Lecture 5: The sliding window protocol

Anirudh Sivaraman

2018/10/16

We'll continue with reliability in this lecture. Last lecture, we introduced the Stop-And-Wait protocol for providing reliability. The Stop-And-Wait protocol is very simple to understand, but its throughput isn't great. It roughly transmits one packet every round-trip time (RTT) between the client and the server. Hence its network utilization (the ratio of throughput to the capacity of the underlying network) is $\frac{\frac{PKT\_SIZE}{RTT}}{C}$, where C is the network's capacity, PKT_SIZE is the size of a single packet, and RTT is the round-trip time for the path from the sender to the receiver. When we use the variable RTT, we mean the minimum round-trip time *excluding* queueing delays inflicted by transmissions between the sender and receiver and *excluding* any packet transmission delays.[1]

Let's plug in some rough numbers for these three parameters to understand how bad the network utilization can be. PKT_SIZE is 1448 bytes on the Ethernet link from my office desktop. Let's pick an RTT of 80 ms to reflect a coast-to-coast US path.[2] The value of C for the path between NYC and a server in SFO is about 370 Mbit/s on my wired network when measured by speedtest.net,[3] The network utilization with these numbers is about 0.04%, which is terrible. In general, the network utilization for the Stop-And-Wait protocol is the least on networks with large RTTs and large link capacity, i.e., long and fat networks (LFN, often pronounced elephen).

Clearly, we would like to do better. In today's lecture, we'll discuss a protocol called the sliding window protocol with significantly better throughput.

1 The sliding window protocol

The sliding window protocol allows the sender to transmit multiple packets while waiting for the acknowledgement for the first packet. The goal here is to keep the link occupied with transmitting packets while the sender waits on acknowledgements (ACKs) for the first packet.

We can summarize the sliding window protocol in 2 sentences. The sender sends out a set of W packets (called a window of packets) at the beginning, i.e., packets with sequence numbers 1 through W. It then releases a new packet in response to an ACK. This idea of releasing a new packet every time a packet is acked is called the packet conservation principle.[4] Here, the quantity being conserved (being held constant) is the window W of unacknowledged, outstanding packets that are still in flight in the link between the sender and the receiver.

The window here refers to the number of unacknowledged packets that are currently traveling on the link between the sender and the receiver. Such packets are said to be "in flight." The sliding window refers to the fact that the set of unacknowledged packets is continuously moving. In the beginning, it is the set of packets

[1] The RTTs reported by ping *include* both queueing and transmission delays. Queueing delays can be removed by looking at the minimum RTT across a set of RTT samples. Transmission delays can be calculated and removed from the RTT by knowing the packet size and dividing it by the link capacity, e.g., on a 10 Mbit/s, a 1500 byte packet incurs a transmission delay of 1.2 ms.

[2] This is the ping latency from my desktop to servers that are physically located in Berkeley and Seattle. Picking such servers is surprisingly hard! Many domain names are geographically replicated and correspond to many physical server locations. For instance, www.google.com maps to different physical servers depending on the client's location.

[3] This is not the most accurate way to measure C. In particular, we are measuring C to a different server and RTT to a different server. But, it'll do for a back-of-the-envelope calculation.

[4] The packet conservation principle was first articulated by Van Jacobson in his paper "Congestion Avoidance and Control,". We'll discuss the core ideas in Van Jacobson's paper two lectures from now, but the packet conservation principle will be useful for this lecture.

with sequence numbers 1, 2, ..., W, where W is the size of the window. Now, as soon as packet 1[5] is acked by the receiver, the sender sends out packet $W+1$; the window now "slides" and consists of packets 2, 3, ..., $W+1$.

We'll now describe the sender and receiver sides of the sliding window protocol more precisely.

1.1 The sliding window sender

The sender keeps track of the number and set of unacknowledged packets at any instant and ensures that this number never exceeds W. At the beginning of time, the sender sends out the first W packets back-to-back at whatever is the capacity of its outgoing link. It then waits for an ACK for any of these packets. This ACK gives it the permission to slide its window by sending out a new packet with a sequence number greater than anything sent out so far.

