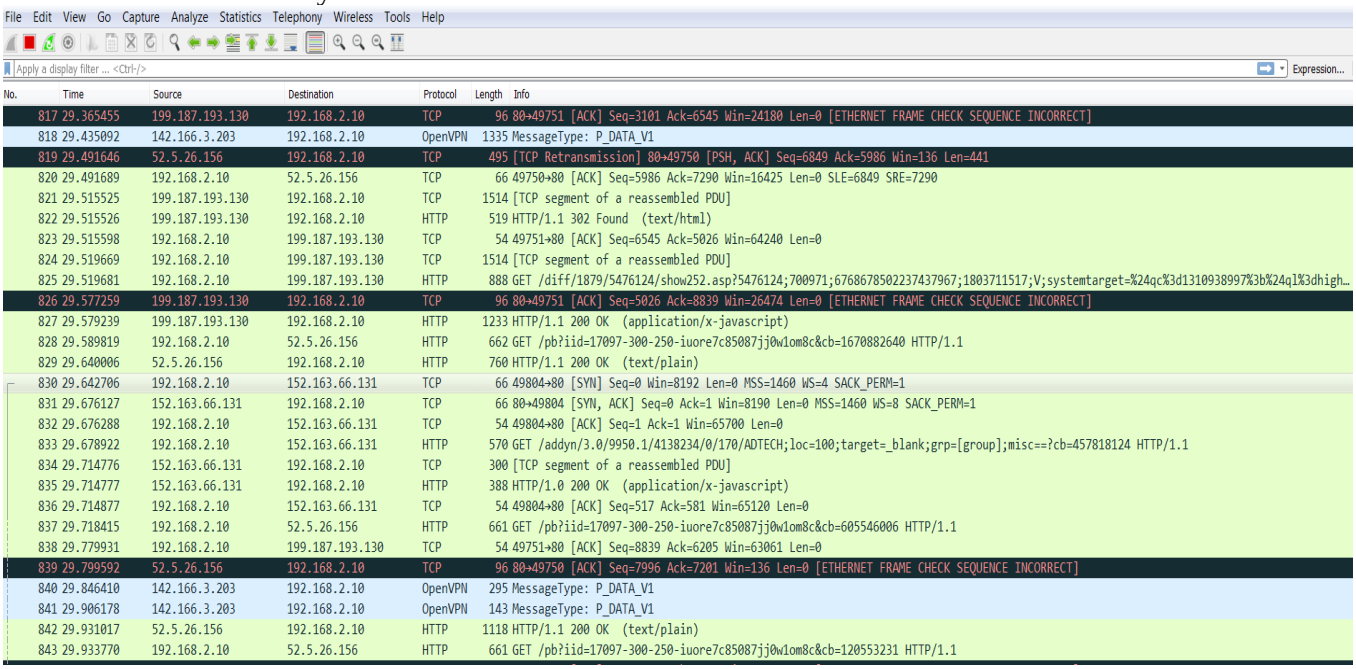
Emel URAS
B00715963

CSCI 6704

Assignment#1

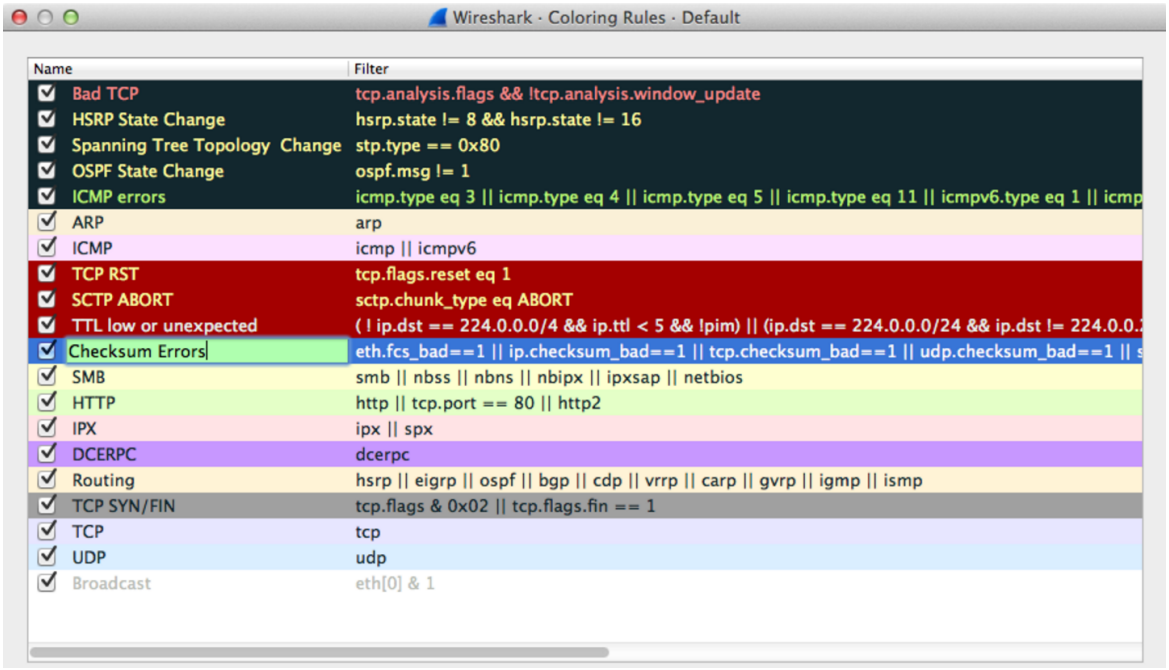
1) Wireshark is an open-source network traffic analyzer mainly used for monitoring the ongoing packets over a network for troubleshooting and managing its security and performance.

I attempted to connect www.google.com after running the Wireshark and the figure below is a sample of packets captured by the tool. As to be seen, each packets' Time, Source, Destination, Protocol, Length and Info attributes are observed by default.



Each color represents a different filter. The table below, presented in https://www.wireshark.org/docs/wsug_html_chunked/ChCustColorizationSection.html shows the rule represented by each color by default in Wireshark. Since I connected to google.com, we have mostly HTTP packets.

Emel URAS
B00715963 CSCI 6704



Clicking on a packet shows a sub-screen includes information about the headers for the packet. I clicked on the packet showed by the arrow in the first figure and observed the headers added by each layer. Transport Layer adds the TCP header shown below and it turns “DATA” to “TCP + DATA (segment)” where port numbers are seen.

```
Transmission Control Protocol, Src Port: 49804, Dst Port: 80, Seq: 0, Len: 0
    Source Port: 49804
    Destination Port: 80
    [Stream index: 28]
    [TCP Segment Len: 0]
    Sequence number: 0    (relative sequence number)
    Acknowledgment number: 0
    Header Length: 32 bytes
  ▾ Flags: 0x002 (SYN)
      000. .... .... = Reserved: Not set
      ...0 .... .... = Nonce: Not set
