CS 118 - Homework 3

Kuruvilla Cherian, Jahan
UID: 104436427
Section 1A

October 19, 2016

Problem 1

a

ns1.cs.ucla.edu:
(cs.ucla.edu, 14400, NS, NS1.cs.ucla.edu) $\rightarrow IP$: 131.179.128.16
ns2.cs.ucla.edu:
(cs.ucla.edu, 14400, NS, NS2.cs.ucla.edu) $\rightarrow IP$: 131.179.128.17
ns3.cs.ucla.edu:
(NS3.cs.ucla.edu, 14400, NS, NS3.cs.ucla.edu) $\rightarrow IP$: 131.179.128.18

b

www.cs.ucla.edu:
(cs.ucla.edu, 14400, A, www.cs.ucla.edu) $\rightarrow IP$: 164.67.100.181

c

mail.cs.ucla.edu:
(cs.ucla.edu, 14400, MX, mail.CS.UCLA.EDU) $\rightarrow IP$: 131.179.128.60

d

ftp.cs.ucla.edu:
(cs.ucla.edu, 14400, CNAME, ftp.CS.UCLA.EDU) $\rightarrow IP$: 131.179.128.22

e

lion.cs.ucla.edu:
(cs.ucla.edu, 14400, A, lion.cs.ucla.edu) $\rightarrow IP$: 131.179.128.39

Problem 2

We imagine running the process of a DNS name resolution starting from the root servers, so we begin by running the command **dig @a.root-servers.net com NS** to get all the name servers from the root server. From the list of all the Generic Top Level domain servers, I choose the first and run the same command on this server as **dig @e.gtld-servers.net verisignlabs.com NS**, but now we look for the verisignlabs.com domain within this name server to find it's respective name server. Running this we find several servers of which I run the dig command but with more of the domain as follows **dig @av1.nstld.com validatorsearch.verisignlabs.com**. This returns the locations of this domain of which I choose the first and run dig **dig @vfns1.verisignlabs.com prefetch.validatorsearch.verisignlabs.com**. This finally gives us the CNAME resolution for the link as *cn1476863900233428.validatorsearch.verisignlabs.com*. Doing the final dig

(**dig cn1476863900233428.validatorsearch.verisignlabs.com**)to search for this domain we find it at **127.0.0.1** (localhost). Essentially we ran through all the servers from root down to the location of the given domain name in an iterative process.

Problem 3

$01010011 + 01100110 = 10111001 \rightarrow 10111001 + 01110101 = 00101110 + 1 \rightarrow 00101111 \rightarrow$
$1'scomplement =$ **11010000**
We use the 1's complement of the sum (the *checksum*) to reduce the computation time with machine cycles and to reduce ambiguity with endianness. Essentially when the transport protocol was made, the computation in terms of machine cycles was important, and if we used the sum from sender and compared it to the receiver's end then that would take longer than the current schema where we simply add the receiver's sum with the sender's checksum. If there is no error in the transfer, then this addition should yield all 1's, thus if we have anything different then we have an error! This comparison to the zero value, was a lot faster than comparing two sums back in the day's of the protocol's conception, and as mentioned before avoids endianness issues as all it is doing is checking for all 1's which would be the same regardless of the machine's architecture. There is no possibility of a 1-bit error going undetected, but there is a chance of a 2-bit error going undetected. The reason for the former is that if there is a 1-bit error then the sum between the sender's checksum and the receiver's sum would not be all 1's, in which case we have an error. However, there is a chance (however low it may be) for two bits in the data segments to be flipped in exactly the same bit positions, which would then yield a valid sum and checksum, but would have a bit error within it.

Problem 4

The most common port number for SSH is port 22. When ssh'ing from a client to a server, the client's port number isn't as important because it is local to the host machine.

a

$$port_A = 4 \rightarrow port_S = 22$$

b

$$port_B = 6 \rightarrow port_S = 22$$

c

$$port_S = 22 \rightarrow port_A = 4$$

d

$$port_S = 22 \rightarrow port_B = 6$$

e

Because A and B are now on different host machines (i.e. different clients) then there is a chance of the port numbers from segment A to S and B to S are the same, as they are independent of each other and are local to their hosts. The chances are relatively unlikely given how many free port numbers exists, but it is definitely possible. On separate machines because they have separate IP addresses, you send to them directly, hence different port numbers.

f

If they are on the same host then the port numbers *have to be different* because once a port number is reserved for A, that port becomes occupied, and so B will have to find a different number. This would mean we would be on the same IP address and so the server needs some way of distinguishing between the two, which would require separate port numbers.

Problem 5

a

For the cases of infrequent data transfer a NAK-only protocol would not be effective at all. This all boils down to the probability of packet loss and its frequency. If we send 10 packets, and the receiver only gets segments, 1, 3, 4 and 5, then it will send a NAK for the 2^{nd} packet but would not realize it lost segments 6-10. Basically if we lost a packet somewhere in data transmission, then the sender will assume the receiver got the packet due to the lack of the NAK and will then continue to send the packet after some time, and then the receiver will notice an out of order error and then proceed to send a NAK for the previously lost segment thus creating an out of sync issue, which would then require a retransmission of all the data again, causing more overhead and waste in time.

b

If the packets are being sent frequently with a few losses then a NAK-only protocol would be preferable to save on bandwidth costs with an ACK-only protocol. Essentially the receiver would not have to waste any bandwidth in continuously sending ACK's. Because it will experience on a few problems, then it is more optimal to send an acknowledgment on the basis of a loss which would be resolved quickly due to the high frequency of data transmission. Thus a NAK-only protocol would be beneficial in this scenario.