Liting Huang CPS 373 Midterm

1. Bandwidth and Latency

   (a) $32kb = 32 * 8000 = 256000(bits)$
   for the copper wire solution, we need to find the time based on the bitrate, while for the carrier pigeon way, the time is fixed.
   copper wire: $0.02 + 256,000/10,000,000 = 0.02 + 0.0256 = 0.0456(seconds)$
   carrier pigeon: 250,000 seconds
   Clearly, the copper wire is faster in this case.

   (b) copper wire: 0.02 + 4TB/10mbps = 0.02 + (3TB/1250KB) = 2,400,000.02 seconds
   carrier pigeon: 250,000 seconds
   The carrier pigeon is the faster way in this case.

2. Custom top-level domain in DNS

   (a) We must convince the root DNS server first.

   (b) The user will first contact one of the root servers, which returns the IP address for the TLD, .coffee. Then the client will contact one of the TLD servers, which returns the IP address of brad.coffee.

   (c) If any of the servers go down, customers, especially those near the broken server may somehow experience slow connection. Since there are only three DNS server worldwide, problems relating to any of the three servers may cause slow connections for the entire website.

3. HTTP Multiplexing

   (a) Transmission time for all files: (3200 + 8000 + 8000 + 8000)B/(8mbps) = 0.7ms
   There are four files, so we have four pairs of request/response connections.
   The headers will transmit eight times in total:$(26bytes \times 8)/(8mbps) = 0.026ms$
   There are also four client-to-server and server-to-client latency.
   Thus, in total it takes 0.7+0.026+4*100 = 400.726ms

   (b) As for a TCP connection with multiplexing, we can send the pictures a, b, c at the same time. They also have the same size, so transmission time for all files: $(3200 + 800)B/(8mbps) = 0.5ms$. Four times of headers' transmission will be counted: $(26bytes \times 4)/(8mbps) = 0.013ms$ plus two times of the client-to-server and server-to-client latency: $2 * 100 = 200ms$.
   Thus, in total it takes $0.5 + 0.013 + 200 = 200.513ms$

   (c) As we can see from the two results above, multiplexing does not make a huge difference for the file transmission time but largely reduces the total latency time. The multiple rounds of latency time add up and become one of the slowest time for requesting files. By decreasing the client-to-server and server-to-client times, multiplexing limits latency times and efficiently optimizes the file fetching time.

4. Distributed Hash Table

(a) The problem here is that while the peer who has already updated the table may get a new key-value pair from its local hash table, the other peers are not able to get the same information. Therefore, this will make the information sharing not the same across the network.

(b) Random peers may be malicious. If we trust random peers, we might set up some incorrect key-value pairs and these pairs may even change our original correct ones. The entire hash table may be crashed because of trusting random peers.

(c) I think querying multiple random peers for the same key-value pair does reduce the security concern, because we can compare the results from different peers and only update the hash table when the responses reach a consensus. In this way can we make sure that we will not make many incorrect changes.

5. Custom Transport Protocol

(a) If the five packets are sent back-to-back, many packets may be stuck in the buffer. Once they are stuck for too long and even cause the timeout, they will need to be re-transmitted, resulting in a lot of inefficiency.

(b) We can add a timeout period for every packet, and confirm that the transmission is successful once we get at least three same packets during this period. We need a countdown timer that can interrupt the sender after the given amount of time has expired.

(c) A potential problem is that due to delay, some transmissions of the current packet may be pushed into the next time period, corrupting the whole system. Another potential problem is that if only one/two of the five transmissions are successful, the packet will be lost.

6. TCP Congestion Control

(a) The achieved throughput for a, b, and c should all be R bits/second. Since the link connecting d and e can only transmit at a rate of R, the maximum throughput equals R. The link simply cannot deliver packets at a rate higher than R. The first packet reaches d will be sent to c first and the other two will be held in a buffer and wait for sending.

(b) Similarly, the achieved throughput of a and c should still be R bits/second.

(c) If c opens multiple TCP connections, the size of each connection will be smaller, so that the overall throughput will instead be limited.

7. Packet Forwarding

(a) The packet will propagate through the path $c \to a \to b \to e$. According to the question, the packet originate at c, so we first look at the table for Node c and find the next node with destination e – a. Then we go to the table for Node a and find that the next node is b. Consequently, we find node e in the table for b, and this is our destination.

(b) The packet will never reach node c and will be stuck in a self-loop of nodes b, d, e. We use the same approach as we did in part a. First, we loop up the table for node e and find that the next hop is b. Then we loop up the table for node b and

go to the table for node d. In the table for node d, we realize that the next hop for destination c is e again. This means that we come back to our starting point, so there's no way that we can ultimately reach node c.