

John Carlo Sumabat
2018-03037
Lab 1

CS 145 Project Documentation

Parameter-Adaptive Reliable UDP-based Protocol

1 Implementation Level

The implementation level is **level 3**. Thus, the program is able to send Intent Message (to the receiver or test server), receive the Accept Message (from the receiver or test server), and send the payload completely in 120 seconds.

2 Documentation Video

The documentation video can be accessed by clicking [this link](#).

3 Network Setup

For trace file generation, I used TShark on an Amazon EC2 t2.micro instance. The network interface used was `eth0`. Trace analysis was done on a Windows machine.

4 Introduction

4.1 Implementation Overview

On a high level, the program works as follows:

- From a terminal, the name of the program is entered, followed by 5 optional arguments which define the program constants. For each argument that is not specified, a default value is used. Included in these constants are the IP address and port of the server to transact with.
- The program creates an intent message and sends it to the specified server address. Upon success, it receives a transaction ID, indicating that a transaction has begun. In case a previous transaction has not yet terminated, it retries again in 10 seconds.
- Upon receiving a transaction ID, the program begins parameter estimation to determine the maximum payload length that the server can process.
- Lastly, the program sends the remaining payload in chunks equal to the maximum payload length. It sends the data in a stop-and-wait manner, such that an ACK from the server is required for a succeeding packet to be sent. If no ACK is received for a while, the program terminates.

4.2 Parameter Estimation Algorithm

The parameter that the program estimates is the maximum payload length that the server will acknowledge. It achieves this as follows:

- The program initializes the maximum payload length as one eighth the size of the original payload.
- A packet is sent to the server containing the payload with the initial length.
- If there is no ACK received within half a second, the maximum payload length is reduced by 5%, and a new packet is sent.
- The process of decreasing the payload length repeats until an ACK is received or the payload length is 1 byte.
- Once an ACK is received, its checksum is used to determine which packet was acknowledged. From the said packet, the final maximum payload length is obtained.

This is explored in more detail in [Code Block 8](#) in [Section 5.4](#).

5 Implementation Details

5.1 Python Standard Library Facilities Used

Code Block 1 shows the standard library modules, functions, and classes that the program imports. Their uses are shown in **Table 1**.

Facility	Purpose
<code>argparse.ArgumentParser</code>	Class responsible for handling logic of parsing command line arguments.
<code>hashlib.md5</code>	Function to compute hash value of packet
<code>time.sleep</code>	Function to make thread pause execution
<code>socket</code>	Module that provides low-level networking interface

Table 1. Descriptions of imported facilities

Code Block 1 <code>import</code> statements
<pre>1 from argparse import ArgumentParser 2 from hashlib import md5 3 from time import sleep 4 import socket</pre>

5.2 Program Constants

Code Block 2 shows the program constants¹. Default values are used unless a corresponding command line argument is supplied. Descriptions for the constants are shown in **Table 2**.

Line	Description
7, 11	Client address; used to bind a UDP socket for listening
9, 10	Server address; used for packet sending
8	Pathname of payload to send
12	Unique ID assigned to student

Table 2. Descriptions of program constants

5.3 Helper Functions

To help with testing and repetitive tasks, some helper functions were created.

- The function `announce` (**Code Block 3**) accepts a string and prints it with a border. It is used to differentiate between different “stages” of the program.
- Upon receiving an ACK, `parse_ack` (**Code Block 4**) parses the packet and returns its contents via a dictionary.
- The function `make_msg` (**Code Block 5**) constructs a data packet given a unique ID, sequence number, transaction ID, `LAST` flag, and payload. It returns this message as a string.
- Every time a packet is sent, the function `compute_checksum` (**Code Block 6**) is called to compute the hash value of the packet. This is used to determine the client packet that the server acknowledges.

¹ Although there are no “official” constants in Python, these variables are functionally the same since they never get modified after being initialized either by default or via command line arguments.

Code Block 2 Program constants

```
7 CLIENT_HOSTNAME = socket.gethostname(socket.gethostname())
8 FILE = "2b97c5ee.txt" # file to be sent by client
9 SERVER_HOSTNAME = "10.0.7.141" # IP address of receiver
10 UDP_PORT_SEND = 9000 # sending port for UDP
11 UDP_PORT_LISTEN = 6750 # listening port for UDP
12 ID = "2b97c5ee" # unique student ID
```

Code Block 3 announce function

```
15 def announce(msg):
16     """
17     Print a message surrounded by a border for emphasis
18
19     :param msg: message to be printed
20     """
21     print("\n" + "=" * len(msg))
22     print(msg)
23     print("=" * len(msg))
```

Code Block 4 parse_ack function

```
26 def parse_ack(ack):
27     """
28     Extract the contents of an ack packet
29
30     :param ack: ack packet received
31     :returns: contents of packet stored in dictionary
32     """
33     sn = ack[3:10]
34     txn = ack[13:20]
35     chksum = ack[23:]
36
37     print(f"Received ack: {ack}")
38
39     return {"sn": sn, "txn": txn, "chksum": chksum}
```

Code Block 5 make_msg function

```
42 def make_msg(idd, sn, txn, last, payload):
43     """
44     Construct a data packet
45
46     :param idd: unique student ID
47     :param sn: sequence number
48     :param txn: transaction ID
49     :param last: denotes whether the packet is the final packet
50     :param payload: data to be sent
51     :returns: formatted packet
52     """
53     return f"ID{idd}SN{sn:>07d}TXN{txn:>07d}LAST{last}{payload}"
```

Code Block 6 compute_checksum function

```
56 def compute_checksum(packet):
57     """
58     Compute the hash value of a packet
59
60     :param packet: packet whose hash is to be computed
61     :returns: 128-bit hash value
62     """
63     return md5(packet.encode("utf-8")).hexdigest()
```

5.4 Main Functions

When the client wishes to send a payload, the program performs this in three stages:

1. First, `begin_transaction` (Code Block 7) is called. The function begins in line 166 by creating a socket named `udp_socket` via `socket.socket`. The program notifies the user that it is sending an intent message by calling `announce`. Using the constants `CLIENT_HOSTNAME` and `UDP_PORT_LISTEN`, the program calls `socket.bind` to bind the socket to an address for receiving packets in line 169. In line 170, the intent message is constructed using the constant `ID`.

In lines 174–184, the function attempts to retrieve a transaction ID by sending the intent message to the server (whose address is stored in the constants `SERVER_HOSTNAME` and `UDP_PORT_SEND`). If there is a preexisting alive transaction, the server specifies this, and the client will retry in 10 seconds. Otherwise, the server sends a transaction ID, and the transaction is commenced. Afterwards, the transaction ID is printed in line 186. Finally, the function calls `get_max_payload_size` with the UDP socket and transaction ID as parameters.

Code Block 7 begin_transaction function

```
162 def begin_transaction():
163     """
164     Send an intent message and commence a transaction
165     """
166     with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as udp_socket:
167         announce("SENDING INTENT MESSAGE")
168
169         udp_socket.bind((CLIENT_HOSTNAME, UDP_PORT_LISTEN))
170         intent_msg = f"ID{ID}"
171
172         print(f"Intent message: {intent_msg}")
173
174         while True:
175             udp_socket.sendto(
176                 intent_msg.encode(), (SERVER_HOSTNAME, UDP_PORT_SEND)
177             )
178             data, addr = udp_socket.recvfrom(2048)
179
180             if data.decode() == "Existing alive transaction":
181                 print("Existing alive transaction found. Retrying in 10s.")
182                 sleep(10)
183             else:
184                 break
185
186         print(f"Transaction ID: {data.decode()}")
187
188         get_max_payload_size(udp_socket, txn_number=data.decode())
```

2. Upon receiving a transaction ID, `get_max_payload_size` (Code Block 8) performs parameter estimation.

For lines 121–125: first, the function sets the sequence number and `LAST` flag to `0`, and saves the transaction ID. It informs the user via `announce` that it will begin estimating the maximum payload size that the server will ACK.

