

Project 2

Berkay Barlas, 0054512

December 9, 2020

Contents

1	Part 1 - SSL Implementation and Experiments	2
1.1	Questions	2
2	Part 2 - TCP Experiments	5
2.1	Questions	5
3	Part 3 - UDP Experiments	7
3.1	Questions	7
4	Part 4 - Stop-and-Wait ARQ Protocol	9
4.1	Part 4.b - Analyzing the Implementation	9

1 Part 1 - SSL Implementation and Experiments

The maximum possible port number is 65535 (max 16-bit unsigned integer) therefore choosing 54512 + 05 = 54517 as port number will not be a problem.

1.1 Questions

1. Locate the SSL Server IP address and port number, client IP address and port number through which these agents are communicating by using Wireshark.

In this I used 4444 port as default for TCP connections and the port used for SSL connections is 54517. Both client and server IP addresses are 127.0.0.1 because it's the dedicated localhost IP.

No.	Time	Source	Destination	Protocol	Length	Info
55	8.529241	127.0.0.1	127.0.0.1	TCP	44	4444 → 55020 [RST] Seq=901 Win=0 Len=0
56	8.898383	127.0.0.1	127.0.0.1	TCP	68	55023 → 54517 [SYN] Seq=0 Win=65535 Len=0
57	8.898357	127.0.0.1	127.0.0.1	TCP	68	0.00005... 54517 → 55023 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
58	8.898363	127.0.0.1	127.0.0.1	TCP	56	0.00000... 55023 → 54517 [ACK] Seq=1 Ack=1 Win=408256 Len=0
59	8.898370	127.0.0.1	127.0.0.1	TCP	56	[TCP Window Update] 54517 → 55023 [ACK] Seq=1 Ack=1 Win=408256 Len=0
60	8.929289	127.0.0.1	127.0.0.1	TLSv1...	464	Client Hello
61	8.929253	127.0.0.1	127.0.0.1	TCP	56	0.00004... 54517 → 55023 [ACK] Seq=1 Ack=409 Win=408256 Len=0
62	8.931310	127.0.0.1	127.0.0.1	TLSv1...	216	Server Hello
63	8.931346	127.0.0.1	127.0.0.1	TCP	56	0.00003... 55023 → 54517 [ACK] Seq=409 Ack=161 Win=408256 Len=0
64	8.932678	127.0.0.1	127.0.0.1	TLSv1...	62	Change Cipher Spec
65	8.932723	127.0.0.1	127.0.0.1	TCP	56	0.00004... 55023 → 54517 [ACK] Seq=409 Ack=167 Win=408256 Len=0
66	8.932950	127.0.0.1	127.0.0.1	TLSv1...	122	Application Data
67	8.932965	127.0.0.1	127.0.0.1	TCP	56	0.00001... 55023 → 54517 [ACK] Seq=409 Ack=233 Win=408256 Len=0
68	8.933033	127.0.0.1	127.0.0.1	TLSv1...	980	Application Data
69	8.933080	127.0.0.1	127.0.0.1	TCP	56	0.00004... 55023 → 54517 [ACK] Seq=409 Ack=1157 Win=408256 Len=0
70	8.937486	127.0.0.1	127.0.0.1	TLSv1...	358	Application Data
71	8.937585	127.0.0.1	127.0.0.1	TCP	56	0.00001... 55023 → 54517 [ACK] Seq=409 Ack=1459 Win=408256 Len=0
72	8.937569	127.0.0.1	127.0.0.1	TLSv1...	130	Application Data

Frame 57: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface lo0, id 0
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
Transmission Control Protocol, Src Port: 54517, Dst Port: 55023, Seq: 0, Ack: 1, Len: 0
Source Port: 54517
Destination Port: 55023
[Stream index: 4]
[TCP Segment Len: 0]
Sequence Number: 0 (relative sequence number)
Sequence Number (raw): 1565444546
[Next Sequence Number: 1 (relative sequence number)]
Acknowledgment Number: 1 (relative ack number)
0000 02 00 00 00 45 00 00 40 00 00 40 00 00 06 00 00 ...E...@...
0010 7f 00 00 01 7f 00 00 01 d4 f5 d6 ef 5d 4e c9 c2N..
0020 96 a9 91 0b 0b 12 ff ff fe 34 00 00 02 04 3f d84...?
0030 01 03 03 06 01 01 00 0a 2e fd d3 96 2e fd d3 96
0040 04 02 00 00

Figure 1: Wireshark in Loopback mode. client port is 55023

2. Locate the data containing TCP segments. What is written in the data field? Compare it with the data you exchanged between the client and server. Why do you think is this the case?