      .... 0... .... = Congestion Window Reduced (CWR): Not set
      .... .0.. .... = ECN-Echo: Not set
      .... ..0. .... = Urgent: Not set
      .... ...0 .... = Acknowledgment: Not set
      .... .... 0... = Push: Not set
      .... .... .0.. = Reset: Not set
    ▸ .... .... ..1. = Syn: Set
      .... .... ...0 = Fin: Not set
      [TCP Flags: ·····S·]
    Window size value: 8192
    [Calculated window size: 8192]
    Checksum: 0x70ea [unverified]
    [Checksum Status: Unverified]
    Urgent pointer: 0
  ▾ Options: (12 bytes), Maximum segment size, No-Operation (NOP), Window scale, No-Operation (NOP), No-Operation (NOP), SACK permitted
    ▸ Maximum segment size: 1460 bytes
    ▸ No-Operation (NOP)
    ▸ Window scale: 2 (multiply by 4)
    ▸ No-Operation (NOP)
    ▸ No-Operation (NOP)
    ▸ TCP SACK Permitted Option: True
```

Then, the segment goes to Network Layer where IP Header is added like below. In this header, we can see the source/destination IP addresses, Time-To-Live value (128) and Protocol ID (TCP). So, the segment is turned out a Datagram (IP Header + TCP + DATA).

```
▲ Internet Protocol Version 4, Src: 192.168.2.10, Dst: 152.163.66.131
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
  ▷ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
    Total Length: 52
    Identification: 0x10c9 (4297)
  ▷ Flags: 0x02 (Don't Fragment)
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (6)
    Header checksum: 0x4c22 [validation disabled]
    [Header checksum status: Unverified]
    Source: 192.168.2.10
    Destination: 152.163.66.131
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
```