For lines 127–132: the function opens the file that contains the payload, which is specified in the constant `FILE`. An empty dictionary `sent_packets` is initialized which would contain the checksum of each packet sent and its corresponding payload length. The initial size of the payload is set to $1/8^{\text{th}}$ the size of the file and is assigned to the variable `payload_size`. The timeout duration for receiving ACKs is set to 0.5 seconds.

For lines 134–149 and 154–155: the client constructs a packet containing the contents of `FILE` from index `0` to `payload_size`. The checksum of this packet is obtained via `compute_checksum`. The client attempts to send this packet and waits 0.5 seconds for an ACK. If no ACK is received, then either the server dropped the packet or it accepted the packet but is currently processing it. In any case, the checksum and corresponding payload length of the packet are saved in `sent_packets`. The value of `payload_size` is then decreased by 5%, and the client tries again. However, if the value of `payload_size` is `1`, then the client will wait for an ACK indefinitely since the payload length cannot get any smaller.

For lines 150–153: the `while` loop exits when the client receives **one** ACK from the server. The checksum of this ACK is used to find the corresponding payload size in `sent_packets`. The variable `payload_size` is then updated to represent the maximum payload length.

After parameter estimation, `send_payload` is called (line 159).

3. Finally, the remaining payload is sent via `send_payload` (Code Block 9).

For lines 74–78: the function begins by setting the `LAST` flag to `0` and the transaction ID as specified in the parameters. Since the client has received one ACK at this point, the sequence number `SN` is set to `1`. It calls `announce` to inform the user that the remaining payload will be transmitted to the server.

For lines 80–86: the function also takes in a parameter `offset` which should be equal to the maximum payload size. This variable represents the location in the file where a payload begins. Before transmission, `FILE` is reopened and the timeout duration is set to 30 seconds. Since `offset` is updated for every ACKed packet, `payload_size` is set to the initial offset value.

For lines 88–100: the client sends the remainder of the payload via a `while` loop. For every iteration, the client constructs a packet containing the contents of `FILE`. The content of the payload portion of the packet depends on the amount of remaining data to transmit. In case the contents of `FILE` are all exhausted, `LAST` is set to `1`. For each packet constructed, the checksum is also computed.

For lines 103–111: the client waits for an ACK for each packet sent. If an ACK is received within 30 seconds, then the sequence number is incremented and `offset` is updated. Otherwise, the server has terminated the transaction. In this case (or when all of `FILE` is successfully sent), the program terminates. The user is informed of this via a final `announce` call.

Code Block 8 get_max_payload_size function

```
114 def get_max_payload_size(udp_socket, txn_number=0):
115     """
116     Determine maximum payload size
117
118     :param udp_socket: active socket
119     :param txn_number: transaction ID
120     """
121     SN = 0
122     TXN = int(txn_number)
123     LAST = 0
124
125     announce("DETERMINING MAXIMUM PAYLOAD SIZE")
126
127     with open(FILE) as file:
128         f = file.read()
129
130         sent_packets = {}
131         payload_size = int(len(f) * 0.125) # initial payload size: 1/8 of file
132         udp_socket.settimeout(0.5)         # waiting time for ack: 0.5 seconds
133
134         while True:
135             PAYLOAD = f[0:payload_size]
136
137             msg = make_msg(ID, SN, TXN, LAST, PAYLOAD)
138             chksum = compute_checksum(msg)
139             udp_socket.sendto(msg.encode(), (SERVER_HOSTNAME, UDP_PORT_SEND))
140
141             sent_packets[chksum] = payload_size
142
143             print(f"\nSent message: {msg}")
144             print(f"Checksum: {chksum}")
145
146             # decrease payload size by 5% for every timeout
147             try:
148                 if payload_size == 1:
149                     udp_socket.settimeout(None)
150                     data, addr = udp_socket.recvfrom(2048)
151                     ack = parse_ack(data.decode())
152                     payload_size = sent_packets[ack["chksum"]]
153                     break
154             except socket.timeout:
155                 payload_size = int(payload_size * 0.95)
156
157             print(f"\nMax payload size = {payload_size} characters")
158
159     send_payload(udp_socket, txn_number=txn_number, offset=payload_size)
```

Code Block 9 send_payload function

```
66 def send_payload(udp_socket, txn_number=0, offset=0):
67     """
68     Send the entire payload
69
70     :param udp_socket: active socket
71     :param txn_number: transaction ID
72     :param offset: which section of the file to begin sending
73     """
74     SN = 1
75     TXN = int(txn_number)
76     LAST = 0
77
78     announce("SENDING PAYLOAD")
79
80     with open(FILE) as file:
81         f = file.read()
82         maxlen = len(f)
83
84         payload_size = offset
85
86         udp_socket.settimeout(30)
87
88         while offset < maxlen:
89             if offset + payload_size < maxlen:
90                 PAYLOAD = f[offset : offset + payload_size]
91             else:
92                 PAYLOAD = f[offset:]
93                 LAST = 1
94
95             msg = make_msg(ID, SN, TXN, LAST, PAYLOAD)
96             checksum = compute_checksum(msg)
97             udp_socket.sendto(msg.encode(), (SERVER_HOSTNAME, UDP_PORT_SEND))
98
99             print(f"\nSent message: {msg} ({offset} / {len(f)})")
100            print(f"Checksum: {' '*28}{checksum}")
101
102            # end transaction if no longer receiving acks (120 seconds exceeded)
103            try:
104                data, addr = udp_socket.recvfrom(2048)
105                ack = parse_ack(data.decode())
106                SN += 1
107                offset += payload_size
108            except socket.timeout:
109                break
110
111            announce("TRANSACTION TERMINATED")
```

5.5 Program Entry Point

Code Block 10 shows where program execution begins. Upon launch, an `ArgumentParser` object is set up in lines 193–199 to handle command line arguments. The constants discussed in Section 5.2 are initialized in lines 204–208 and are revealed to the user in lines 210–216. Afterwards, `begin_transaction` is called to attempt initiating a transaction with the server.

Code Block 10 Program entry point

```
191 if __name__ == "__main__":
192     # command line argument parser
193     parser = ArgumentParser()
194
195     parser.add_argument("-f", default=FILE)
196     parser.add_argument("-a", default=SERVER_HOSTNAME)
197     parser.add_argument("-s", type=int, default=UDP_PORT_SEND)
198     parser.add_argument("-c", type=int, default=UDP_PORT_LISTEN)
199     parser.add_argument("-i", default=ID)
200
201     args = parser.parse_args()
202
203     # initialize program constants
204     FILE = args.f
205     SERVER_HOSTNAME = args.a
206     UDP_PORT_SEND = args.s
207     UDP_PORT_LISTEN = args.c
208     ID = args.i
209
210     announce("TRANSACTION DETAILS")
211
212     print(f"File to send: {FILE}")
213     print(f"IP address of server: {SERVER_HOSTNAME}")
214     print(f"Port used by server: {UDP_PORT_SEND}")
215     print(f"Port used by client: {UDP_PORT_LISTEN}")
216     print(f"Unique ID: {ID}")
217
218     begin_transaction()
```

6 Testing the Program

To test whether the implementation works, 5 transactions were initiated with the server. Table 3 shows the command line arguments supplied when testing the program.

Flag	Supplied Option
-f	2b97c5ee.txt
-a	10.0.7.141
-s	9000
-c	6750
-i	2b97c5ee

Table 3. Command line arguments used for testing

For each test, a new payload is downloaded. Figure 1 highlights each of the 5 transactions and their respective IDs. As expected of a level 3 implementation, all tests have successfully sent the data in at most 120 seconds and have “Frequency Used: 1”.

