Homework 3

Due: 16 November, 2012, 4pm

Problem 1 - TCP and Loss

[4 pts]

It is challenging to have TCP perform well in paths with high bandwidth-delay products. Here we will see that TCP is very sensitive to loss, and that it takes a long time to get close to capacity of the path, in these scenarios.

We saw in class an approximation of the relationship between TCP's throughput S and loss rate λ (measured in fraction of segments that are lost):

$$S \approx \sqrt{\frac{3}{2}} \times \frac{MSS}{RTT\sqrt{\lambda}}.$$

a. According to our simplified model, for a path with bandwidth of 1Gbps, RTT of 100ms, and a MSS of 1250B, what does the loss rate have to be so that we achieve $0.75 \times$ 1Gbps throughput?

b. What is the average throughput if everything remains the same, but the loss rate increases by 3×? And if everything remains the same as in (a), but the RTT reduces to 10ms?

c. Now suppose you upgrade your network to 10Gbps, with the same RTT and MSS as in (a). What is the loss rate you need to achieve $0.75 \times$ 10Gbps throughput?

d. In the same 10Gbps network with 100ms RTT and MSS of 1250B, determine the ideal window size W (in segments) to fill the pipe. Now assume that TCP Reno reached this size, had a loss, and halved its window to $W/2$ (segments). How long will it take for TCP to reach the window size of W, and how much data will it send during that time?

Problem 2 - TCP Timeout

[6 pts]

The original algorithm for TCP timeout was suggested as

$$\text{EstRTT} = (1 - \alpha) \times \text{EstRTT} + \alpha \times \text{SampleRTT}$$

$$\text{Timeout} = 2 \times \text{EstRTT}.$$

The improved Jacobson/Karels algorithm adds a calculation for the standard error and uses it in the timeout calculation:

$$\text{EstRTT} = (1 - \alpha) \times \text{EstRTT} + \alpha \times \text{SampleRTT}$$

$$\text{EstDev} = (1 - \beta) \times \text{EstDev} + \beta \times | \text{SampleRTT} - \text{EstRTT} |$$

$$\text{Timeout} = \text{EstRTT} + 4 \times \text{EstDev}.$$

Assuming that the current values for EstRTT and EstDev are, respectively, 1.0 and 0, that $\alpha = \beta = 1/8$, and that suddenly the RTT becomes 5 with no deviation, **compare the two algorithms over 6 update intervals**. How many timeouts does each algorithm experience in this situation? Show a little table for each algorithm, in the format below, with the successive values of EstRTT, EstDev, and Timeout. (Use a spreadsheet program or write a little program to calculate the table. You can use either the floating point version of the algorithm, as above, or the integer version that multiplies both sides of the equations by $1/\alpha$ or $1/\beta$, and uses integer operations only, see the Implementation section of these algorithms in the book, or Appendix A.2 of Van Jacobson's original paper[1]).

Original Algorithm (Initial EstRTT = 1.0)

Time	SampleRTT	EstRTT	Timeout Value	Timeout?
1	5			
...				
6	5			

Jacobson/Karels Algorithm (Initial EstRTT = 1.0, Initial EstDev = 0.0)

Time	SampleRTT	EstRTT	EstDev	Timeout Value	Timeout?
1	5				
...					
6	5				

Problem 3 - DNS and the Great Firewall of China

[4 pts]

Part of the operation of the GFC is to respond to DNS queries with wrong information (it's not the only way it prevents access to unwanted sites, though). You are going to see this for yourself below.

a. Use `dig` to find at least two addresses for www.facebook.com and list them. Look these up to find if they do belong to Facebook. You can use a tool such as `http://ip-lookup.net/index.php`, and look the different sections of information. Show some (brief) evidence.

b. Do a reverse (PTR) lookup and show the command and the answer. Is it consistent with the above (it doesn't have to be the same name, but some name that belongs to Facebook)?

[1] Van Jacobson and Michael Karels, *Congestion Avoidance and Control*, In Proceedings of SIGCOMM 88. `http://ee.lbl.gov/papers/congavoid.pdf`

c. Let's use `dig` with the `+norec` option to get the answer from the source. Starting from the root, i.e., '.', list the series of queries you make, until you get the answer *from Facebook's own nameservers*. List the queries you make and the final answer.

d. Now let's look up Facebook's address from within China. Go to `https://sites.google.com/site/kiwi78/public-dns-servers` and find a DNS server in China. List the DNS server you use. Look up www.facebook.com a few times, list the answers, and try to find where in the world the addresses are, and, if possible, who owns them. Are any of them Facebook's addresses?

e. As an aside, use `http://just-ping.com` to measure the latency from several vantage points on the Internet to Facebook's web server (choose one the IP addresses from (a)). There are probably some inconsistencies (two servers separated by a large geographical distance with ping times that add up to less than the speed-of-light latency between the two places). How is Facebook pulling this off? What mechanism are they probably using?

Problem 4 - HTTP

[4 pts]

Web pages are composed of several objects, and a browser has to download all of them to properly display the intended content. HTTP 1.0 allowed one object per TCP connection. HTTP 1.1 introduced persistent connections, that allow several requests to be made in the same HTTP connection, and pipelined requests, that allow several requests to be sent at once, before the answers come back. (This question is looking for qualitative answers, no need for elaborate examples...)

a. With persistent connections alone, the browser has to request one object after it receives the previous one. Do persistent connections, then, offer any advantage over a sequence of individual connections, one after the other?

b. Pipelined connections allow the user agent to send several requests at once, and the server can respond to all of them in sequence. What is the advantage of this strategy over persistent connections alone?

c. We saw that another common strategy to increase performance is for the Browser to open several parallel connections. What is the effect that this has on the bandwidth, if the client-server flows are sharing the bottleneck link with one other flow and uses 8 parallel, persistent connections? If you have N objects, is it always better to open as many connections as possible? How does that depend on the sizes of the objects?

d. Originally the suggested initial window for TCP was one or two segments. In 2002, RFC 3390 raised it to four. Recently, Google proposed raising it to 10, or even 16 segments. Why is this good for small web pages? Give two potential dangers of this change.