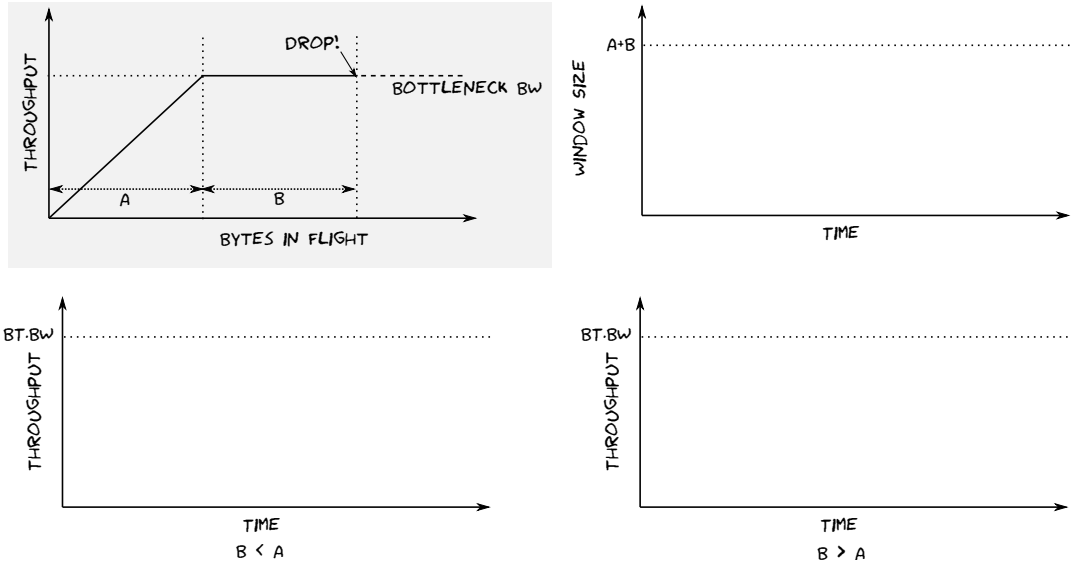
Homework 3

Due: 11:59pm, 10 Dec 2019

1. TCP Congestion Consider a single TCP RENO, going through a bottleneck link, and the figure below. The shaded graph in the figure, similar to the one we saw in class, shows the throughput as we increase the window size (Bytes in Flight). Assume that every time the window size reaches size A + B, a single drop occurs, and that it is detected by a three duplicate acks.



a. What does the number of bytes A in the shaded graph represent?

b. What does the number of bytes B in the shaded graph represent?

c. Draw a graph of the **window size versus time**, for this connection. Annotate the y axis with any important values in your graph (no need to annotate the x axis). Time starts at the beginning of the connection. You should include at least two periods if anything will repeat.

d. Now draw a graph for the **throughput versus time**, for the case in which B is less than A. (Hint: use the shaded graph, and your knowledge of the evolution of the window, as guides).

e. Draw a similar graph of **throughput versus time**, for the case in which B is greater than A.

2. Transport Layer

a. What is the main functionality that UDP adds on top of IP?

b. TCP provides the abstraction of an in-order, reliable stream of bytes, with congestion control. How would you change TCP to, instead, offer the abstraction of reliable, congestion-controlled messages, where each message would be one or more segments? The segments of each message would have to be delivered in order, but different messages wouldn't have to be delivered in order relative to each other (i.e., an incomplete message shouldn't block the receiver application of getting other messages). You should describe any extra fields you would add to the header, and how the sender and receiver would react to them. You should also describe how acks would work. There should still be a connection, over which many messages would flow, and congestion control should still operate at the level of the connection.

2. DNS I issued two queries a few seconds apart to resolve `www.google.com`, and got the following responses:

Query 1:

```
;; QUESTION SECTION:
;www.google.com. IN A

;; ANSWER SECTION:
www.google.com.     300 IN  A   172.217.12.164
www.google.com.     300 IN  A   172.217.30.100

;; Query time: 44 msec
```

Query 2:

```
;; QUESTION SECTION:
;www.google.com. IN A

;; ANSWER SECTION:
www.google.com.     294 IN  A   172.217.30.100
www.google.com.     294 IN  A   172.217.12.164

;; Query time: 1 msec
```

a. How long apart were these queries, and how do you know?

b. Why are the two answers in different orders in the responses to queries 1 and 2?

c. Why did the second query take so much less time than the first one?

d. Google uses values up to 300 in the TTL. `cs.brown.edu` uses values up to 86400. Why could this be?

e. Explain how you can use DNS (assuming you control the authoritative server) to send users to close-by servers?

f. Given that the DNS server from the previous answer communicates with the client's resolver, and not with the client itself, explain why the solution may not always work.

g. Explain how, if you control a country's ISPs, you can use the DNS system to block access to sites you don't want people to access.

4. RPC Consider the snowcast project. The handout for snowcast defines two types of commands and three types of replies.

a. Choose one of grpc[1], thrift[2], or cap'n'proto[3], and define, using their IDLs, the 5 messages of snowcast. (Note that this is asking you to write the IDL description in text, not to download any of the frameworks and/or compile anything! Also, you may ignore fields in the snowcast messages that loose their functions in the RPC equivalents.)

b. How would using the above RPC system simplify your snowcast code? What would not change?

[1] `https://grpc.io/`

[2] `https://github.com/apache/thrift`. If you choose thrift, take a look at the tutorial/tutorial.thrift file in the repo.

[3] `https://capnproto.org/`