Transaction ID	Time Started	Duration	Result	Frequency Used
0007432	2022-06-02 21:53:00.177263	94.744	Successfully sent data	1
0007430	2022-06-02 21:51:57.258538	90.403	Successfully sent data	2
0007428	2022-06-02 21:51:41.621591	87.898	Successfully sent data	1
0007427	2022-06-02 21:51:16.266963	79.468	Successfully sent data	1
0007425	2022-06-02 21:49:41.348705	90.498	Successfully sent data	1
0007424	2022-06-02 21:49:32.537533	93.660	Successfully sent data	1
0007423	2022-06-02 21:48:59.954708	91.327	Successfully sent data	1
0007421	2022-06-02 21:47:11.684122	96.897	Successfully sent data	1
0007420	2022-06-02 21:47:01.351947	97.777	Successfully sent data	1
0007419	2022-06-02 21:46:52.795404	83.455	Successfully sent data	1
0007418	2022-06-02 21:46:18.957073	91.300	Successfully sent data	1
0007416	2022-06-02 21:44:53.916823	108.699	Successfully sent data	2
0007415	2022-06-02 21:44:31.320561	100.047	Successfully sent data	1
0007414	2022-06-02 21:44:25.878103	85.656	Successfully sent data	1
0007408	2022-06-02 21:41:13.138273	82.433	Successfully sent data	1

Figure 1. Screenshot of Transaction Generation/Results Server

To explore the details of each transaction, we would show the following via screenshots of the terminal and trace files:

- The intent message sent and the transaction ID received
- The first packet to be acknowledged by the server and the corresponding ACK
- The last packet to be sent by the client and the last ACK sent by the server

For brevity, some screenshots show a combined view of the program’s terminal output and the corresponding trace file, as demonstrated by Figure 2. Additionally, only UDP packets were captured, since all packets sent by the client and server are in UDP.

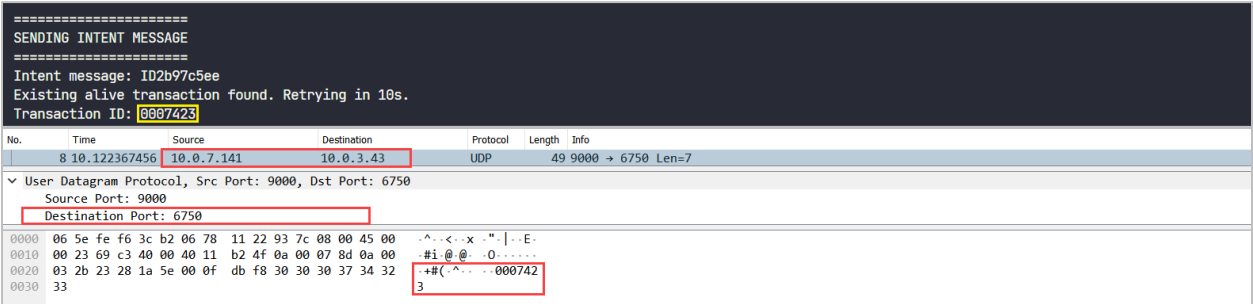


Figure 2. Sample of combined view of terminal output and Wireshark

6.1 First Test (Transaction ID: 0007408)

For this test, the corresponding trace file is test1_transaction0007408.pcap.

Figure 3 shows packet 5 which contains the intent message “ID2b97c5ee” sent to the IP address of the server via the supplied port 9000. Figure 4 shows through the terminal and packet 6 that the transaction ID sent by the server is 0007408. Note that the source and destination IP addresses in this packet are reversed, and that the server is addressing the supplied port 6750.

As shown in packet 29 via Figure 5, the first ACK that the server sent corresponds to the packet with a payload size of 77 bytes. Included in the ACK is the checksum of the said packet. This checksum corresponds to packet 18 in Figure 6, as shown by the yellow rectangle.

Figure 7 shows the last packet sent by the client (with LAST flag set to 1) and the corresponding last ACK sent by the server. Since the checksum of the packet and the ACK are equivalent, this is the correct pair of packets. Figure 8 shows via packet 56 that the last client packet was indeed sent to the correct IP address (10.0.7.141) and port (9000). In the same way, Figure 9 shows through packet 57 that the content of the last ACK matches with what is shown in the terminal and that the IP address and port (6750) belong to the client, as expected.

No.	Time	Source	Destination	Protocol	Length	Info
	5.006476202	10.0.3.43	10.0.7.141	UDP	52	6750 → 9000 Len=10
User Datagram Protocol, Src Port: 6750, Dst Port: 9000						
Source Port: 6750						
Destination Port: 9000						
0000	06 78 11 22 93 7c 06 5e fe f6 3c b2 08 00 45 00	.x.. ^ ^<...E.				
0010	00 26 e7 10 40 00 40 11 34 ff 0a 00 03 2b 0a 00	.8.@.@.4...+				
0020	07 8d 1a 5e 23 28 00 12 1e db 49 44 32 62 39 37	...^#(...ID2b97				
0030	63 35 65 65	c5ee				

Figure 3. Intent message for the first transaction

=====						
SENDING INTENT MESSAGE						
=====						
Intent message: ID2b97c5ee						
Transaction ID: 0007408						
=====						
No.	Time	Source	Destination	Protocol	Length	Info
	6.025532108	10.0.7.141	10.0.3.43	UDP	49	9000 → 6750 Len=7
User Datagram Protocol, Src Port: 9000, Dst Port: 6750						
Source Port: 9000						
Destination Port: 6750						
0000	06 5e fe f6 3c b2 06 78 11 22 93 7c 08 00 45 00	.^<...x.. ^...E.				
0010	00 23 80 b0 40 00 40 11 9b 62 0a 00 07 8d 0a 00	#..@..-b.....				
0020	03 2b 23 28 1a 5e 00 0f d6 fa 30 30 30 37 34 30	..+#(^...000740				
0030	38	8				

Figure 4. Transaction ID for the first transaction

Received ack: ACK0000000TXN0007408MD5af9e6b6eacaa5c9996d9d78d1f517eb4						
Max payload size = 77 characters						
=====						
No.	Time	Source	Destination	Protocol	Length	Info
	29.10.555804048	10.0.7.141	10.0.3.43	UDP	97	9000 → 6750 Len=55
User Datagram Protocol, Src Port: 9000, Dst Port: 6750						
Source Port: 9000						
Destination Port: 6750						
0000	06 5e fe f6 3c b2 06 78 11 22 93 7c 08 00 45 00	.^<...x.. ^...E.				
0010	00 53 88 a0 40 00 40 11 93 42 0a 00 07 8d 0a 00	S..@..B.....				
0020	03 2b 23 28 1a 5e 00 3f 99 51 41 43 4b 30 30 30	..+#(^.? QACK000				
0030	30 30 30 30 54 58 4e 30 30 30 37 34 30 38 4d 44	0000TXN0 007408MD				
0040	35 61 66 39 65 36 62 36 65 61 63 61 61 35 63 39	Saf9e6b6 eacaa5c9				
0050	39 39 36 64 39 64 37 38 64 31 66 35 31 37 65 62	996d9d78 d1f517eb				
0060	34	4				

