Homework 3

Due: 22 April, 2011, 4pm

Problem 1 - TCP

a. What is the difference between TCP Tahoe and TCP Reno? Why is Reno an improvement?

TCP Reno introduces Fast Recovery: on triple duplicate ACKs, the congestion window is halved instead of reset to 1; the missing segment is retransmitted immediately (fast retransmit); and the algorithm remains in congestion avoidance mode, growing the window using AIMD. On timeout, TCP Reno is identical to TCP Tahoe. TCP Reno is an improvement because the congestion window remains larger and more data can be sent.

b. Why does TCP ignore retransmitted segments when calculating the RTT?

When an ACK for the retransmitted segment returns, it is not possible to determine whether the ACK is for the first transmission of the segment, or for a later retransmission. Therefore, it is not possible to calculate the RTT for the segment.

c. We saw that an approximation for the average throughput of TCP is given by

$$T = \frac{1.22MSS}{RTT\sqrt{L}},$$

where L is the loss rate. For a path with 1,500 byte segments, RTT of 100ms, what is the loss rate if we want to saturate a link of 100Mbps? What about a link of 10Gbps?

First, we re-write the formula to solve for L,

$$L = \left(\frac{1.22 \times MSS}{T \times RTT}\right)^2$$

Next, we convert our MSS to bits or our throughput to bytes to solve for 100Mbps:

$$\left(\frac{8 \times 1.22 \times 1500}{100 \times 10^6 \times 100 \times 10^{-3}}\right)^2 \approx 2.143 \times 10^{-4}\%$$

And for 10Gbps:

$$\left(\frac{8 \times 1.22 \times 1500}{10 \times 10^9 \times 100 \times 10^{-3}}\right)^2 \approx 2.143 \times 10^{-8}\%$$

d. Suppose you have two TCP connections sharing a bottleneck link on the network: connection A has an RTT of 50ms and connection B has an RTT of 100ms, and they are both operating in TCP Reno congestion avoidance mode. In the long run, will they reach the same throughput? Justify your answer using a Chiu-Jain phase plot (lecture 11). Assume the flows start at a point well below the full utilization of the link, and that they both reduce their rates by half whenever they cross the full utilization line. Hint: draw what happens at every increment of 50ms.

   This is the answer.

Problem 2 - TCP and HTTP

Suppose that a Web browser has to download 30 objects from the same server to properly display a page. Assume that these objects are all 15KB long and that the MSS for the connection is 1KB. The communication between the client and the server has to go through a bottleneck link that has a total bandwidth of 1MB/s, and there is one TCP flow already present in this bottleneck link. Assume that the RTT for your flows, as well as for the other flow already there, is 100ms.

In this problem you have to answer the following question: how many TCP connections should the browser open to the server to finish downloading all of the files in the fastest way?

We are going to assume an idealized version of TCP, that has only two phases after regular connection establishment: slow start and constant rate. You can assume that TCP will magically know the size of the `cwnd` that will lead to the "fair" sharing of the bottleneck link. In other words, assume that TCP will start each connection in slow start and then stop growing the window once it reaches the "fair" size of the congestion window. You can also ignore the closing of the connections.

a. If yours were the only flow in the bottleneck link, what would be the size of the congestion window that would maximally utilize the bottleneck link? What would the window size be if you added the other flow we mentioned above? (Assume this extra flow is going to be there for all subsequent questions).

   This is the answer.

b. How large would an object have to be for your connection to reach this window size?

   This is the answer.

c. What would be the size of the congestion window if you now have n flows (plus the one extra flow that is not yours and it already there) ?

   This is the answer.

d. If the browser can open just one concurrent TCP connection to the server, using HTTP/1.0, how long would it take to transfer all of the 30 objects?

This is the answer.

e. What if the browser switches to HTTP/1.1 and requests a persistent connection to the server?

This is the answer.

f. What if the browser adds HTTP/1.1 pipelining? (Assume all of the 30 requests can fit in one segment), how long does it take now?

This is the answer.

g. Now the browser gets greedy, and decides to open 30 parallel connections to the server, requesting one object on each connection. Again assume that TCP will do slow start, and magically stop growing the window once (and if) it reaches the fair size. How long will it take to transfer all files?

This is the answer.

h. How many connections should the browser open (between 1 and 30) to minimize the total transfer time?

This is the answer.

Problem 3 - DNS

a. Give two reasons for which the DNS system scales to serve the entire Internet.

The DNS system can scale to serve the entire Internet because:

- DNS delegates authority for zones in order to spread and share responsibility for record updates.
- A client's first DNS resolver is usually local, and it maintains a cache of DNS responses, which prevents the majority of DNS traffic (reads) from burdening the Internet.
- The DNS record format allows for multiple IP addresses to be returned for a lookup, which can be used to implement load-balancing.
- Changes to DNS to not need to be propagated to all Internet hosts, which also do not have to store records for all other hosts.

b. Give a scenario in which you would set the TTL for a DNS mapping to a large value. What is the down side?

If you are running a service for which you get many DNS requests and cannot support a high load, you will want the answers to be cached for a long time. The down side is that if you want to change the mapping, you will need to wait for a long time until the TTL expires on all the records for your change to fully propagate.

c. Give a scenario in which you would set the TTL to a very small value. What is the down side?

For example, you may want to do load-balancing across many requests, sending different clients to different IP addresses on each lookup – this can be done with small TTL values. The down side is that the DNS servers will have to support a high request load.

Problem 4 - Consistent Hashing

We talked about consistent hashing in class. Consider two situations, and explain why CH is better than modulo hashing for each one.

a. When you have a fixed set of cache servers implementing consistent hashing, and a population of clients who have incomplete views of the system, *i.e.* each client only knows about a fraction of the servers.

The storage locations of the keys will not depend on the number of servers. Clients will be able to contact a server they know about, which can then answer the request by contacting another member of the DHT if necessary.

b. When you have a set of cache servers that changes (nodes come and go).

Only a few items stored in the distributed hash table will need to be transferred when the cache servers come and go. In traditional hashing, the keys are stored modulo the number of storage buckets – removing one storage bucket will affect all keys.

Problem 5 - P2P

Chord In class we discussed how the STABILIZE and NOTIFY operations in Chord will handle joins.

a. Explain why that doesn't quite work for node failures. As an example, take the ring in slide 39 of lecture 16, and assume node 50 fails.

Nodes only keep a single predecessor and successor. The STABILIZE and NOTIFY operations would not know how to reach their new successor and predecessor after the node between them fails.

b. What would you change in the description of the protocol to make the ring stabilize again after a failure? (You can, for example, change what nodes do when they detect that their predecessors fail, or change the set of nodes to which nodes store pointers).

Nodes will need to employ timeouts to detect when a NOTIFY or STABILIZE operation has failed. If it has failed, it can either try to contact a previous predecessor/successor, or it can attempt to route along the ring using predecessor/successor pointers to find the node on the other side of the failed node.

BitTorrent

a. What does a *tracker* do in BitTorrent? How would you use a DHT to replace BitTorrent trackers? Who would be the peers? What would be the keys, and the values stored for each key? How would you bootstrap the system?

The BitTorrent tracker contains a list of peers in the BitTorrent swarm from which you can download pieces of the file. In a trackerless BitTorrent, a DHT can be used to lookup which peers (the values) contain each block of the file (the keys). The peers in the DHT could be all nodes participating in the BitTorrent swarm. To bootstrap the system, the original seeder would register its own address as the value for each key in the DHT.