In the data field I can clearly see the messages between client and server in TCP segments.

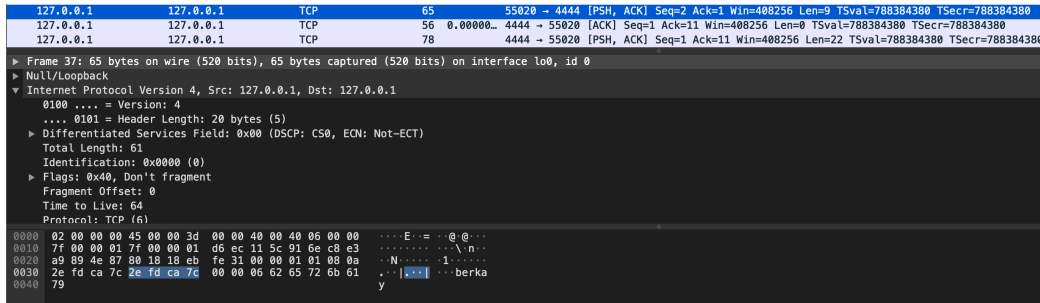


Figure 2: Wireshark Data Field for Username Message

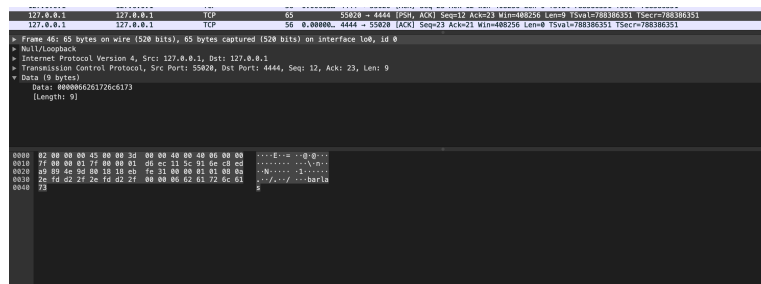


Figure 3: Wireshark Data Field for Password Message

- How many TCP segments are transmitted in total while your KUSIS username + KUSIS ID is exchanged one by one with non-persistent connections?

If investigate from the Wireshark filtering with ssl port I used, 54517, we can see that 285 TCP segments are transmitted during SSL connection for ACK messages.

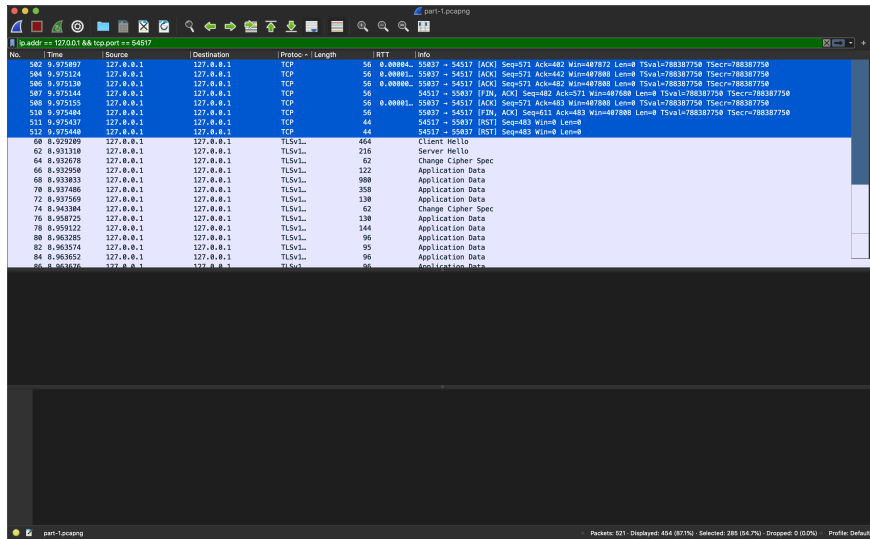


Figure 4: Wireshark Number of TCP segments During SSL connection

4. What difference did you see in the payload of SSL and TCP? Can you locate the login name and password entered by the user? Can you locate the email information?

