Week #5

Week 5 - Understanding Transport layer and Network Layer

SRN : PES2UG20CS237

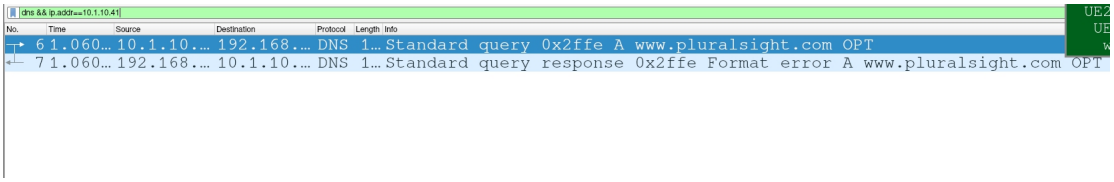
Name : P K Navin Shrinivas

Section : D

Task 1 :UDP and DNS

Opening wireshark and filtering for DNS and UDP signals specific to my ip address.

Pinging www.pluralsight.com to generate DNS packets.



Filtered packets

```
navin@usermachine:~ » dig www.pluralsight.com

; <<>> DiG 9.18.1 <<>> www.pluralsight.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: FORMERR, id: 12286
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: ba9a2fe3b83195f2 (echoed)
;; QUESTION SECTION:
;www.pluralsight.com.           IN      A

;; Query time: 0 msec
;; SERVER: 192.168.3.2#53(192.168.3.2) (UDP)
;; WHEN: Thu Mar 31 05:46:32 UTC 2022
;; MSG SIZE  rcvd: 60
```

dig command to generate UDP packets

(Q) My predictions for UDP structure :

2 bytes for source port	2 bytes for dest port
2 bytes for length	2 bytes for UDP checksum
X bytes for payload	X bytes of payload

(Q) And in wireshark : Hence concluding predictions were right!

```
▸ Internet Protocol Version 4, Src: 10.1.10.41, Dst: 192.168.3.2
▾ User Datagram Protocol, Src Port: 58300, Dst Port: 53
    Source Port: 58300
    Destination Port: 53
    Length: 68
    Checksum: 0xd829 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  ▸ [Timestamps]
    UDP payload (60 bytes)
▸ Domain Name System (query)
```

UDP in Wireshark

Why is the checksum “unverified” in wireshark UDP packets? It is because these UDP packets were genrated by dig, iptrace or tcpdump will calculate these checksums and elad to “verified” status.

```
▾ User Datagram Protocol, Src Port: 58300, Dst Port: 53
    Source Port: 58300
    Destination Port: 53
    Length: 68
    Checksum: 0xd829 [unverified]
    [Checksum Status: Unverified]
    [Stream index: 0]
  ▸ [Timestamps]
    UDP payload (60 bytes)
▸ Domain Name System (query)
```

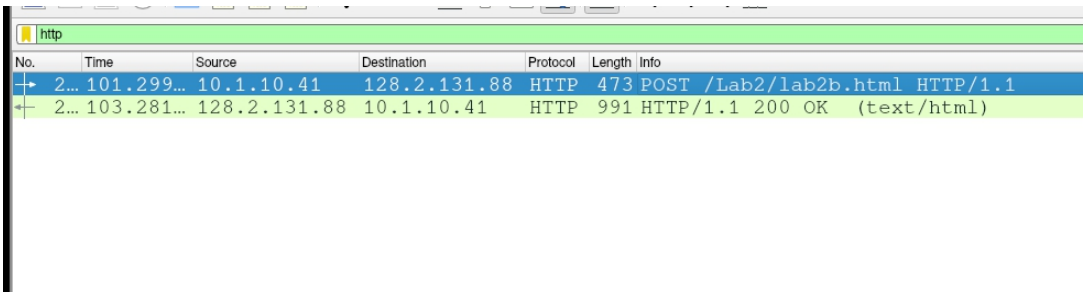
Unverified Checksum in UDP

Task 2 :TCP

Download the file from http://www.gutenberg.org/ebooks/2383.txt.utf-8

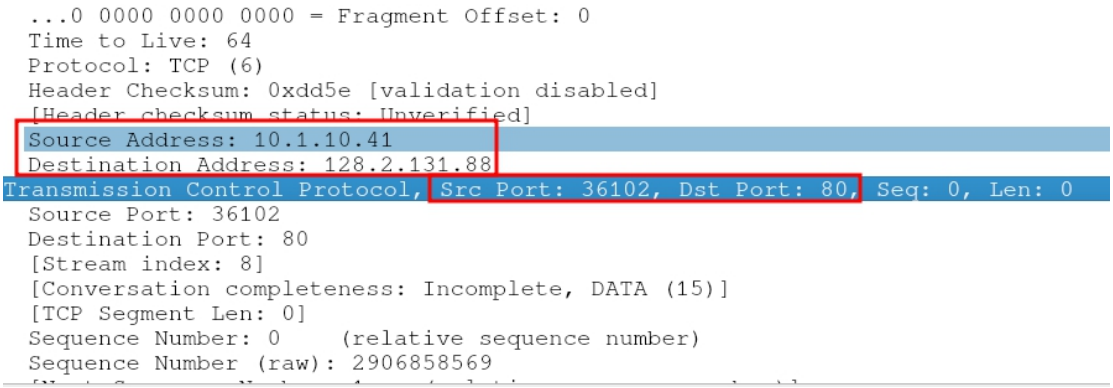
Open up wireshark and keep it prime to filter TCP packets from and to http://www.ini740.com/Lab2/lab2a.html