Figure 5. First acknowledgement received for the first transaction

Sent message: ID2b97c5eeSN0000000TXN0007408LAST0Ni0ErNtScBTLBpCcFrHqLhaLfSyFbWbNbhtySnyiCbPwCKGkvxeDHfrvRvxRKvULxHBFGEIPg						
Checksum: af9e6b6eacaa5c9996d9d78d1f517eb4						
=====						
No.	Time	Source	Destination	Protocol	Length	Info
	18.5.535811914	10.0.3.43	10.0.7.141	UDP	153	6750 → 9000 Len=111
User Datagram Protocol, Src Port: 6750, Dst Port: 9000						
Source Port: 6750						
Destination Port: 9000						
0000	06 78 11 22 93 7c 06 5e fe f6 3c b2 08 00 45 00	.x.. ^ ^<...E.				
0010	00 8b e9 ac 40 00 40 11 31 fe 0a 00 03 2b 0a 00	...@.@.1...+				
0020	07 8d 1a 5e 23 28 00 77 1f 40 49 44 32 62 39 37	...^#(^w @ID2b97				
0030	63 35 65 65 53 4e 30 30 30 30 30 30 54 58 4e	c5eeSN00 00000TXN				
0040	30 30 30 37 34 30 38 4c 41 53 54 30 4e 69 49 4f	0007408L AST0Ni0				
0050	45 72 52 6e 54 73 63 42 54 4c 42 70 43 63 46 52	ErRnTscB TLBpCcFr				
0060	65 48 71 6c 68 61 4c 66 53 79 46 62 57 42 6e 42	eHqIhaLf SyFbWbNb				
0070	68 70 74 79 53 6e 79 69 43 62 50 77 43 4b 47 6b	htySnyiC bPwCKGk				
0080	76 78 65 44 48 66 72 76 52 76 78 52 4b 76 55 6c	vxexDHfrv RvxRKvU				
0090	78 48 42 46 47 45 49 50 67	xHBFGEIP g				

Figure 6. First packet that was acknowledged in the first transaction

Sent message: ID2b97c5eeSN0000014TXN0007408LAST1EUTHMDpJAIPYXBtpnC6dGjKEFXwzyVpdLKWbCNkBrkQ0uaUD6NdWdn6RoeZihQasYhTagYiDhKDD (1078 / 1154)						
Checksum: 2ce72548fcc94fb3e394dae4878bc943						
Received ack: ACK0000014TXN0007408MD52ce72548fcc94fb3e394dae4878bc943						
=====						
TRANSACTION TERMINATED						
=====						

Figure 7. Last packet sent and last ACK received in the first transaction

No.	Time	Source	Destination	Protocol	Length	Info
	56.76.135602373	10.0.3.43	10.0.7.141	UDP	152	6750 → 9000 Len=110
User Datagram Protocol, Src Port: 6750, Dst Port: 9000						
Source Port: 6750						
Destination Port: 9000						
0000	06 78 11 22 93 7c 06 5e fe f6 3c b2 08 00 45 00	.x.. ^ ^<...E.				
0010	00 8a 0d f2 40 00 40 11 0d ba 0a 00 03 2b 0a 00	...@.@.1...+				
0020	07 8d 1a 5e 23 28 00 76 1f 3f 49 44 32 62 39 37	...^#(^v ?ID2b97				
0030	63 35 65 65 53 4e 30 30 30 30 31 34 54 58 4e	c5eeSN00 00014TXN				
0040	30 30 30 37 34 30 38 4c 41 53 54 31 45 55 54 48	0007408L AST1EUTH				
0050	4d 44 70 4a 41 49 50 59 58 42 74 70 72 43 47 64	MDpJAIPY XBtpnCgD				
0060	47 6a 4b 45 66 58 77 7a 79 56 70 64 4c 4b 57 62	GjKEFXwz yVpdLKWb				
0070	43 4e 6b 42 72 6b 51 4f 75 61 55 44 47 4e 64 57	CNkBrkQ0 uaUDGNdW				
0080	64 6e 47 52 6f 65 5a 69 68 51 61 73 59 68 54 61	dnGRoeZi hQasYhTa				
0090	67 59 69 44 68 4b 44 44	gYiDhKDD				

Figure 8. Wireshark view of last packet sent in first transaction

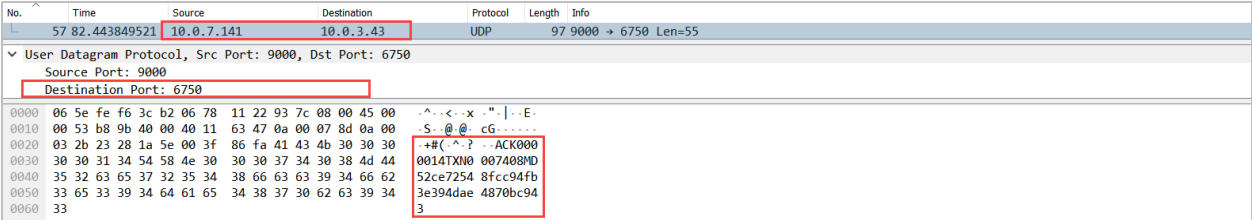


Figure 9. Wireshark view of last ACK received in first transaction

6.2 Second Test (Transaction ID: 0007414)

For this test, the corresponding trace file is test2_transaction0007414.pcap.

Figure 10 shows packet 5 which contains the intent message “ID2b97c5ee” sent to the IP address of the server via the supplied port 9000. Figure 11 shows through the terminal and packet 6 that the transaction ID sent by the server is 0007414. Note that the source and destination IP addresses in this packet are reversed, and that the server is addressing the supplied port 6750.

As shown in packet 28 via Figure 12, the first ACK that the server sent corresponds to the packet with a payload size of 94 bytes. Included in the ACK is the checksum of the said packet. This checksum corresponds to packet 15 in Figure 13, as shown by the yellow rectangle.

Figure 14 shows the last packet sent by the client (with LAST flag set to 1) and the corresponding last ACK sent by the server. Since the checksum of the packet and the ACK are equivalent, this is the correct pair of packets. Figure 15 shows via packet 51 that the last client packet was indeed sent to the correct IP address (10.0.7.141) and port (9000). In the same way, Figure 16 shows through packet 52 that the content of the last ACK matches with what is shown in the terminal and that the IP address and port (6750) belong to the client, as expected.