I can't locate email information because it's not used in this communication. The payloads in TCP are visible therefore I can locate my kusus information however SSL payload is encrypted.

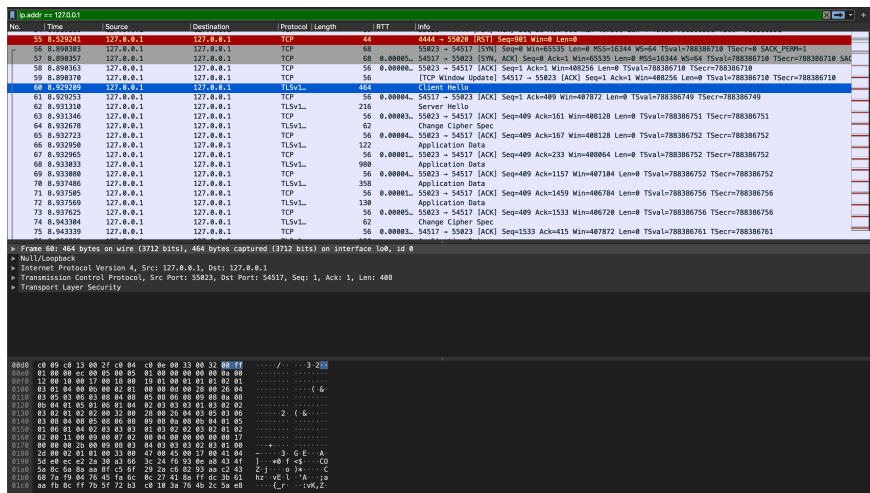


Figure 5: Wireshark SSL Payload

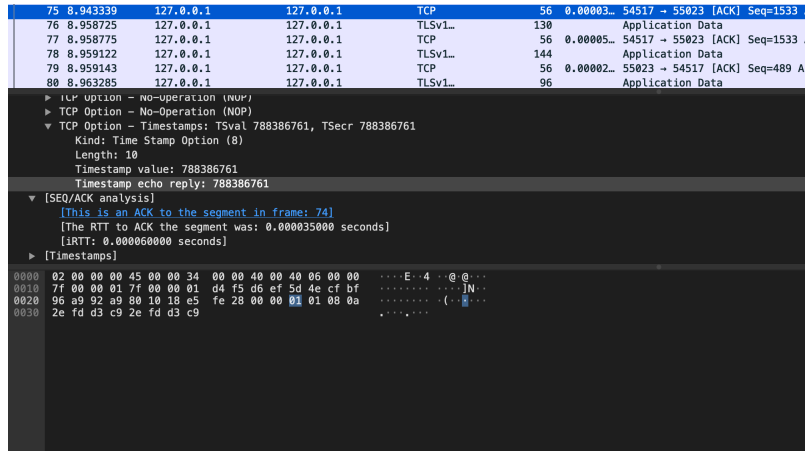


Figure 6: Wireshark SSL Payload

2 Part 2 - TCP Experiments

2.1 Questions

- What is the significance of the various IP addresses shown in the Flow Graph?
There are multiple IP addresses in the flow that shows the source and the destination of packets. This IP addresses identifies the different source and network connections.
- What are the sequence numbers (which appear in the Wireshark program) of the segments used for the 3-way handshake protocol that initiates the first TCP connection? Sequence number for first ACK from the client and SYN/ACK from the server in 3-way handshake protocol are 0. The
The raws values: 302580735,1535182375, 302580736

