

Ans 1.

No.	Time	Source	Destination	Protocol	Length	Info
296	53.8715130	192.168.0.5	173.237.115.251	TCP	54	61847-443 [FTN. ACK] Seq=2 Ack=64 Win=16358 Len=0
Frame 1: 55 bytes on wire (440 bits), 55 bytes captured (440 bits) on interface 1						
Interface id: 1 (\Device\NPF_{3119D517-6ABE-4771-B545-66CAC42F9975})						
Encapsulation type: Ethernet (1)						
Arrival Time: Oct 24, 2016 01:15:11.833656000 Atlantic Daylight Time						
[Time shift for this packet: 0.000000000 seconds]						
Epoch Time: 1445746511.833656000 seconds						
[Time delta from previous captured frame: 0.000000000 seconds]						
[Time delta from previous displayed frame: 0.000000000 seconds]						
[Time since reference or first frame: 0.000000000 seconds]						
Frame Number: 1						
Frame Length: 55 bytes (440 bits)						
Capture Length: 55 bytes (440 bits)						
[Frame is marked: False]						
[Frame is ignored: False]						
[Protocols in frame: eth:ethertype:ip:tcp:ssl]						
[Coloring Rule Name: TCP]						
[Coloring Rule String: tcp]						
Ethernet II, Src: Azurewav_49:65:da (94:db:c9:49:65:da), Dst: ArrisGro_1e:b1:bf (5c:8f:e0:1e:b1:bf)						
Destination: ArrisGro_1e:b1:bf (5c:8f:e0:1e:b1:bf)						
Address: ArrisGro_1e:b1:bf (5c:8f:e0:1e:b1:bf)						
.... 0. = LG bit: Globally unique address (factory default)						
.... 0. = IG bit: Individual address (unicast)						
Source: Azurewav_49:65:da (94:db:c9:49:65:da)						

1. From the screenshot above, type of protocol in this frame can be described. Eg. TCP, IP etc.
2. Frame length can also be determined from this screenshot. And moreover, the frame no. time difference between earlier & latest packet received can be determined.
3. Source and destination address can also be determined.
4. Echo time for the current packet can be determined.

No.	Time	Source	Destination	Protocol	Length	Info
296	53.8715130	192.168.0.5	173.237.115.251	TCP	54	61847-443 [EST. ACK] Seq=2 Ack=64 win=16358 len=0
<div> <div>Source: Azurewav_49:65:da (94:db:c9:49:65:da)</div> <div>Address: Azurewav_49:65:da (94:db:c9:49:65:da)</div> <div>....0. = LG bit: Globally unique address (factory default)</div> <div>....0. = IG bit: Individual address (unicast)</div> <div>Type: IP (0x0800)</div> </div>						
<div> <div>Internet Protocol Version 4, Src: 192.168.0.5 (192.168.0.5), Dst: 192.229.163.25 (192.229.163.25)</div> <div>Version: 4</div> <div>Header Length: 20 bytes</div> <div> <div>Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00: Not-ECT (Not ECN-Capable Transport))</div> <div>0000 00.. = Differentiated Services Codepoint: Default (0x00)</div> <div>....00 = Explicit Congestion Notification: Not-ECT (Not ECN-Capable Transport) (0x00)</div> </div> <div>Total Length: 41</div> <div>Identification: 0x2fac (12204)</div> <div> <div>Flags: 0x02 (Don't Fragment)</div> <div>0... .. = Reserved bit: Not set</div> <div>.1... .. = Don't fragment: Set</div> <div>..0. = More fragments: Not set</div> </div> <div>Fragment offset: 0</div> <div>Time to live: 128</div> <div>Protocol: TCP (6)</div> <div> <div>Header checksum: 0xa676 [validation disabled]</div> <div>[Good: False]</div> <div>[Bad: False]</div> <div>Source: 192.168.0.5 (192.168.0.5)</div> <div>Destination: 192.229.163.25 (192.229.163.25)</div> <div>[Source GeoIP: Unknown]</div> <div>[Destination GeoIP: Unknown]</div> </div> </div>						
<div> <div>Transmission Control Protocol, Src Port: 61905 (61905), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 1</div> <div>Source Port: 61905 (61905)</div> </div>						
0000	5c 8f e0 1e b1 bf 94 db c9 49 65 da 08 00 45 00	\.....Ie...E.				
0010	00 29 2f ac 40 00 80 06 a6 76 c0 a8 00 03 c0 e5	.)/. @... .v.....				
0020	a3 19 f1 d1 01 bb 90 2b 05 a3 c1 01 98 65 50 10+eP.				
0030	3f 58 69 0c 00 00 00 00	?xi.....				

1. Using this screenshot, source address, IP version with source IP, destination IP, header length can be determined.
2. Header checksum info can also be seen in this.
3. Flag info related to fragmentation, offset TTL & protocol can also be determined using this screenshot.