Upload the files and stop wireshark, wireshark output :



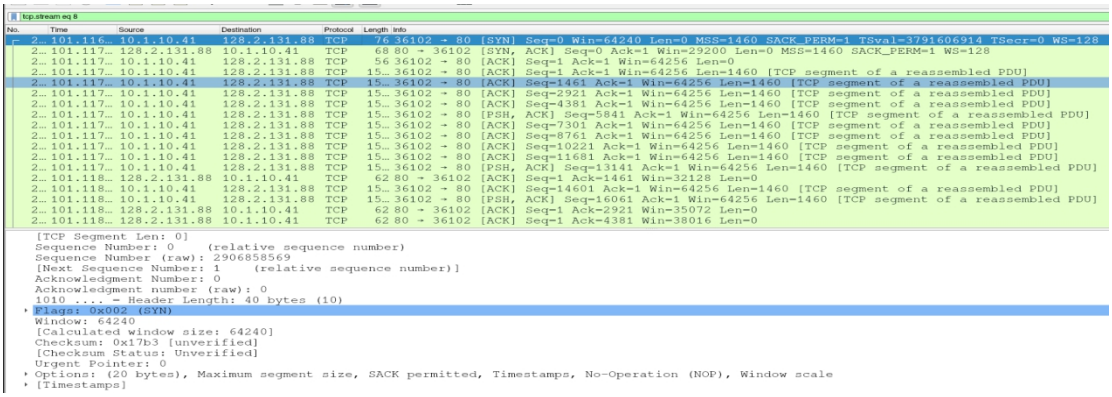
No.	Time	Source	Destination	Protocol	Length	Info
2…	101.299…	10.1.10.41	128.2.131.88	HTTP	473	POST /Lab2/lab2b.html HTTP/1.1
2…	103.281…	128.2.131.88	10.1.10.41	HTTP	991	HTTP/1.1 200 OK (text/html)

(Q) What is the source/dest ip address and source port? IP :



```
...0 0000 0000 0000 = Fragment Offset: 0
Time to Live: 64
Protocol: TCP (6)
Header Checksum: 0xdd5e [validation disabled]
[Header checksum status: Unverified]
Source Address: 10.1.10.41
Destination Address: 128.2.131.88
Transmission Control Protocol, Src Port: 36102, Dst Port: 80, Seq: 0, Len: 0
Source Port: 36102
Destination Port: 80
[Stream index: 8]
[Conversation completeness: Incomplete, DATA (15)]
[TCP Segment Len: 0]
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 2906858569
```

Point of this task is to observe TCP packets, hence filter for them :



tcp.stream eq 8

No.	Time	Source	Destination	Protocol	Length	Info
2…	101.116…	10.1.10.41	128.2.131.88	TCP	76	36102 → 80 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3791606914 TSecr=0 WS=128
2…	101.117…	128.2.131.88	10.1.10.41	TCP	68	80 → 36102 [SYN, ACK] Seq=0 Ack=1 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
2…	101.117…	10.1.10.41	128.2.131.88	TCP	56	36102 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=0
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=1 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=1461 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=2921 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=4381 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [PSH, ACK] Seq=5841 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=7301 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=8761 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=10221 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=11681 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.117…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [PSH, ACK] Seq=13141 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.118…	128.2.131.88	10.1.10.41	TCP	62	80 → 36102 [ACK] Seq=1 Ack=1461 Win=32128 Len=0
2…	101.118…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [ACK] Seq=14601 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.118…	10.1.10.41	128.2.131.88	TCP	15…	36102 → 80 [PSH, ACK] Seq=16061 Ack=1 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
2…	101.118…	128.2.131.88	10.1.10.41	TCP	62	80 → 36102 [ACK] Seq=1 Ack=2921 Win=35072 Len=0
2…	101.118…	128.2.131.88	10.1.10.41	TCP	62	80 → 36102 [ACK] Seq=1 Ack=4381 Win=38016 Len=0

```
[TCP Segment Len: 0]
Sequence Number: 0    (relative sequence number)
Sequence Number (raw): 2906858569
[Next Sequence Number: 1    (relative sequence number)]
Acknowledgment Number: 0
Acknowledgment number (raw): 0
1010 .... = Header Length: 40 bytes (10)
Flags: 0x002 (SYN)
Window: 64240
[Calculated window size: 64240]
Checksum: 0x17b3 [unverified]
[Checksum Status: Unverified]
Urgent Pointer: 0
Options: (20 bytes), Maximum segment size, SACK permitted, Timestamps, No-Operation (NOP), Window scale
[Timestamps]
```

