds T (Figure 3). Informally, introducing retransmissions into the link's queue causes the queueing delay to grow further, which leads to even more retransmissions, which leads to even more queues, and so on. In other words, there is a positive feedback loop, which soon leads to most packets being retransmissions to the point where the useful throughput is almost zero. You'll run a few simple simulations as part of your assignment that should help you explore such congestion collapse scenarios on your own.

6 Historical notes

Congestion collapse was first documented on the Internet in 1984 by John Nagle [1] when internetworking Ford Aerospace's private network with the ARPANET. These were two networks within the Internet in the early 1980s. The same problem was also observed by Van Jacobson in 1986 in data transfers between Lawrence Berkeley Laboratory and the University of California at Berkeley [3]. These are both examples of retransmission-based congestion collapse (§5), where the sender was transmitting multiple copies of the same packet that were eventually all delivered to the destination, wasting scarce link capacity in the process.

A more recent form of congestion collapse is a phenomenon called bufferbloat [2], first reported in 2009, where some router buffers (especially on WiFi and cellular networks) had gotten so large that many of them seldom dropped packets, leading to a large increase in latency. This is similar to the congestion collapse example we discussed in §3.

References

[1] Congestion Control in IP/TCP Internetworks. `https://tools.ietf.org/html/rfc896`, 1984.

[2] Vint Cerf, Van Jacobson, Nick Weaver, and Jim Gettys. BufferBloat: What's Wrong with the Internet? *ACM Queue*, 2011.

[3] V. Jacobson. Congestion Avoidance and Control. SIGCOMM, 1988.