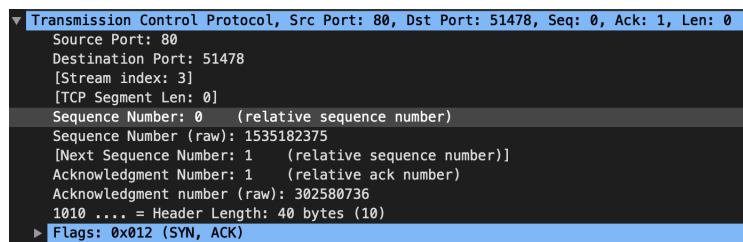


Figure 7: SYNACK segment

- What is the value of the Acknowledgement field in the SYNACK segment? How did gaia.cs.umass.edu determine that value? What is it in the segment that identifies the segment as a SYNACK segment?
As in the previous figure, the value of the ack field is 1. This acknowledgement value is equals to the sequence number of the arriving package incremented by one. The SYN and ACK nonzero flags in the segment identifies as SYNACK segment.

```

[Next Sequence Number: 1      (relative sequence number)]
Acknowledgment Number: 1      (relative ack number)
Acknowledgment number (raw): 302580736
1010 .... = Header Length: 40 bytes (10)
▼ 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

```

Figure 8: SYNACK segment flags

- What is the sequence number of the SYNACK segment sent by gaia.cs.umass.edu to the client computer in reply to the SYN?

The relative sequence number of the SYNACK is 0. Raw sequence number is 1535182375. Also

- What is the sequence number of the TCP segment containing the HTTP POST command?

The (relative) sequence number of the TCP segment is 1 and the raw sequence is 3694263361

```

▼ Transmission Control Protocol, Src Port: 51476, Dst Port: 80, Seq: 1, Ack:
  Source Port: 51476
  Destination Port: 80
  [Stream index: 4]
  [TCP Segment Len: 746]
  Sequence Number: 1      (relative sequence number)
  Sequence Number (raw): 3694263361
  [Next Sequence Number: 747      (relative sequence number)]
  Acknowledgment Number: 1      (relative ack number)
  Acknowledgment number (raw): 3481126241
  1000 .... = Header Length: 32 bytes (8)
  ► Flags: 0x018 (PSH, ACK)
  Window: 2070
  0020 f5 0c c9 14 00 50 dc 31 fc 41 cf 7d c5 61 80 18 .....P.1.A}a..
  0030 08 16 1f 48 00 00 01 01 08 0a 2d 1f c7 d4 c4 69 ...H.....i
  0040 1a a8 50 4f 53 54 20 2f 77 69 72 65 73 68 61 72 ..POST / wireshar
  0050 6b 2d 6c 61 62 73 2f 6c 61 62 33 2d 31 2d 72 65 k-labs/l ab3-1-re
  0060 70 6c 79 2e 68 74 6d 20 48 54 54 50 2f 31 2e 31 ply.htm HTTP/1.1
  0070 0d 0a 48 6f 73 74 3a 20 67 61 69 61 2e 63 73 2e ..Host: gaia.cs.
  0080 75 6d 61 73 73 2e 65 64 75 0d 0a 43 6f 6e 6e 65 umass.ed u...Conne
  0090 63 74 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 ction: k eep-aliv
  00a0 65 0d 0a 43 6f 6e 74 65 6e 74 2d 4c 65 6e 67 74 e...Conte nt-Lengt
  00b0 68 3a 20 31 35 32 33 32 31 0d 0a 43 61 63 68 65 h: 15232 1..Cache
  00c0 2d 43 6f 6e 74 72 6f 6c 3a 20 6d 61 78 2d 61 67 -Control : max-ag
  00d0 65 3d 30 0d 0a 4f 72 69 67 69 6e 3a 20 68 74 74 e=0...Ori gin: htt
  00e0 70 3a 2f 2f 67 61 69 61 2e 63 73 2e 75 6d 61 73 p://gaia .cs.umas
  00f0 73 2e 65 64 75 0d 0a 55 70 67 72 61 64 65 2d 49 s.edu..U pgrade-I
  0100 6e 73 65 63 75 72 65 2d 52 65 71 75 65 73 74 73 nsecure- Requests
  0110 3a 20 31 0d 0a 44 4e 54 3a 20 31 0d 0a 43 6f 6e : 1..DNT : 1..Con