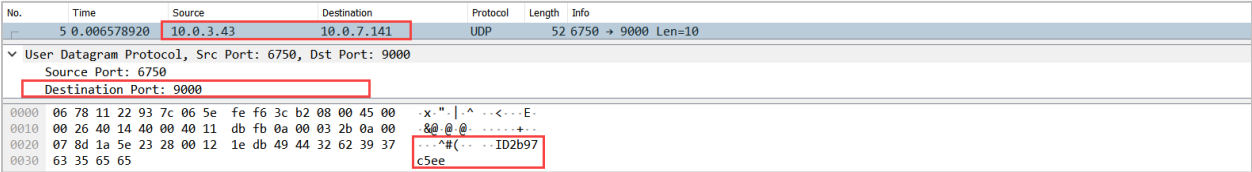


Figure 10. Intent message for the second transaction

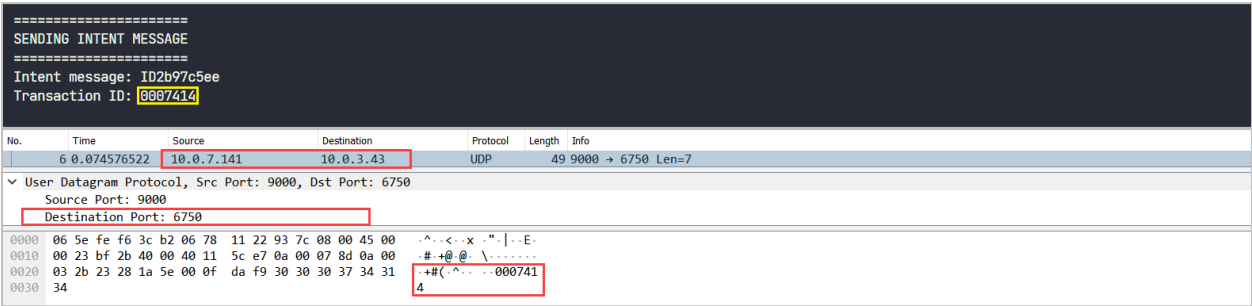


Figure 11. Transaction ID for the second transaction

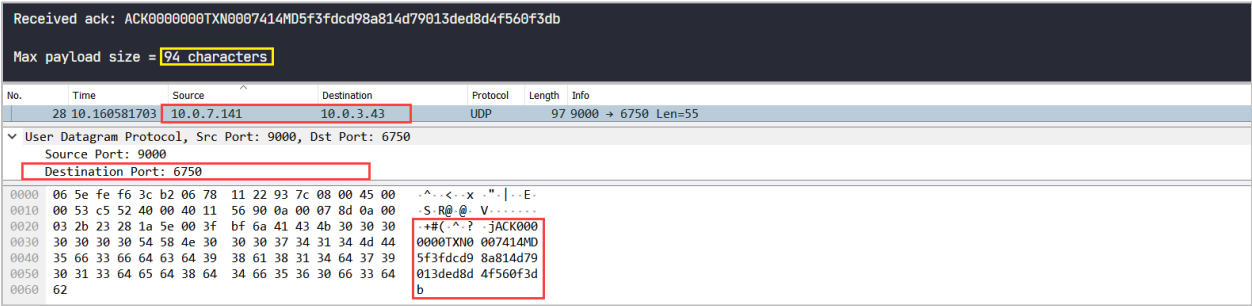


Figure 12. First acknowledgement received for the second transaction

Sent message: ID2b97c5ee5N0008080TXN0807414LAST0HEWJjsD5mxLoaKLnhpAFckOunzLKmnINv0DhnhSyJXhZyXtFIgHOCmyGmdFjWgJxvSrehznkGUBvayYBDDzrwaNeDvkCFE
Checksum: f3fcdcd98a814d79013ded8d4f560f3db

No.	Time	Source	Destination	Protocol	Length	Info
15	4.083277072	10.0.3.43	10.0.7.141	UDP	170	6750 → 9000 Len=128

▼ User Datagram Protocol, Src Port: 6750, Dst Port: 9000

Source Port: 6750

Destination Port: 9000

```
0000 06 78 11 22 93 7c 06 5e fe f6 3c b2 08 00 45 00  .x..|..^...-...E-
0010 00 9c 42 40 40 00 40 11 d9 59 0a 00 03 2b 0a 00  @...@...Y...
0020 07 8d 1a 5e 23 28 08 68 1f 51 49 44 32 62 39 37  ....4...QID2b97
0030 63 35 65 53 4e 30 39 30 30 30 30 39 54 58 4e  c5ee5N00 000007X0I
0040 30 30 37 37 34 31 34 4c 41 53 54 30 48 45 77 4a  00074141 AST0HEWJ
0050 6a 73 44 53 6d 78 6c 6f 61 4b 4c 6e 68 70 41 4e  jsD5mxLo aKLnhpAF
0060 43 6b 4f 75 6e 7a 6c 4b 6d 6e 49 4e 76 4f 4a 68  CkOunzLK mnINv0Dh
0070 6e 68 79 53 79 4a 58 68 5a 79 58 74 66 49 67 48  nhSyJXh ZyXtFIgH
0080 4f 43 6d 79 47 6d 64 46 6a 57 67 6a 78 76 53 72  OCmyGmdF jWgJxvSr
0090 65 68 7a 6e 6b 47 55 42 76 61 59 42 44 7a 72  ehznkGUB vaYBDDzr
00a0 77 61 4e 65 44 76 6b 43 46 45  waNeDvkCFE
```

Figure 13. First packet that was acknowledged in the second transaction

```

Sent message: ID2b97c5eeSN0000012TXN0007414LAST1asaYEhxrNkxYZopCR6rZCsqBUkGpzEJautcVyJLEkcpoq (1128 / 1174)
Checksum: 2fbaa66d87a828a9ba43a07feb9846f7
Received ack: ACK0000012TXN0007414MD52fbaa66d87a828a9ba43a07feb9846f7

=====
TRANSACTION TERMINATED
TRANSACTION TERMINATED

```

Figure 14. Last packet sent and last ACK received in the second transaction

No.	Time	Source	Destination	Protocol	Length	Info
51	79.291179823	10.0.3.43	10.0.7.141	UDP	122	6750 → 9000 Len=80
User Datagram Protocol, Src Port: 6750, Dst Port: 9000 Source Port: 6750 Destination Port: 9000						
0000	06 78 11 22 93 7c 06 5e	fe f6 3c b2 08 00 45 00	.x." .^...E-			
0010	00 6c 5c 92 40 00 40 11	bf 37 0a 00 03 2b 0a 00	1\@.@-7...+			
0020	07 8d 1a 5e 23 28 00 58	1f 21 49 44 32 62 39 37	...#(X-IID2b97			
0030	63 35 65 65 53 4e 30 30	30 30 30 31 32 54 58 4e	cSeeSn00 00012TX0			
0040	30 30 30 37 34 31 34 4c	41 53 54 31 61 73 61 59	0007414L ASt1asaY			
0050	45 68 78 72 4e 6b 78 59	5a 6f 70 63 52 47 72 5a	EhxrNkxY ZopcRgrZ			
0060	43 53 51 62 55 6b 47 70	7a 45 4a 61 75 74 63 56	CSQ0bGp zEJautcV			
0070	79 4a 4c 45 6b 63 70 6f	72 71	yJLEkcpo rq			

Figure 15. Wireshark view of last packet sent in second transaction

No.	Time	Source	Destination	Protocol	Length	Info
52	85.675801918	10.0.7.141	10.0.3.43	UDP	97	9000 → 6750 Len=55
User Datagram Protocol, Src Port: 9000, Dst Port: 6750 Source Port: 9000 Destination Port: 6750						
0000	06 5e fe f6 3c b2 06 78 11 22 93 7c 08 00 45 00				·^··<·x·"· ··E·	
0010	00 53 f1 a0 00 00 40 11 2a 42 0a 00 07 8d 0a 00				·S·@·@·B·0000	
0020	03 2b 23 28 1a 5e 00 3f 51 0e 41 43 4b 0a 30 30				·+·(·^·?·Q·ACK000	
0030	30 30 31 32 54 58 4e 30 30 30 37 34 31 34 4d 44				0012TXN0 007414MD	
0040	35 32 66 62 61 61 36 36 64 38 37 61 38 32 38 61				52fbaa66 d87a828a	
0050	39 62 61 34 33 61 30 37 66 65 62 30 38 34 36 66				9ba43a07 feb0846f	
0060	37				7	

Figure 16. Wireshark view of last ACK received in second transaction

6.3 Third Test (Transaction ID: 0007419)

For this test, the corresponding trace file is `test3_transaction0007419.pcap`.

Figure 17 shows packet 5 which contains the intent message “ID2b97c5ee” sent to the IP address of the server via the supplied port 9000. Figure 18 shows through the terminal and packet 6 that the transaction ID sent by the server is 0007419. Note that the source and destination IP addresses in this packet are reversed, and that the server is addressing the supplied port 6750.

As shown in packet 27 via **Figure 19**, the first ACK that the server sent corresponds to the packet with a payload size of 32 bytes. Included in the ACK is the checksum of the said packet. This checksum corresponds to packet 14 in **Figure 20**, as shown by the yellow rectangle.

Figure 21 shows the last packet sent by the client (with **LAST** flag set to **1**) and the corresponding last ACK sent by the server. Since the checksum of the packet and the ACK are equivalent, this is the correct pair of packets. **Figure 22** shows via packet 50 that the last client packet was indeed sent to the correct IP address (**10.0.7.141**) and port (**9000**). In the same way, **Figure 23** shows through packet 51 that the content of the last ACK matches with what is shown in the terminal and that the IP address and port (**6750**) belong to the client, as expected.