No.	Time	Source	Destination	Protocol	Length	Info
296	53.8715130	192.168.0.5	173.237.115.251	TCP	54	61847-443 [EST. ACK] Seq=2 Ack=64 Win=16358 Len=0
Transmission Control Protocol, Src Port: 61905 (61905), Dst Port: 443 (443), Seq: 1, Ack: 1, Len: 1						
Source Port: 61905 (61905)						
Destination Port: 443 (443)						
[Stream index: 0]						
[TCP Segment Len: 1]						
Sequence number: 1 (relative sequence number)						
[Next sequence number: 2 (relative sequence number)]						
Acknowledgment number: 1 (relative ack number)						
Header Length: 20 bytes						
... 0000 0001 0000 = Flags: 0x010 (ACK)						
000. = Reserved: Not set						
...0 = Nonce: Not set						
... 0... = Congestion window Reduced (CWR): Not set						
... .0.. = ECN-Echo: Not set						
... ..0. = Urgent: Not set						
... ...1... = Acknowledgment: Set						
... 0... = Push: Not set						
...0.. = Reset: Not set						
...0. = Syn: Not set						
...0 = Fin: Not set						
Window size value: 16216						
[Calculated window size: 16216]						
[Window size scaling factor: -1 (unknown)]						
Checksum: 0x690c [validation disabled]						
[Good checksum: False]						
[Bad checksum: False]						
Urgent pointer: 0						
[SEQ/ACK analysis]						
Secure Sockets Layer						
0000	5c 8f e0 1e b1 bf 94 db c9 49 65 da 08 00 45 00	\.....Ie...E.				
0010	00 29 2f ac 40 00 80 06 a6 76 c0 a8 00 05 c0 e5	.)/.@...v.....				
0020	a3 19 f1 d1 01 bb 90 2b 05 a3 c1 01 98 65 50 10+eP.				
0030	3f 58 69 0c 00 00 00	?xi....				

1. Source and destination port numbers, Sequence no., Acknowledgement no., TCP segment no., header length can be determined using this screenshot.
2. The various values on flags can also be determined like Push, Reset, Syn, Finetc, Urgent, Acknowledgement.

Ans2. executable file named Source code.py is attached.
And along with this other files bit_stuffed.txt and data_restored.txt is also attached.

```
import binascii
import re
import sys

def write_file(input_file_path, output_file_path):
    with open("orig_data.txt") as f, open(output_file_path, 'w') as fout:
        for line in f:
            if line.strip():
                binary_string = bin(int(binascii.hexlify(line.strip()), 16))
                bit_stuffed = re.sub('11111', '111110', binary_string)
                fout.write(bit_stuffed+'\n')
            else:
                fout.write(line+'\n')

def read_file(input_file_path, output_file_path):
    with open("bit_stuffed.txt", 'rb') as f, open(output_file_path, 'w') as fout:
        orig = ''
        for line in f:
            if line.strip():
                bit_unstuffed = re.sub('111110', '11111', line.strip())
                n = int(bit_unstuffed, 2)
                orig = binascii.unhexlify('%x' % n)
                fout.write(orig)
            else:
                fout.write(line)

def main():
    write_file(sys.argv[0], "bit_stuffed.txt")
    read_file("bit_stuffed.txt", "data_restored.txt")

if __name__ == "__main__":
    main()
```

[illegible]

Remainder R(x) is 000 and when this is subtracted from M'(x), we get actual transmitted string that is:- $P(x) = M'(x) - R(x) = 110100111101000$
& no error is detected for CRC.

Ans3b

$$(b) \quad P(x) = 10110011101$$

$$G(x) = 1001$$

$$R(x) = 101$$

$$\begin{array}{r}
 \begin{array}{c} 1001 \end{array} \overline{) \begin{array}{c} 10110011101 \\ 1001 \\ \hline 0100 \\ 0000 \\ \hline 1000 \\ 1001 \\ \hline 0011 \\ 0000 \\ \hline 0111 \\ 0000 \\ \hline 1111 \\ 1001 \\ \hline 1100 \\ 1001 \\ \hline 1010 \\ 1001 \\ \hline 010 \end{array}} \\
 \hline
 010 \leftarrow R(x)
 \end{array}$$

Since $R(x) = 010$
There is Error.

Since we get the remainder $R'(x) = 010$ so "Error has been detected" in the transmitted string.