```

Figure 9: TCP segment containing the HTTP POST

10. Consider the TCP segments containing the HTTP POST as the last segment in the TCP connection. What are the sequence numbers of the first six segments in the TCP connection? At what time was each segment sent? When was the ACK for each segment received? Given the difference between when each TCP segment was sent, and when its acknowledgement was received, what is the RTT value for each of the six segments? What is the EstimatedRTT value (see Section 3.5.3 in the textbook) after the receipt of each ACK Assume that the value of the EstimatedRTT is equal to the measured RTT for the first segment, and then is computed using the EstimatedRTT equation (Section 3.5.3 in the textbook) for all subsequent segments.

EstimatedRTT values are calculated in a similar way to moving average procedure.

17	1.189911	192.168.1.11	128.119.245.12	TCP	66	51477 - 88 [FIN, ACK] Seq=1 Win=2848 Len=0 TSval=757857488 TSecr=3295239884
18	1.189138	192.168.1.11	128.119.245.12	TCP	78	51478 - 88 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 TSval=757857488 TSecr=0 SACK_PERM=1
19	1.191348	192.168.1.11	128.119.245.12	TCP	812	51476 - 88 [PSH, ACK] Seq=1 Win=2878 Len=44 TSval=757857488 TSecr=3295232448 [TCP segment of a re
20	1.191415	192.168.1.11	128.119.245.12	TCP	1586	51476 - 88 [ACK] Seq=2187 Ack=1 Win=2878 Len=0 TSval=757857488 TSecr=3295232448 [TCP segment of a re
21	1.194410	192.168.1.11	128.119.245.12	TCP	1586	51476 - 88 [ACK] Seq=2187 Ack=1 Win=2878 Len=0 TSval=757857488 TSecr=3295232448 [TCP segment of a re
28	1.345917	192.168.1.11	128.119.245.12	TCP	66	8.00014. 51478 - 88 [ACK] Seq=1 Ack=1 Win=123488 Len=0 TSval=757857641 TSecr=3295241875
30	1.353710	192.168.1.11	128.119.245.12	TCP	1586	51476 - 88 [ACK] Seq=8227 Ack=1 Win=2878 Len=1440 TSval=757857648 TSecr=3295241884 [TCP segment of a re
33	1.358085	192.168.1.11	128.119.245.12	TCP	1586	51476 - 88 [ACK] Seq=5867 Ack=1 Win=2878 Len=1440 TSval=757857652 TSecr=3295241886 [TCP segment of a re
118	1.9052003	192.168.1.11	128.119.245.12	TCP	1512	51476 - 88 [ACK] Seq=5867 Ack=1 Win=2878 Len=1440 TSval=757857652 TSecr=3295241886 [TCP segment of a re

Figure 10: First six packages

3 Part 3 - UDP Experiments

```
➔ ~ nslookup www.vanderbilt.edu
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
www.vanderbilt.edu canonical name = public.elb.vanderbilt.edu.
Name:   public.elb.vanderbilt.edu
Address: 99.83.227.89
Name:   public.elb.vanderbilt.edu
Address: 75.2.77.85
```

Figure 11: NS lookup command for www.vanderbilt.edu

3.1 Questions

11. What display filter did you apply in order to see appropriate packets?

To see the packet for nslookup for www.vanderbilt.edu, dns.qry.name == "www.vanderbilt.edu" filter can be used.

No.	Time	Source	Destination	Protocol	Length	RTT	Info
579	7.863238	192.168.1.11	8.8.8.8	DNS	78		Standard query 0x311d A www.vanderbilt.edu
598	8.182258	8.8.8.8	192.168.1.11	DNS	135		Standard query response 0x311d A www.vanderbilt.edu

Figure 12: Wireshark filter for www.vanderbilt.edu

12. Which application layer and transport layer protocol do nslookup work on? What is the reason that transport layer protocol is chosen?