No.	Time	Source	Destination	Protocol	Length	Info
5	0.006652340	10.0.3.43	10.0.7.141	UDP	52	6750 → 9000 Len=10
User Datagram Protocol, Src Port: 6750, Dst Port: 9000						
Source Port: 6750						
Destination Port: 9000						
0000	06 78 11 22 93 7c 06 5e	fe f6 3c b2 08 00 45 00	x."	^	^	^
0010	00 26 6d 5f 40 00 40 11	ae b0 0a 00 03 2b 0a 00	&m	@	@	^
0020	07 8d 1a 5e 23 28 00 12	1e db 49 44 32 62 39 37	^	#	(^
0030	63 35 65 65		c5ee			

Figure 17. Intent message for the third transaction

=====

SENDING INTENT MESSAGE

=====

Intent message: ID2b97c5ee

Transaction ID: 0007419

No.	Time	Source	Destination	Protocol	Length	Info
6	0.304823596	10.0.7.141	10.0.3.43	UDP	49	9000 → 6750 Len=7
User Datagram Protocol, Src Port: 9000, Dst Port: 6750						
Source Port: 9000						
Destination Port: 6750						
0000	06 5e fe f6 3c b2 06 78	11 22 93 7c 08 00 45 00	^	^	^	^
0010	00 23 29 07 40 00 40 11	f3 0b 0a 00 07 8d 0a 00	#)	@	@	^
0020	03 2b 23 28 1a 5e 00 0f	d5 f9 30 30 30 37 34 31	+	#	(^
0030	39		9			

Figure 18. Transaction ID for the third transaction

Received ack: ACK0000000TXN0007419MD587cf320097b52eba43c64aef302be003

Max payload size = 32 characters

No.	Time	Source	Destination	Protocol	Length	Info
27	9.882801344	10.0.7.141	10.0.3.43	UDP	97	9000 → 6750 Len=55
User Datagram Protocol, Src Port: 9000, Dst Port: 6750						
Source Port: 9000						
Destination Port: 6750						
0000	06 5e fe f6 3c b2 06 78	11 22 93 7c 08 00 45 00	^	^	^	^
0010	00 53 2b ab 40 00 40 11	f0 37 0a 00 07 8d 0a 00	S+	@	@	^
0020	03 2b 23 28 1a 5e 00 3f	6e 68 41 43 4b 30 30 30	+	#	(^
0030	30 30 30 30 54 58 4e 30	30 30 37 34 31 39 4d 44	0000TXN0	007419MD		
0040	35 38 37 63 66 33 32 30	30 39 37 62 35 32 65 62	587cf320	097b52eb		
0050	61 34 33 63 36 34 61 65	66 33 30 32 62 65 30 30	a43c64ae	f302be00		
0060	33		3			

Figure 19. First acknowledgement received for the third transaction

Sent message: ID2b97c5eeSN0000000TXN0007419LAST0bzJpFZeMwfInKgmVcvhKAMoeBXojxaXh

Checksum: 87cf320097b52eba43c64aef302be003

No.	Time	Source	Destination	Protocol	Length	Info
14	3.812267018	10.0.3.43	10.0.7.141	UDP	108	6750 → 9000 Len=66
User Datagram Protocol, Src Port: 6750, Dst Port: 9000						
Source Port: 6750						
Destination Port: 9000						
0000	06 78 11 22 93 7c 06 5e	fe f6 3c b2 08 00 45 00	x."	^	^	^
0010	00 5e 6f d3 40 00 40 11	ac 04 0a 00 03 2b 0a 00	^	o	@	^
0020	07 8d 1a 5e 23 28 00 4a	1f 13 49 44 32 62 39 37	^	^	#	(
0030	63 35 65 65 53 4e 30 30	30 30 30 30 30 54 58 4e	c5eeSN00	00000TXN		
0040	30 30 30 37 34 31 39 4c	41 53 54 30 62 7a 4a 70	0007419L	AST0bzJp		
0050	46 5a 65 4d 57 66 69 6e	4b 67 6d 56 63 76 68 4b	FZeMwfIn	KgmVcvhK		
0060	41 4d 6f 65 42 58 6f 6a	78 61 58 68	AMoeBXoj	xaXh		

Figure 20. First packet that was acknowledged in the third transaction

Sent message: ID2b97c5eeSN00000012TXN0007419LAST1IISLBazylLoPRLQu (384 / 399)

Checksum: b46fbe81a4ee35d2a464d43e8321ebe5

Received ack: ACK0000012TXN0007419MD5b46fbe81a4ee35d2a464d43e8321ebe5

=====

TRANSACTION TERMINATED

=====

Figure 21. Last packet sent and last ACK received in the third transaction

No.	Time	Source	Destination	Protocol	Length	Info
50	77.481027931	10.0.3.43	10.0.7.141	UDP	91	6750 → 9000 Len=49
User Datagram Protocol, Src Port: 6750, Dst Port: 9000						
Source Port: 6750						
Destination Port: 9000						
0000	06 78 11 22 93 7c 06 5e	fe f6 3c b2 08 00 45 00	x."	^	^	^
0010	00 4d 95 90 40 00 40 11	86 58 0a 00 03 2b 0a 00	M	^	@	^
0020	07 8d 1a 5e 23 28 00 39	1f 02 49 44 32 62 39 37	^	^	#	(
0030	63 35 65 65 53 4e 30 30	30 30 30 31 32 54 58 4e	c5eeSN00	00012TXN		
0040	30 30 30 37 34 31 39 4c	41 53 54 31 49 53 6c 53	0007419L	AST1IISL		
0050	42 61 7a 79 4c 6f 50 52	4c 51 75	BazylLoPR	LQu		

Figure 22. Wireshark view of last packet sent in third transaction

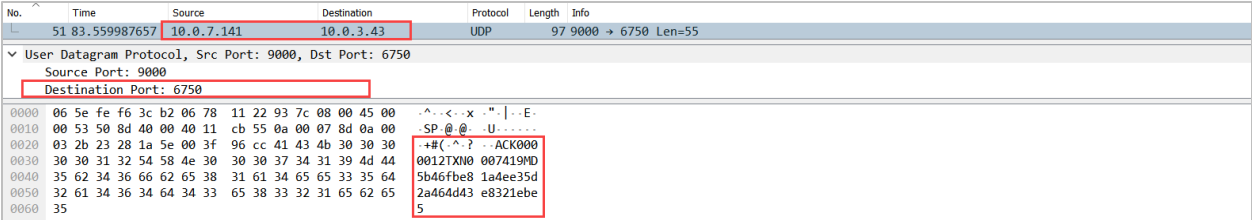


Figure 23. Wireshark view of last ACK received in third transaction

6.4 Fourth Test (Transaction ID: 0007423)

For this test, the corresponding trace file is test4_transaction0007423.pcap.

Figure 24 shows packet 7 which contains the intent message “ID2b97c5ee” sent to the IP address of the server via the supplied port 9000. Figure 25 shows through the terminal and packet 8 that the transaction ID sent by the server is 0007423. Note that the source and destination IP addresses in this packet are reversed, and that the server is addressing the supplied port 6750.

As shown in packet 34 via Figure 26, the first ACK that the server sent corresponds to the packet with a payload size of 21 bytes. Included in the ACK is the checksum of the said packet. This checksum corresponds to packet 27 in Figure 27, as shown by the yellow rectangle.

Figure 28 shows the last packet sent by the client (with LAST flag set to 1) and the corresponding last ACK sent by the server. Since the checksum of the packet and the ACK are equivalent, this is the correct pair of packets. Figure 29 shows via packet 85 that the last client packet was indeed sent to the correct IP address (10.0.7.141) and port (9000). In the same way, Figure 30 shows through packet 86 that the content of the last ACK matches with what is shown in the terminal and that the IP address and port (6750) belong to the client, as expected.