After that, the datagram/packet is sent to Data Link Layer and Ethernet header/trailer are added here. The datagram turns into a frame. Please see the figures below:

```
▲ Ethernet II, Src: IntelCor_28:38:90 (ac:fd:ce:28:38:90), Dst: Actionte_b4:b8:b0 (4c:8b:30:b4:b8:b0)
  ▷ Destination: Actionte_b4:b8:b0 (4c:8b:30:b4:b8:b0)
  ▷ Source: IntelCor_28:38:90 (ac:fd:ce:28:38:90)
    Type: IPv4 (0x0800)
```

```
▲ Frame 830: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
    Interface id: 0 (\Device\NPF_{34FF98B6-C2F8-4B79-B2E0-3F2FFF0893C7})
    Encapsulation type: Ethernet (1)
    Arrival Time: Oct 24, 2016 21:33:44.985725000 Atlantic Daylight Time
    [Time shift for this packet: 0.000000000 seconds]
    Epoch Time: 1477355624.985725000 seconds
    [Time delta from previous captured frame: 0.002700000 seconds]
    [Time delta from previous displayed frame: 0.002700000 seconds]
    [Time since reference or first frame: 29.642706000 seconds]
    Frame Number: 830
    Frame Length: 66 bytes (528 bits)
    Capture Length: 66 bytes (528 bits)
    [Frame is marked: False]
    [Frame is ignored: False]
    [Protocols in frame: eth:ethertype:ip:tcp]
    [Coloring Rule Name: HTTP]
    [Coloring Rule String: http || tcp.port == 80 || http2]
```

In those figures, source/destination MAC addresses, Type Code (IPv4), Encapsulation Type (Ethernet), Arrival Time and Frame Length are observed. As to be seen, Wireshark is such a handy tool for network analyzers to sniff the packets over a network and reach the information carried by headers.

2) The java code , the input and output files are placed into assignment folder. Please note that, i fixed the number of bits of ASCII decimal vaules for each charachter as 7, which is the maximum digit as bit for an ASCII charachter. (127 = 1111111) The aim of fixing digit number is to make easier to convert back binary string to charachter string.

The example output for the code as follows;

```
Original Bit String :
1000100110010111101001110010110000111000111111010001000001111001110010111101001011111000101011001001100101110110011010011100111110100011101001011111
Stuffed Bit String:
1000100110010111101001110010110000111000111111001000100000111100111001011110100101111100001010110010011001011101100110100111001111100100011101001011
Bit String After Discarded Zeros:
1000100110010111101001110010110000111000111111010001000001111001110010111101001011111000101011001001100101110110011010011100111110100011101001011111
Conveted Text:
Detract yet_
delight_written_farther_his_general._If_in_so_bred_at_dare_rose_lose_good._Feel_and_make_two_real_miss_use_easy._Celebrated_delightful_an_especiall

Done
```

Emel URAS
B00715963 CSCI 6704

3)

a) Given a 12 - bit sequence 110100111101 and a divisor of 1011, find the CRC and the transmitted string.

M(x) = 110100111101
G(x) = 1011 = x^3+x^1+1 (Degree of the referrence is 3.)
M'(x) = 110100111101000

P(x) is the message to be sent. R(x) stands for the remainder.

P(x) = M'(x) – R(x)

```
1011 | 110100111101000
       1011
       ----
        1100
        1011
        ----
        1110
        1011
        ----
        1011
        1011
        ----
         0001
         0000
         ----
          0011
          0000
          ----
           0111
           0000
           ----
            1110
            1011
            ----
             1011
             1011
             ----
              0000
              ...
              ----
               000        -> Remainder
```

There is no remainder so the message to be sent is;
P(x) = M'(x) = 110100111101000

Emel URAS
B00715963 CSCI 6704

b)Given a remainder of 101, a transmitted data unit of 10110011101, and a divisor of 1001, is there an error in the data unit. Show all steps