As shown in figure from Wireshark application, nslookup works on DNS Application layer and UDP Transport layer protocols. UDP is used because it's faster than TCP. TCP has 3-way handshake and congestion control methods that makes it slower compared to connectionless UDP protocol.

No.	Time	Source	Destination	Protocol	Length
579	7.863238	192.168.1.11	8.8.8.8	DNS	
598	8.182258	8.8.8.8	192.168.1.11	DNS	

▶	Frame 598: 135 bytes on wire (1080 bits), 135 bytes captured (1080 bits) on interface en0, id 0
▶	Ethernet II, Src: ASUSTekC_75:8d:08 (d4:5d:64:75:8d:08), Dst: Apple_55:31:be (f0:18:98:55:31:be)
▶	Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.1.11
▼	User Datagram Protocol, Src Port: 53, Dst Port: 51660
	Source Port: 53
	Destination Port: 51660
	Length: 101
	Checksum: 0x8e33 [unverified]
	[Checksum Status: Unverified]
	[Stream index: 8]
▶	[Timestamps]
	UDP payload (93 bytes)

Figure 13: nslookup protocols

13. Can you derive the local DNS server you connected work in iterative or recursive manner? If you can or cannot please provide a detailed explanation. Please also briefly explain the advantages and disadvantages of iterative and recursive approach over each other.

Yes, we can determine that using Wireshark. The flag in Domain Name System states that the local DNS that is for "www.vanderbilt.edu" works in recursive manner as shown in the figure.

dns.qry.name == "www.vanderbilt.edu"							
No.	Time	Source	Destination	Protocol	Length	RTT	Info
579	7.863238	192.168.1.11	8.8.8.8	DNS	78		Standard query 0x311d A www.vanderbilt.edu
598	8.182258	8.8.8.8	192.168.1.11	DNS	135		Standard query response 0x311d A www.vanderbilt.edu

▶	Frame 598: 135 bytes on wire (1080 bits), 135 bytes captured (1080 bits) on interface en0, id 0
▶	Ethernet II, Src: ASUSTekC_75:8d:08 (d4:5d:64:75:8d:08), Dst: Apple_55:31:be (f0:18:98:55:31:be)
▶	Internet Protocol Version 4, Src: 8.8.8.8, Dst: 192.168.1.11
▶	User Datagram Protocol, Src Port: 53, Dst Port: 51660
▼	Domain Name System (response)
	Transaction ID: 0x311d
▼	Flags: 0x8180 Standard query response, No error
	1... .. = Response: Message is a response
	.000 0... .. = Opcode: Standard query (0)
0... .. = Authoritative: Server is not an authority for domain
0... .. = Truncated: Message is not truncated
01... .. = Recursion desired: Do query recursively
1... .. = Recursion available: Server can do recursive queries
0... .. = Z: reserved (0)
0... .. = Answer authenticated: Answer/authority portion was not authenticated by the server
0... .. = Non-authenticated data: Unacceptable
0000 = Reply code: No error (0)
	Questions: 1
	Answer RRs: 3
	Authority RRs: 0
	Additional RRs: 0
▼	Queries
▶	www.vanderbilt.edu: type A, class IN
▶	Answers
	[Request In: 579]
	[Time: 0.319020000 seconds]

Figure 14: DNS Flags

14. What are the header lengths of application layer protocol and transport layer protocol

that nslookup works on?

The application layer, DNS, has 12 byte length header which includes transaction ID (2 bytes), flags (2 bytes), number of queries (questions, 2 bytes), number of answers (2 bytes), number of authoritative records (2 bytes), number of additional records in packet (bytes) fields.

The transport layer, UDP, has 8 byte length header which includes source(2 bytes) and destination(2 bytes) ports, length(2 bytes) and checksum(2 bytes) fields.

15. How many checksums does an UDP segment have in the checksum field? Why?

UDP itself has one 2 byte checksum for checking payload. There is also another 2 byte checksum for header for checking the received datagram.