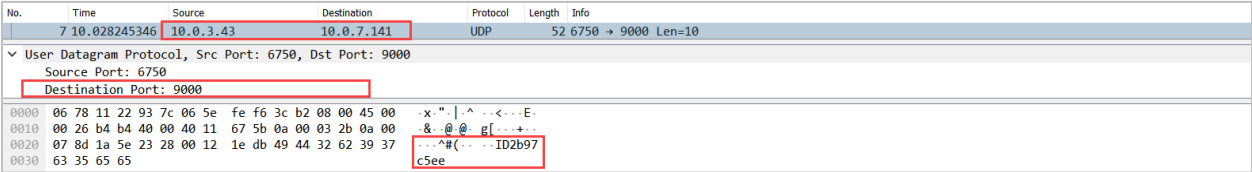


Figure 24. Intent message for the fourth transaction

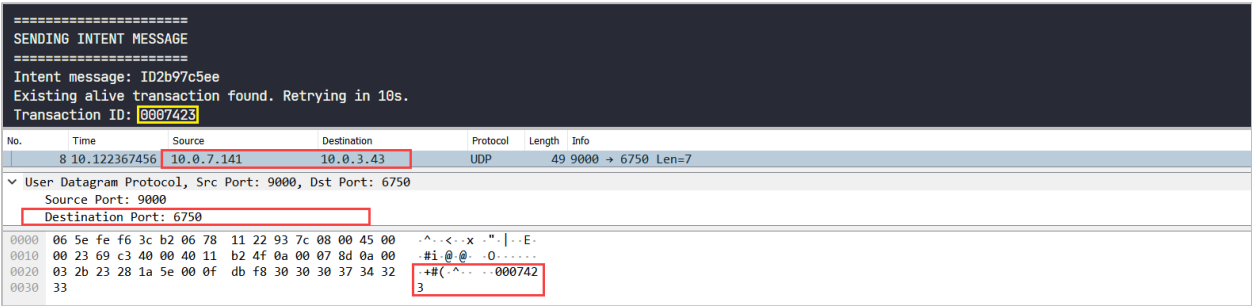


Figure 25. Transaction ID for the fourth transaction

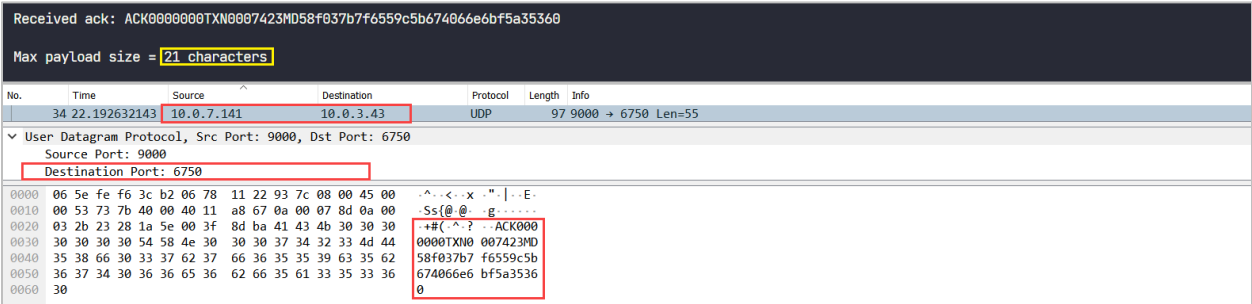


Figure 26. First acknowledgement received for the fourth transaction

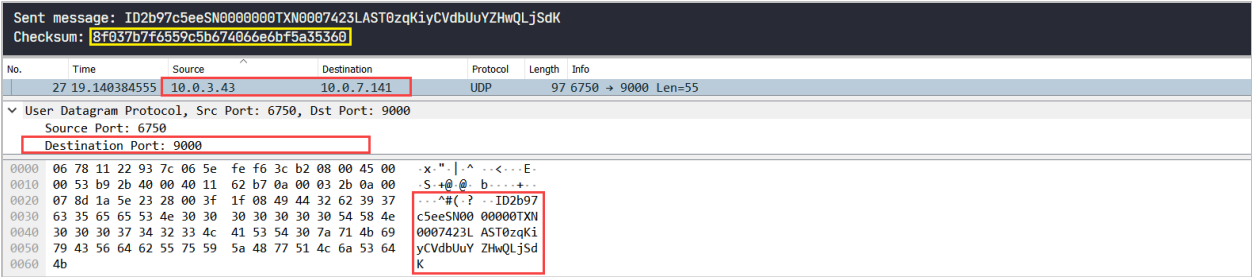


Figure 27. First packet that was acknowledged in the fourth transaction

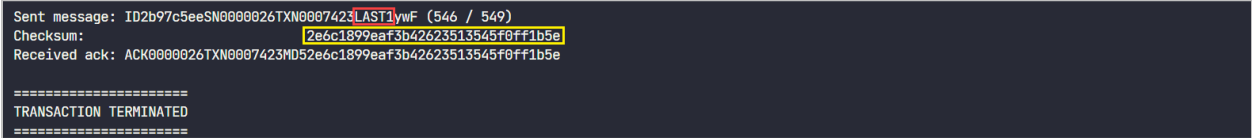


Figure 28. Last packet sent and last ACK received in the fourth transaction

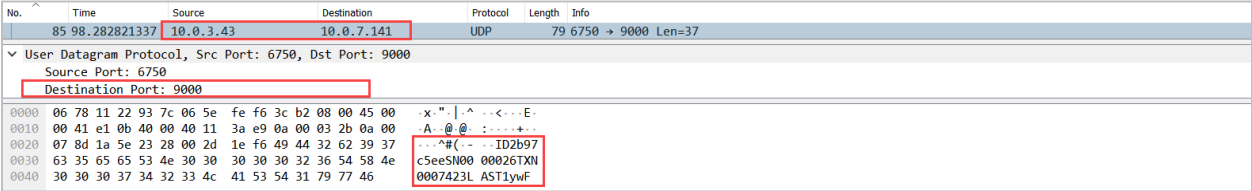


Figure 29. Wireshark view of last packet sent in fourth transaction

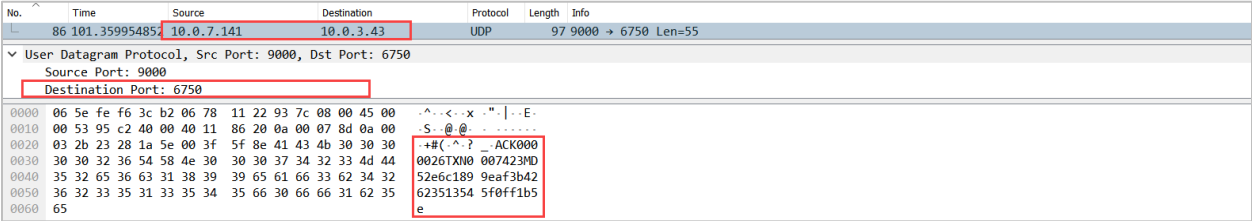


Figure 30. Wireshark view of last ACK received in fourth transaction

6.5 Fifth Test (Transaction ID: 0007427)

For this test, the corresponding trace file is `test5_transaction0007427.pcap`.

Figure 31 shows packet 5 which contains the intent message “ID2b97c5ee” sent to the IP address of the server via the supplied port 9000. Figure 32 shows through the terminal and packet 6 that the transaction ID sent by the server is 0007427. Note that the source and destination IP addresses in this packet are reversed, and that the server is addressing the supplied port 6750.

As shown in packet 25 via Figure 33, the first ACK that the server sent corresponds to the packet with a payload size of 38 bytes. Included in the ACK is the checksum of the said packet. This checksum corresponds to packet 11 in Figure 34, as shown by the yellow rectangle.

Figure 35 shows the last packet sent by the client (with LAST flag set to 1) and the corresponding last ACK sent by the server. Since the checksum of the packet and the ACK are equivalent, this is the correct pair of packets. Figure 36 shows via packet 44 that the last client packet was indeed sent to the correct IP address (10.0.7.141) and port (9000). In the same way, Figure 37 shows through packet 45 that the content of the last ACK matches with what is shown in the terminal and that the IP address and port (6750) belong to the client, as expected.