Ans 3c File named Ans3c.py is attached. Code for both a and b are done together below.

```
def transmitter(msg,gen,gencode):
    msgg = msg+gencode
    msg = list(msg)
    msgg = list(msgg)
    gen = list(gen)
    for p in range(len(msgg)-len(gen)):
        if msgg[p] == '1':
            for q in range (len(gen)):
                msgg[p+q] = str((int(msgg[p+q])+int(gen[q]))%2)

    remainder = (msgg[-(len(gen)-1):])
    print("remainder is","".join(remainder))
    transmitted_msg = msg+remainder
    print("message to be transmitted, without error:","".join(transmitted_msg))

def error_checker(transmitted_msgg,gen):
    transmitted_msgg = list(transmitted_msgg)
    gen = list(gen)
    for i in range(len(transmitted_msgg)-len(gen)):
        if transmitted_msgg[i] == '1':
            for j in range (len(gen)):
                transmitted_msgg[i+j] = str((int(transmitted_msgg[i+j])+int(gen[j]))%2)

    error = (transmitted_msgg[-(len(gen)-1):])
    error = list(error)
    for m in range(len(error)):
        if error[m] == '1':
            print("since there is remainder, error has been detected","".join(error))
            break
        elif m == (len(error)-1):
            print("there is No error")

transmitter("100000011000","10011","0000")
error_checker("1000000110000000","10011")
```

O/P screenshots:-

```
transmitter("100000011000","10011","0000")
error_checker("1000000110000000","10011")
```

```
('remainder is', '1100')
('message to be transmitted, without error:', '1000000110001100')
('since there is remainder, error has been detected', '1100')
>>>
```

Console History log IPython console

Encoding: UTF-8-GUESSED Line: 34 Column: 42 Memory: 40 %

If receiver receives the same message as sent from transmitter, then there is no error.

```
transmitter("100000011000","10011","0000")
error_checker("1000000110001100","10011")
```

```
('remainder is', '1100')
('message to be transmitted, without error:', '1000000110001100')
there is No error
>>>
```

Console History log IPython console

Encoding: UTF-8 Line: 34 Column: 42 Memory: 39 %

Ans3d

executable file is named Ans3d.py is attached.

Burst error length	No of frame	No of frame error detected
<32 bits	1000	1000
=32 bits	1000	1000
>32 bits	1000	1000

Currently the code is running for error bit less than 32, need to add/remove few comment of code manually to run it under different conditions.

```
import random
```

```
x=0
```

```
def transmitter(msg,gen,gencode):
```

```
    msgg = msg+gencode
```

```
    msg = list(msg)
```

```
    errorr = "10010001010101111111111000011111"
```

```
    errorr = list(errorr)
```

```
    gen = list(gen)
```

```
    msgg = list(msgg)
```

```
    for p in range(len(msgg)-len(gen)):
```

```
        if msgg[p] == '1':
```

```
            for q in range (len(gen)):
```

```
                msgg[p+q] = str((int(msgg[p+q])+int(gen[q]))%2)
```

```
    remainder = (msgg[-(len(gen)-1):])
```

```
    print("remainder is:","".join(remainder))
```

```
    transmitted_msg = msg+remainder
```

```
    print("message to be transmitted, without error is:","".join(transmitted_msg))
```

```
    #32-bit error_generator, remove comments from below 2 lines to generate 32-bit error and  
comment other lines of code accordingly.
```

```
    #error_length = 32
```

```
    #transmitted_msg = msg+errorr
```

```
    #<32-bit error_generator, remove comments from below 3 lines to generate error less than  
32-bit and comment other lines of code accordingly.
```

```
    for b in range(1505,1533):
```

```
        error_length = 27
```

```
        transmitted_msg[b] = str(random.randint(0,1))
```

```
    #>32-bit_error generator, remove comments from below 3 lines to generate error more than  
32-bit and comment other lines of code accordingly.
```

```
    """for b in range(1475,1533):
```

```
        error_len = 57
```

```
        transmitted_msg[b] = str(random.randint(0,1))"""
```

```
    print("message transmitted, with error of length:",error_length,"".join(transmitted_msg))
```

```
    error_checker(transmitted_msg,gen)
```

```
def error_checker(transmitted_msgg,gen):
```

```
    transmitted_msgg = list(transmitted_msgg)
```

```
    gen = list(gen)
```

```

for p in range(len(transmitted_msgg)-len(gen)):
    if transmitted_msgg[p]== '1':
        for q in range (len(gen)):
            transmitted_msgg[p+q] = str((int(transmitted_msgg[p+q])+int(gen[q]))%2)

error = (transmitted_msgg[-(len(gen)-1):])
error = list(error)
for m in range(len(error)):
    if error[m] == '1':
        global x
        x+=1
        print("since there is remainder, error has been detected","",join(error),"& it is in the frame
no.",x)
        break
    elif m == (len(error)-1):
        print("there is No error")

for q in range(1000):
    rand_byte = []
    for p in range(1520):
        d = str(random.randint(0,1))
        rand_byte.append(d)

    rand_bit = "".join(rand_byte)
    new_gen = "100000100110000010001111110110001"
    new_gen_code = "0"*32

transmitter(rand_bit,new_gen,new_gen_code)

```