# Homework 3: TCP

*Due: 11:59 PM, Dec 05, 2017*

## Contents

## Question 1

BBR gives us an interesting view of the relationship between window size, throughput, and round-trip time. Consider slide 19 of lecture 15 ('Another View of Congestion Control'):

1. As the window size increases past the BDP, why does the throughput stop increasing?

The thoughput stops increasing because it is limited by the bottleneck link. There is no way to send any faster than the BDP of that link from that node.

2. Assume you set the window to a fixed size, larger than the BDP, and smaller than the 'BDP+Bottleneck Queue' point in the graph. Explain why the queue at the bottleneck router is fixed at this point, instead of being ever growing.

The queue at the bottleneck router is fixed because after the first constant window size is sent, new packets are sent only as ACKs arrive (sliding window). The consant send rate is equal to the constant ACK receive rate, hence the queue does not grow infinitely.

3. Explain why you can't measure the minimum RTT (RTT$_{prop}$) and the bottleneck bandwidth at the same time. How does BRR get around that?

Minimum RTT and bottleneck bandwidth cannot happen at the same time. Mininum RTT happens only when there is minimum queue delay on the intermediate routers, hence when the throughput is not high enough to fill the bottleneck bandwidth. Bottleneck bandwidth happens only when all the intermediate router queues are full, hence when the RTT is not minimum. BBR is possible to measure both by measuring throughput when RTT starts to increase (hence when throughput increases from BDP to bottleneck), and measuring RTT when throughput is increasing (hence when throughput increases from 0 to BDP).

## Question 2

TCP Reno is not fair when two flows sharing the bottleneck link have very different round trip times. In the original Raj Jain diagram we used in class (e.g., slide 5 in lecture 15), we assumed the nodes were making decisions synchronously (i.e., with the same RTT). If A has double the RTT of B, draw a similar diagram, indicating what the final rates of A and B will be.

Flow rate of A and B are still bound by the capacity line $A + B = C$, while TCP Reno allows AIMD by increasing window size by 1 each RTT, where A has double the RTT of B. Hence, A and B will converge to the intersection of line $B = 2A$ and line $A + B = C$.

## Question 3

When requesting several objects to display a web page from the same server, HTTP gives the client and server many options, which you will compare here.

1. Assume that packets are 1500B, and let's ignore header sizes here. The bottleneck bandwidth of the path to the server is 120 Mbps = (10,000 packets/s) and the RTT is 0.1s. Assume that the initial TCP window is 1 packet. Considering slow start, **what is the smallest object the server has to transfer before its congestion window leaves slow start?** (Assume that the content can have sizes that are multiples of the packet size. Assume you leave slow start as soon as the size of your window is larger than the BDP).

   BDP is $10000 \times 0.1 = 1000$ packets. Considering slow start, starting from 1, we need the window to reach at least 1000 packets. When that happens, we will have sent 1+2+4+...+1024 packets = 2047 packets = $3,070,500$B.

2. One option for HTTP is to do several requests on the same connection, with `keep-alive`. What are two advantages of this keep-alive over the old one-connection-per-request in HTTP 1.0?

   keep-alive doesn't have to initialize the TCP 3-way handshake. It also doesn't have to go through the slow start phase by having previous knowledge of RTT and BDP. It also requires less CPU and memory load from deleting and recreating TCP sockets.

3. What extra advantage does pipelining give? Is it more advantageous (over just keep-alive) when the objects are larger or smaller?

   Advantage of pipelining over keep-alive is that it saves many round trips. It is analogous to the difference between "stop and wait" and "sliding window." Pipelining is more advantageous for smaller objects, as the overhead of extra round trips is a lot larger for smaller objects.

4. Another option clients have is to have several parallel connections. If your metric for success is the total time to download all objects, give one situation in which parallel connections are better than pipelining, and one in which it is worse than pipelining.

Parallel connections are better than pipelining when there are other competing flows, as it gets a larger share of the bandwidth. It might also be better if there are losses, because only one object gets affected instead of all. Pipelined requests are better than parallel connections if you are the only one of the bottleneck link, as it would have to ramp up the window one time, rather than with every request. It might also be better in latency and memory for the server if the server gets overloaded with too many parallel connections.

## Question 4

TTL in DNS is quite different from TTL in IP.

1. Give one advantage for a server to set a very high TTL in a DNS entry.

The server can restrict the rate of requests by setting high TTL, as the caches of the entry would be valid longer.

2. Give one situation in which a server would want to set a very low TTL in a DNS entry.

A low TTL would give the server the flexibility to dynamically control the information given to requesters, such as load balancing. It also gives less latency for the clients to receive the correct response when an entry is changed.

3. During an attack last year, Dyn, a company that was serving as the primary DNS provider for a number of other companies, had its authoritative DNS servers go down. After the TTL for the cached DNS responses for their clients expired (Twitter, for example, was affected), people couldn't access them. One DNS resolver provider, OpenDNS, however, decided to disrespect the protocol and was returning entries for the affected companies even after the TTL expired. (They were doing this in cases in which the authoritative server for an expired entry could not be reached). While this let people access the affected sites, cite one potential disadvantage of this approach.

By disrespecting the protocol of TTL, the cached entry becomes untrustable data. Even if the stale entry is used only when the authoritative server doesn't respond, it can result to the authority already changing to a new entry while a middleman can use the old entry in their favor, e.g. obtaining the old IP address to direct to a different server.

4. CDNs use DNS responses to return good servers for you to find content with low latency. Explain why using a DNS resolver out on the Internet can result in poor choice of CDN servers.

By using a public DNS server that is not geographically close to the client, since CDN can choose a server closest to the DNS server, it doesn't necessarily guarantee the closest CDN server from the client.

5. IP Anycast is one way to alleviate this problem. Explain how IP anycast is implemented through BGP. Explain how the use of anycast by such a DNS provider (e.g., Google's 8.8.8.8) can alleviate the problem in the previous question.

IP anycast is implemented through BGP by assigning the same IP address to different physical machines around the world. By doing so, the routing entries naturally select a path to the IP address being the closest server that has such IP address. By using a DNS provider with anycast, the DNS resolver would be selected to be the closest among DNS resolvers, and CDN would choose the closest CDN server of that DNS resolver, hence alleviates the problem of discrepency of geographical location of DNS server and client.

---

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS168 document by filling out the anonymous feedback form:

`https://piazza.com/brown/fall2017/cs168`.