Task 2b : TCP Basics

(Q) Sequence number of SYN packets are 0, SYN segments can be identified using the TCP flag segments. Yes wireshark can display absolute seq numbers by : edit->prefrence->protocols->tcp->untick relative numbering.

```
1010 .... = Header Length: 40 bytes (10)
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: 64240
[Calculated window size: 64240]
```

SYN Identification

```
No.  Time      Source        Destination   Protocol Length Info
224  94.7031…  10.1.10.41    54.68.82.156  TCP      56     39644 → 443 [ACK] Seq=2854338923 Ack=3373276650 Win=501 Len=0
229  97.0668…  10.1.10.41    108.159.15.…  TCP      56     [TCP Dup ACK 19#9] 57466 → 443 [ACK] Seq=2531889710 Ack=2650907141 Win=501 Len=0
230  97.0675…  108.159.15.…  10.1.10.41    TCP      62     [TCP Dup ACK 20#9] [TCP ACKed unseen segment] 443 → 57466 [ACK] Seq=2650907141 Ack=2531889711 Win=245 Len=0
231  97.7036…  54.68.82.156  10.1.10.41    TLS…     123    Application Data
232  97.7037…  10.1.10.41    54.68.82.156  TCP      56     39644 → 443 [ACK] Seq=2854338923 Ack=3373276717 Win=501 Len=0
235  100.732…  54.68.82.156  10.1.10.41    TLS…     123    Application Data
236  100.732…  10.1.10.41    54.68.82.156  TCP      56     39644 → 443 [ACK] Seq=2854338923 Ack=3373276784 Win=501 Len=0
241  101.116…  10.1.10.41    128.2.131.88  TCP      76     36102 → 80 [SYN] Seq=2906858569 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3791606914 TSecr=0 WS=128
242  101.117…  128.2.131.88  10.1.10.41    TCP      68     80 → 36102 [SYN, ACK] Seq=43468893 Ack=2906858570 Win=29200 Len=0 MSS=1460 SACK_PERM=1 WS=128
243  101.117…  10.1.10.41    128.2.131.88  TCP      56     36102 → 80 [ACK] Seq=2906858570 Ack=43468894 Win=64256 Len=0
245  101.117…  10.1.10.41    128.2.131.88  TCP      15…    36102 → 80 [ACK] Seq=2906858570 Ack=43468894 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
246  101.117…  10.1.10.41    128.2.131.88  TCP      15…    36102 → 80 [ACK] Seq=2906860030 Ack=43468894 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
247  101.117…  10.1.10.41    128.2.131.88  TCP      15…    36102 → 80 [ACK] Seq=2906861490 Ack=43468894 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
248  101.117…  10.1.10.41    128.2.131.88  TCP      15…    36102 → 80 [ACK] Seq=2906862950 Ack=43468894 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
249  101.117…  10.1.10.41    128.2.131.88  TCP      15…    36102 → 80 [PSH, ACK] Seq=2906864410 Ack=43468894 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
250  101.117…  10.1.10.41    128.2.131.88  TCP      15…    36102 → 80 [ACK] Seq=290[illegible]256 Len=1460 [TCP segment of a reassembled PDU]
251  101.117…  10.1.10.41    128.2.131.88  TCP      15…    36102 → 80 [ACK] Seq=2906867330 Ack=43468894 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
252  101.117…  10.1.10.41    128.2.131.88  TCP      15…    36102 → 80 [ACK] Seq=2906868790 Ack=43468894 Win=64256 Len=1460 [TCP segment of a reassembled PDU]
```