When there are no losses, the workings of this protocol are easy to see. At time 0, the window consists of packets 1, 2, ..., W. After the first ACK an RTT later, the window slides and consists of packets 2, 3, ..., W+1. After the second, the window slides to 3, 4, ..., W+2, and so on. What if a packet is lost? Let's say at some instant in time, the sender has received acks for packets 1, 2, ..., W, except for packet k ($k \leq W$). Let's assume packet k or its ACK was dropped. Then, the window slides and now consists of packets k, $W+1$, $W+2$, $W+3$, ... $W+W-1$. Thus, lost packets can cause the window to be *fragmented*. For instance, when packet k is lost, the window has 2 fragments of packets with continuous sequence numbers: one with just k and the other with packets $W+1$ through $2W-1$.

1.2 Retransmissions

How does the sender handle a packet (or its ACK) being lost? This part is identical to the Stop-And-Wait protocol. This is another example of modularity where the strategy for handling retransmissions is decoupled from the strategy for sending new packets on the link (also called flow control). Every time a packet is transmitted, the sender sets a timer, which expires after timeout seconds. This timeout is computed using the mean and variance of the RTT samples. It is picked conservatively to ensure that the probability that an RTT sample exceeding this timeout is low. The mean and variance are obtained by running an exponentially weighted moving average on top of the RTT samples as discussed in the previous lecture.

1.3 The sliding window receiver

The sliding window receiver is similar to the Stop-And-Wait receiver in that it acks every packet using the sequence number in that packet. The difference in the sliding window protocol is that the receiver now needs to maintain a buffer for all packets that have been received, but are currently not in order. This change is required to deal with window fragmentation that we discussed above. For instance, let's say that at a particular instant, the receiver has received packets 1, 3, 4, 5, 6, 7, 8, 9, 10, and 11. Then it needs to buffer all these packets. This is because, until packet 2 has arrived, the receiver cannot deliver these packets up to the application because they are not yet in order.

This requires the receiver to keep track of the next packet that is expected in order to maintain the in-order delivery of data. This is the variable `next_in_order` from the previous lecture. It is one more than the sequence number of the latest packet delivered to the application until this instant. In the example in the previous paragraph, `next_in_order` is 2. Then, whenever a new packet arrives, the receiver checks if the most recently received packet allows the receiver to advance `next_in_order`. In the example above, `next_in_order` advances to 12 as soon as 2 arrives.

The evolution of `next_in_order` is different between the Stop-And-Wait and sliding window protocols. In the Stop-And-Wait protocol, `next_in_order` can only increase by at most 1 on a packet reception. In the sliding window protocol, `next_in_order` can advance arbitrarily depending on how many packets have been received out-of-order. This is what contributes to the higher throughput for the sliding window protocol.

[5]We'll use the shorthand packet N to refer to the packet with sequence number N.

2 Throughput of the sliding window protocol

If we assume losses are rare,[6] we can calculate the throughput of the sliding window protocol as follows. At time 0, the sender sends out W packets back-to-back at the capacity of its outgoing link. We'll assume the sender is directly connected to the receiver using its outgoing link. We'll also assume that the outgoing link has a capacity C, measured in packets per second.

In a real Internet path, the sender and receiver are not directly connected by a link. Instead, there are multiple links between the sender and the receiver, where each link connects the sender to a router, a router to a router, or a router to the receiver. We can still abstract this real Internet path using a single link with capacity C—provided we choose C to be the capacity of the bottleneck (slowest) link between the sender and the receiver.[7]

Back to the single link setting, let's say W packets are sent out at time 0. Because the link can carry only C packets per second, the last bit of each of these packets departs the sender end of the link at times $\frac{1}{C}$, $\frac{2}{C}$, $\frac{3}{C}$, $\ldots$, $\frac{W}{C}$. This is because the transmission delay of a single packet is $\frac{1}{C}$ and each packet has to wait for the transmission delay of the previous one before it can get onto the link.

As a rough analogy, you can imagine a square wave propagating from the sender to the receiver. A bit value equal to 1 is represented by a high voltage on the square wave and a bit value of 0 is represented by a low voltage on the square wave. Each bit—and hence each packet—takes up a minimum amount of time within the square wave so that it can be successfully detected at the receiver. We'll discuss the propagation of bits on a wire in more detail when we get to the physical layer.

These last bits reach the receiver at times $\frac{RTT}{2}+\frac{1}{C}$, $\frac{RTT}{2}+\frac{2}{C}$, $\frac{RTT}{2}+\frac{3}{C}$, $\ldots$, $\frac{RTT}{2}+\frac{W}{C}$. The ACKs for these packets are received at the sender at times $RTT+\frac{1}{C}$, $RTT+\frac{2}{C}$, $RTT+\frac{3}{C}$, $\ldots$, $RTT+\frac{W}{C}$. We'll assume the ACKs are small enough that their transmission delays can be ignored relative to the transmission delays of the packets.