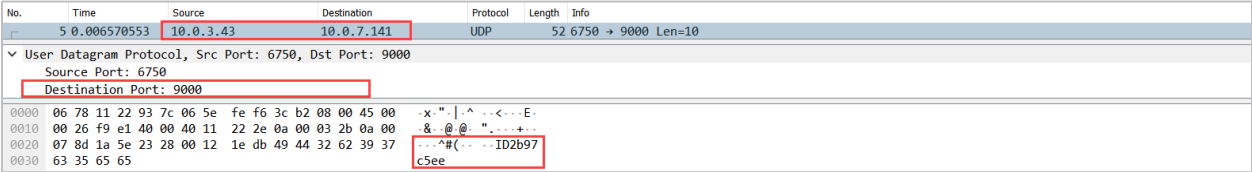


Figure 31. Intent message for the fifth transaction

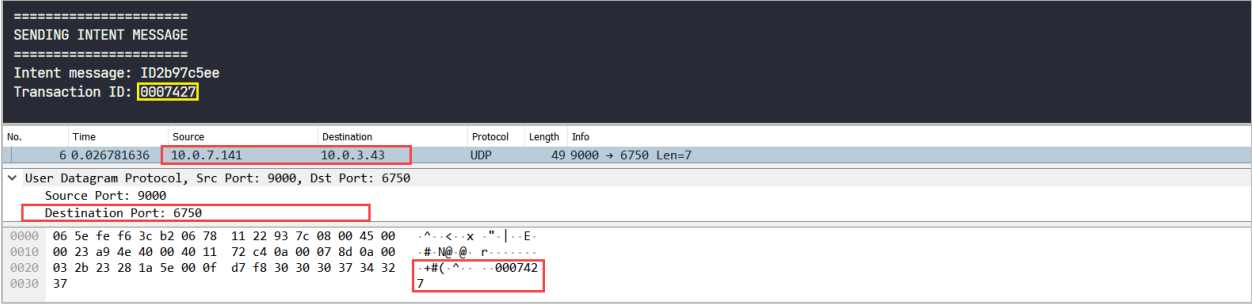


Figure 32. Transaction ID for the fifth transaction

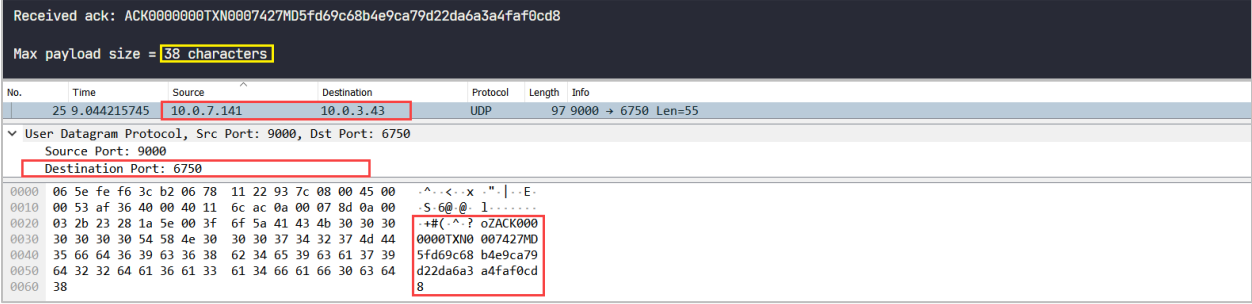


Figure 33. First acknowledgement received for the fifth transaction

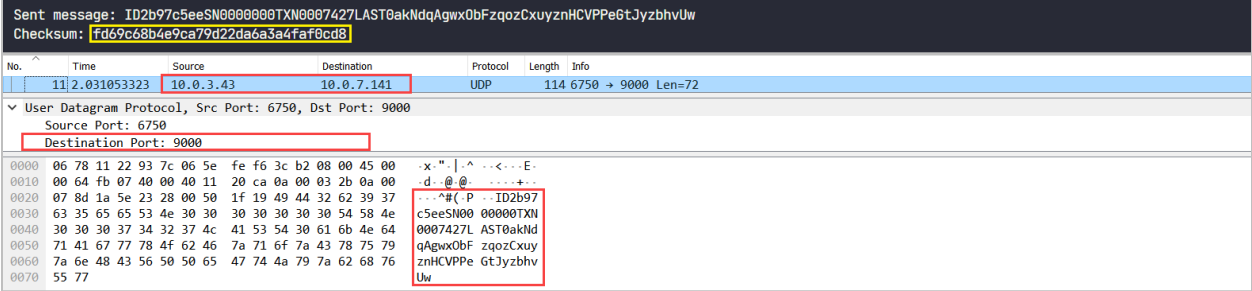


Figure 34. First packet that was acknowledged in the fifth transaction

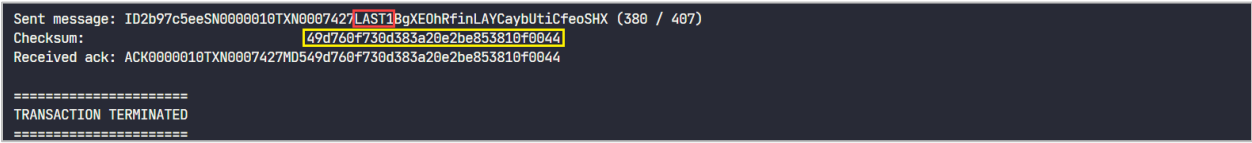


Figure 35. Last packet sent and last ACK received in the fifth transaction

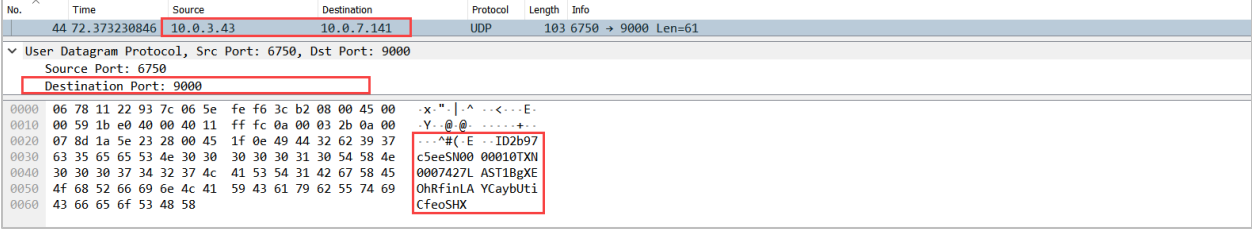


Figure 36. Wireshark view of last packet sent in fifth transaction

No.	Time	Source	Destination	Protocol	Length	Info
45	79.480132431	10.0.7.141	10.0.3.43	UDP	97	9000 → 6750 Len=55
User Datagram Protocol, Src Port: 9000, Dst Port: 6750						
		Source Port: 9000				
		Destination Port: 6750				
0000	06 5e fe f6 3c b2 06 78	11 22 93 7c 08 00 45 00	^..<.x.. ..E-			
0010	00 53 cd fd 40 00 40 11	4d e5 0a 00 07 8d 0a 00	S..@.M.....			
0020	03 2b 23 28 1a 5e 00 3f	f6 95 41 43 4b 30 30 30	+#(^.^?..ACK000			
0030	30 30 31 30 54 58 4e 30	30 30 37 34 32 37 4d 44	0010TXN0 007427MD			
0040	35 34 39 64 37 36 30 66	37 33 30 64 33 38 33 61	549d760f 730d383a			
0050	32 30 65 32 62 65 38 35	33 38 31 30 66 30 30 34	20e2be85 3810f004			
0060	34		4			

Figure 37. Wireshark view of last ACK received in fifth transaction