G(x) = 1001
P(x) = 10110011101
R(x) = 101

If there is an error in the transmitted data, it should have a reminder of division to reference. Since transmitted data should be divisible to reference.

```
1001 | 10110011101
       1001
       ____
        0100
        0000
        ____
         1000
         1001
         ____
          0011
          0000
          ____
           0111
           0000
           ____
            1111
            1001
            ____
             1100
             1001
             ____
              1011
              1001
              ____
               010  --> Remainder
```

As you can see there is a remainder for P(x) that shows there is an error for this data. In order to find the correct data to be transmitted;

1) Reach the original message

P(x) = M'(x) – R(x)
M'(x) = P(x) + R(x)

```
P(x)      10110011101
R(x)      00000000101
          ___________
M'(x) = 10110011000
```

Emel URAS
B00715963 CSCI 6704

2) Find whether M'(x) is divisible to G(x) or not. If there is a remainder i will substact this remainder from the message, othervise M'(x) is equal to the massage to be transmitted P(x).

```
1001 |  10110011000
         1001
         ____
          0100
          0000
          ____
           1000
           1001
           ____
            0011
            0000
            ____
             0111
             0000
             ____
              1110
              1001
              ____
               1110
               1001
               ____
                1110
                1001
                ____
                 111      --> Remainder
```

I see that there is a remainder and this remainder helps me to find correct P(x).

P(x) = M'(x) – R(x)

```
M'(x)   10110011000
R(x)    00000000111
        ___________
P(x)    10110011111
```

The correct data to be tansmitted P(x) =10110011111.

Emel URAS
B00715963 CSCI 6704

c) Implement the sending and receiving CRC protocol by writing program routines (functions/methods) for each of the following:

The java code is placed into submitted folder.

a. Given a bit string, compute the CRC remainder and generate the bit string to be transmitted.

The message to be sent is asked to user. The G(x) is defined in the main folder. The example output for this question is represented as follows.

```
Enter message M(x):
1101011
1101011
___________Assignment#2 - Q3 -c) - a)  __________

Ref. No. : G(x) = 1101
Message: M(x) = 1101011
Transmitted Message: P(x) = 1101011010
```

In order to use the functions also for long bit strings i didnt convert the bit string to long or BigInteger. Since i wrote long division and xor operation functions to be used with bit string inputs.

b. Given a bit string with CRC remainder appended, divide by G(x) and determine if the message is error-free.

The bit string is required to be entered by user. Example output for having a reminder :

```
___________Assignment#2 - Q3 -c) - b)  __________

Enter a bit string with CRC remainder appended:

1101011000
1101011000
There is an error! - Remainder:010
```

If the message is divisible by reference number:

```
___________Assignment#2 - Q3 -c) - b)  __________

Enter a bit string with CRC remainder appended:

1101011010
1101011010
There is no error.
```

Emel URAS
B00715963 CSCI 6704

d) Use the program in (c) to run the following experiment. Use the standard CRC-32 generator polynomial. Generate a random binary number of 1520 bytes. This will be your frame. Find the remainder (4 bytes). Now introduce a random burst error of length = 32 bits in the frame of 1524 bytes. Check to see if the error is detected.

For CRC-32 G(x) is defined as:

```
String CRC32_g_X = "10000010011000001000111010110111";
```

Since, it is binary representation for CRC-32 standard polynomial.

For the experiment, i provided information about the process for each 100 iteration. After 1000 loop is comlpeted an summary information is represented as follows:

```
___________Assignment#2 - Q3 -c) - d)  __________

The experiment started!
The 100. iteration is comlpeted.
The 200. iteration is comlpeted.
The 300. iteration is comlpeted.
The 400. iteration is comlpeted.
                iteration is comlpeted.
Eclipse         iteration is comlpeted.
The 700. iteration is comlpeted.
The 800. iteration is comlpeted.
The 900. iteration is comlpeted.
The 1000. iteration is comlpeted.
Burst error length > 32: 330 Number  of  frames-error was detected: 312
Burst error length = 32: 10 Number  of  frames-error was detected: 10
Burst error length < 32: 660 Number  of  frames-error was detected: 660
```

As you can see, the algorithm is able to find all burst errors with lenght is equal to 32 or less. The rates for finding errors:

Burst Error Lenght	Number of frames	Number of frames in which error was detected	Rate for finding error
<32	660	660	%100
32	10	10	%100
>32	330	312	%94.5