Hence, at the end of $RTT+\frac{W}{C}$, W packets have been ACKed at the sender. Now, if we simulate the working of the sliding window protocol on the ACKs received at times $RTT+\frac{1}{C}$, $RTT+\frac{2}{C}$, $RTT+\frac{3}{C}$, $\ldots$, $RTT+\frac{W}{C}$, it is easy to see that at the end of $RTT+\frac{W}{C}+\frac{1}{C}+RTT$, $2W$ packets would have been acked. This is because at time $RTT+\frac{W}{C}$, the ACK for packet W is received, causing packet $2W$ to be sent. Packet $2W$ is received at the receiver $\frac{RTT}{2}+\frac{1}{C}$ time units later, and its ACK is received at the sender a further $\frac{RTT}{2}$ time units later.

Similarly, after $RTT+\frac{W}{C}+\frac{1}{C}+RTT+\frac{1}{C}+RTT$, $3W$ packets would have been acked. In general, after k such rounds, the time elapsed is $k.RTT+\frac{k-1}{C}+\frac{W}{C}$, kW packets would have been acked, giving us a throughput of $\frac{kW}{k.RTT+\frac{W+k-1}{C}}$. If we take the limit as k gets large, this reduces to $\frac{W}{RTT+\frac{1}{C}}$. Further, let's assume $1 << C * RTT$, because otherwise, we may as well use the Stop-And-Wait protocol instead. Under this condition, the throughput is $\frac{W}{RTT}$.

The analysis above is valid until packets from different rounds can fit on the wire without overlapping with each other. The number of packets that can "fit" within the wire without the rounds overlapping is $\frac{RTT}{\frac{1}{C}}$, because each packet takes up $\frac{1}{C}$ time on the wire and each round is about an RTT, ignoring the smaller $\frac{1}{C}$ term. This number is $C.RTT$, also called the bandwidth delay product.[8] So once W exceeds $C * RTT$, the transmissions from different rounds above start overlapping.

Once W reaches $C * RTT$, the link is continuously occupied with the transmission of packets because there is no gap between the rounds. The throughput is now C, the link capacity. Increasing W further does not increase the throughput any further because we are at the link's capacity. But where are the W packets located in time when the packets from different rounds start overlapping? Clearly there is no longer any time left to

[6] Most Internet paths have a loss rate of at most 1% as long as the traffic is well behaved, i.e., you don't blast packets out of your network.

[7] This still assumes that there are no other sender-receiver pairs sharing the same path as our sender and receiver. When there are other pairs, the C is a function of how many such pairs wish to share the network. This complicates things considerably, but we'll not get into this here. There is an entire subfield of networking research, called available bandwidth estimation, that essentially estimates C in the presence of other sender-receiver pairs.

[8] In many contexts, the term bandwidth is used to refer to link capacity. We'll use the term capacity instead because bandwidth is strictly speaking an analog quantity that captures the range of frequencies (in Hertz) of an analog signal.

"fit" these packets anywhere on the wire.

Instead, the packets build up in a queue causing queueing delay. What is the extent of the queueing delay? Any packets beyond the number that can be accommodated on the wire ($C.RTT$) must be sitting around in the sender's queue awaiting transmission. The total number of packets that have left the sender, but are as yet unacknowledged, is W. This means $W - C.RTT$ are sitting around in queues. Each packet's transmission delay is $\frac{1}{C}$, i.e., it takes up this much time on the wire. This means that the queueing delay is $(W - C.RTT).\frac{1}{C}$, which is $\frac{W}{C} - RTT$. Thus, a packet's queueing delay is just the product of the link's transmission delay and the number of packets ahead of that packet in the queue.

2.1 The right value of W

So, the goal here is to set W to $C.RTT$, but not much more. Because once W exceeds this point, the throughput does not increase, but the queueing delay goes up, which can frustrate interactive applications running over TCP (e.g., Secure Shell (SSH)). In the next two lectures, we'll see what happens if W is not set to the right value of $C.RTT$ (congestion collapse) and how end hosts automatically determine this value in a decentralized manner (congestion control).