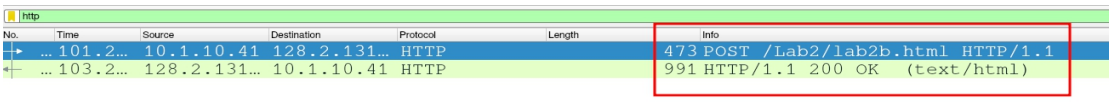
Absolute SEQ numbers

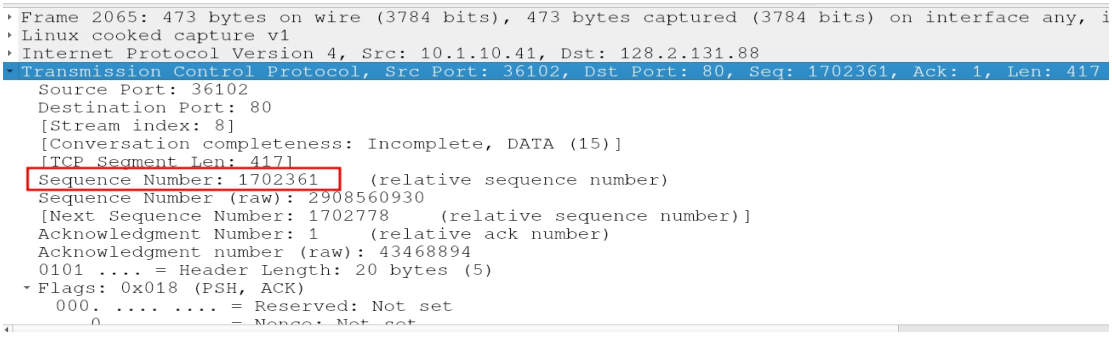
Relative TCP seq numbering

(Q) SYNACK has a relative sequence number of 1, ACK number of SYNACK packets is 1, this is because the server is asking for the best bit waiting for the three way handshake.The Flag field in TCP packets shows it is a SYNACK packet:

```
1000 .... = Header Length: 32 bytes (8)
Flags: 0x012 (SYN, 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
  .... .... ..1. = Syn: Set
  .... .... ...0 = Fin: Not set
```

(Q) TCP SEQ number (relative) of HTTP post is : 1702361, this I feel is the entirty of uploaded content and remaining payload (1702278) are for headers.

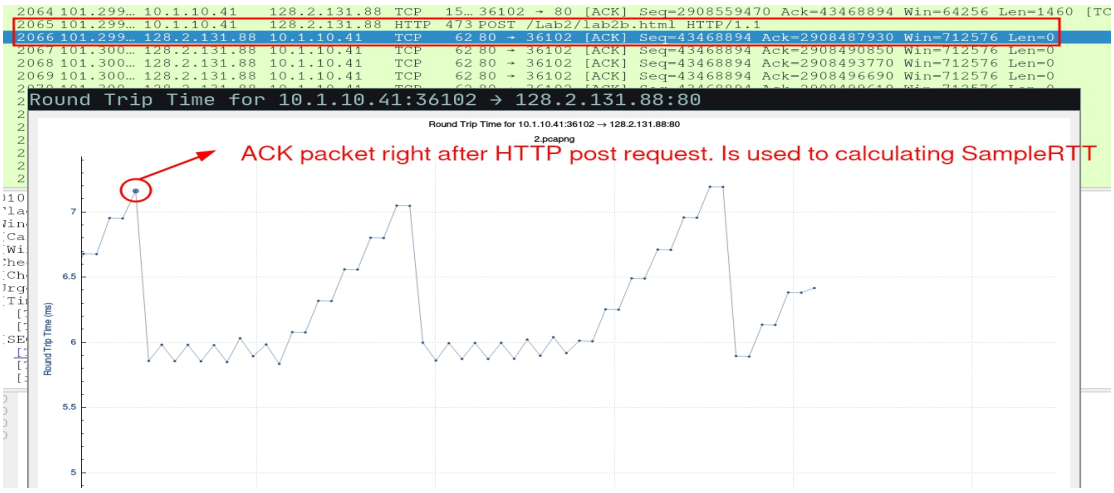




(Q)Sample RTT is simply the time taken to ACK a given segment, but to get smoother EstimatedRTT we take many samples from recent packets :

			0.005857519
		0.005854865	0.00585718725
		0.005853887	0.005856774719
		0.005848369	0.005855724004
		0.005892719	0.005860348378
		0.005834262	0.005857087581
		0.006075354	0.005884370883
		0.006315608	0.005938275523
		0.006557328	0.006015657083
		0.006799189	0.006113598572
		0.007047824	0.006230376751
		0.067936924	0.06532691974
		0.005643582273	0.005938810886

			0.005857519
		0.005854865	0.00585718725
		0.005853887	0.005856774719
		0.005848369	0.005855724004
		0.005892719	0.005860348378
		0.005834262	0.005857087581
		0.006075354	0.005884370883
		0.006315608	0.005938275523
		0.006557328	0.006015657083
		0.006799189	0.006113598572
		0.007047824	0.006230376751
		0.067936924	0.06532691974
		0.005643582273	0.005938810886



Now calculating the EstimatedRTT (In Table):

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

- Exponential weighted moving average
- influence of past sample decreases exponentially fast
- typical value: α = 0.125

The window size on packets from A to B indicate how much buffer space is available on A for receiving packets. So when B receives a packet with

window size 1, it would tell B how many bytes it is allowed to send to A. Hence looking at the window sizes here on reponses (ACK). We also see the windows are scaled with a factor of 128. Lowest observed window size : 32128