No.	Time	Source	Destination	Protocol	Length
579	7.863238	192.168.1.11	8.8.8.8	DNS	78
598	8.182258	8.8.8.8	192.168.1.11	DNS	135

```

▶ Ethernet II, Src: Apple_55:31:be (f0:18:98:55:31:be), Dst: ASUSTekC_75:8d:08 (d4:5d:6
▼ Internet Protocol Version 4, Src: 192.168.1.11, Dst: 8.8.8.8
    0100 .... = Version: 4
    .... 0101 = Header Length: 20 bytes (5)
    ▶ Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
      Total Length: 64
      Identification: 0x59a2 (22946)
    ▶ Flags: 0x00
      Fragment Offset: 0
      Time to Live: 64
      Protocol: UDP (17)
      Header Checksum: 0x4f48 [validation disabled]
      [Header checksum status: Unverified]
      Source Address: 192.168.1.11
      Destination Address: 8.8.8.8
    ▼ User Datagram Protocol, Src Port: 51660, Dst Port: 53
      Source Port: 51660
      Destination Port: 53
      Length: 44
      Checksum: 0xbc41 [unverified]
      [Checksum Status: Unverified]
      [Stream index: 8]
      ▶ [Timestamps]
      UDP payload (36 bytes)
    ▼ Domain Name System (query)
  
```

Figure 15: UDP DNS segment fields

4 Part 4 - Stop-and-Wait ARQ Protocol

4.1 Part 4.b - Analyzing the Implementation

16. How many retransmissions have occurred from the sender to the receiver? Explain.

According to Wireshark, there were 50 packages send from sender(port 8000) to receiver(port 8001), 32 packages(ACKs) from receiver to sender. This means there is at least $50-32=18$ re-transmissions occurred during this process.

17. How long did it take to transfer all the packets? Does this result make sense? Why or why not?

Reducing the timeout in sender side reducing the transfer time significantly, however, it might cause unnecessary re-transmission.

With 500ms sender side timeout, the time for total transmission took roughly 24s. This doesn't make sense and it indicates the packet loss. Because 18 re-transmission means it should roughly take 9s assuming that the local transmission time for successful packages are in milliseconds. The results from Wireshark indicates that there were around 48 timeouts in sender side. These losses not visible through Wireshark due to underlying implementation.

ip.dst == 192.168.1.11								
No.	Time	Source	Destination	Protocol	Length	RTT	Info	
25	0.800584	192.168.1.11	192.168.1.11	UDP	187		8001 → 8000 Len=155	
26	0.803383	192.168.1.11	192.168.1.11	UDP	182		8000 → 8001 Len=150	

Figure 16: Stop-and-Wait ARQ Protocol Analysis start

112	25.963037	192.168.1.11	192.168.1.11	UDP	182		8000 → 8001 Len=150	
113	25.010894	192.168.1.11	192.168.1.11	UDP	185		8001 → 8000 Len=153	
114	25.011461	192.168.1.11	192.168.1.11	UDP	182		8000 → 8001 Len=150	

Figure 17: Stop-and-Wait ARQ Protocol Analysis end

```

public Packet[] receiveWithARQ() {

    List<Packet> packets = new ArrayList<>();
    int lastSequence = -1;
    while(true) {
        Packet p = receivePacket(15000);
        // Stop receiving either on timeout or when we received the last
        // message packet.
        if(p.timedOut) break;

        sendAck(p.sequenceNumber % 2, p.lastPacket);

        if(lastSequence == p.sequenceNumber) continue;
        lastSequence = p.sequenceNumber;
        packets.add(p);

        if(p.lastPacket) break;
    }
    Packet[] pks = new Packet[packets.size()];
    for(int i = 0; i < pks.length; i++) pks[i] = packets.get(i);
    return pks;
}

```

```
public void sendWithARQ(Packet[] packets) {
    int currentPacket = 0;
    while (currentPacket < packets.length) {
        sendMsgPacket(currentPacket % 2, currentPacket == packets.length-1,
            packets[currentPacket]);
        Packet p = receivePacket(500);
        if(!p.timedOut) {
            currentPacket++;
        }
    }
}
```
