1. Bit Stuffing

   The bit stuffing scheme shown in the slides is to stuff a 0 when it sees 5 consecutive 1's. After stuffing, we add the flag to both side of the data. For de-stuffing, we first remove its flags and then recover the original data.

   (a) Raw data: 10011111100101111100011

   Processed data: 01111110 1001 1111 0 1001 0111 11 0 00 011 01111110

   (b) Processed data: 01111110 0011111011001111110011 01111110

   De-stuffed data: 0011 1111 1001 1111 011

2. Link Layer Protocols

   ~~If we don't need to wait for ACK and just send through the channel, it's simply to see how many bits we can send in a 20ms window. 4000 * 0.02 / 1 = 80 bits.~~

   Edit - after talking with TA and refer to the slides, I found the answer above is not correct.

   If we do need to wait for ACK, then the propagation time will be doubled. In this case the window size becomes 4000 * 2 * 0.02 / 1 = 160 bits.

   The window should be larger than or equal to the given lower bounds.

3. Distance Vector Routing

   After discussing with TAs, we found that there should be an error in the statement. Therefore, I use 9, 6, 3 rather than 6, 6, 3 for the costs between C and B, D, E.

   First we have some table like this one. It represents all the costs of pathways from B, D and E to other nodes.

B		D		E	
A	5	A	16	A	7
B	0	B	12	B	6
C	9	C	6	C	3
D	12	D	0	D	9
E	6	E	9	E	0
F	2	F	10	F	4

   Then we can create the routing table for node C.

   Routing table below.

Destination	Minimum Cost	Next Hop
A	10	E
B	9	B
C	0	NA
D	6	D
E	3	E
F	7	E

4. TCP Sequence Numbers

   If the sequence numbers have looped through all possible combinations of the 64 bits, then it

heads back and start over. Therefore, it will be $T = \frac{2^{64} bits \cdot 8 bits / byte}{75 \times 10^{12} bits / \sec} \approx 22.77 days$. So in order to make sure it won't wrap around, we need to set the lifetime smaller than that amount. TCP header related discussion can be found on piazza @171.

5. DNS

With whois command on Ubuntu, I got the following information.

```
tc233@vcm-615:~$ whois duke.edu
This Registry database contains ONLY .EDU domains.
The data in the EDUCAUSE Whois database is provided
by EDUCAUSE for information purposes in order to
assist in the process of obtaining information about
or related to .edu domain registration records.

The EDUCAUSE Whois database is authoritative for the
.EDU domain.

A Web interface for the .EDU EDUCAUSE Whois Server is
available at: http://whois.educause.edu

By submitting a Whois query, you agree that this information
will not be used to allow, enable, or otherwise support
the transmission of unsolicited commercial advertising or
solicitations via e-mail.  The use of electronic processes to
harvest information from this server is generally prohibited
except as reasonably necessary to register or modify .edu
domain names.

-----------------------------------------------------------

Domain Name: DUKE.EDU

Registrant:
        Duke University
        905 W. Main Street, Suite 18B
        Suite 2106
        Durham, NC 27701
        United States of America

Administrative Contact:
        Domain Administrator
        Duke University
```

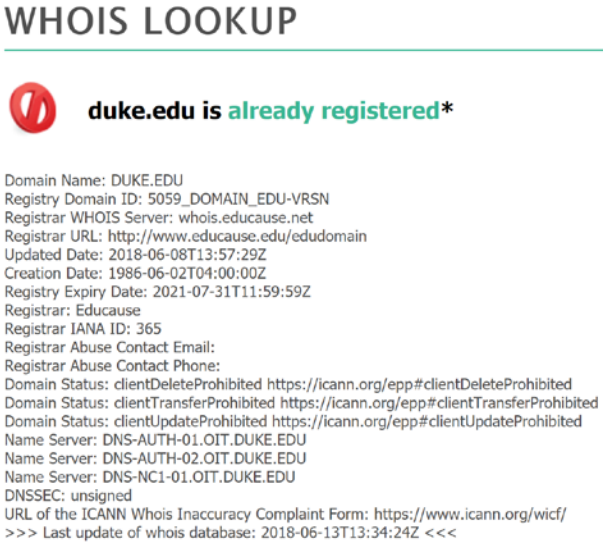
```
        334 Blackwell St.
        Suite 2106
        Durham, NC 27701
        United States of America
        +1.9196845300
        datacom-hostmaster@duke.edu

Technical Contact:
        Domain Administrator
        Duke University
        334 Blackwell St.
        Suite 2106
        Durham, NC 27701
        United States of America
        +1.9196842200
        datacom-hostmaster@duke.edu

Name Servers:
        DNS-AUTH-02.OIT.DUKE.EDU
        DNS-AUTH-01.OIT.DUKE.EDU
        DNS-NC1-01.OIT.DUKE.EDU

Domain record activated:    02-Jun-1986
Domain record last updated: 11-Sep-2018
Domain expires:             31-Jul-2021
```

As we can see from this information, the domain is registered on 19860602, and it expires on 20210731. The DNS servers for duke.edu are the 3 listed above Name Servers. Below is a screenshot.



6. Internet Services

   First, I use nc -h to see the basic usage of netcat. Then I wrote this command with pipe to post a GET request to the server of instructor.

```
echo -en "GET /awesome.txt HTTP/1.1\r\nHost: rabihyounes.com\r\nUser-
Agent: nc/0.0.1\r\nAccept: */*\r\n\r\n" | netcat rabihyounes.com 80
```

   This command works fine under my macOS system, but when I tried with vcm@Duke machine, it just returns a 400 Bad Request. I'll consult TA later.

   During dealing with this problem, I met some strange situation.

   https://piazza.com/class/jqo85inavzer4?cid=135

   Edit -

   After consulting the TA, we found that there might be several reason causing this issue. First is the EOF and end-of-line characters. Some server only establish a connection or respond when receiving strictly valid requests. Second might be server settings, as I tested on public server such as Google and my own vm@Duke and they both worked, while the server of instructor behaved inconsistently. Therefore, we could create a file and use *cat file* - to disable sending the final end-of-line character to keep the session open.

   And of course, the cat -