

## بخش اول:

تحلیل و بررسی کد UDP با Wireshark

پس از اجرای کد UDP Server، برنامه UDP Client می‌تواند به سرور پیام بفرستد و زمانی که پیام "shutdown" را ارسال کند، socket کلاینت بسته و سرور نیز shutdown خواهند شد.

## UDP Client

```
In [1]: import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
serverAddress = ('localhost', 12345)

while True:
    message = input('Enter your message: ').encode()

    print('Sending', len(message), 'bytes to', serverAddress[0], 'port', serverAddress[1], '\n')
    sent = sock.sendto(message, serverAddress)
    if message == 'shutdown'.encode(): break

print('\nClosing socket')
sock.close()
```

```
Enter your message: Hi
Sending 2 bytes to localhost port 12345
```

```
Enter your message: My name is Parinaz Akef.
Sending 24 bytes to localhost port 12345
```

```
Enter your message: This is a UDP message
Sending 21 bytes to localhost port 12345
```

```
Enter your message: shutdownn
Sending 9 bytes to localhost port 12345
```

```
Enter your message: shutdown
Sending 8 bytes to localhost port 12345
```

```
Closing socket
```

## UDP Server

```
In [1]: import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) # Create a UDP socket

serverAddress = ('localhost', 12345)
sock.bind(serverAddress)

while True:
    print('Waiting to receive...')
    data, address = sock.recvfrom(4096)

    print('Received', len(data), 'bytes from', address)
    print(data.decode(), '\n')

    if data.decode() == 'shutdown':
        print('\nShutting down the server.')
        break

sock.close()

Waiting to receive...
Received 2 bytes from ('127.0.0.1', 54430)
Hi

Waiting to receive...
Received 24 bytes from ('127.0.0.1', 54430)
My name is Parinaz Akef.

Waiting to receive...
Received 21 bytes from ('127.0.0.1', 54430)
This is a UDP message

Waiting to receive...
Received 9 bytes from ('127.0.0.1', 54430)
shutdown

Waiting to receive...
Received 8 bytes from ('127.0.0.1', 54430)
shutdown

Shutting down the server.
```

### شکل ۲ – کد پایتون UDP Server

مطابق اطلاعات نمایش داده شده در شکل ۳، مقادیر نشان داده شده را می‌توان به این شکل تحلیل کرد:

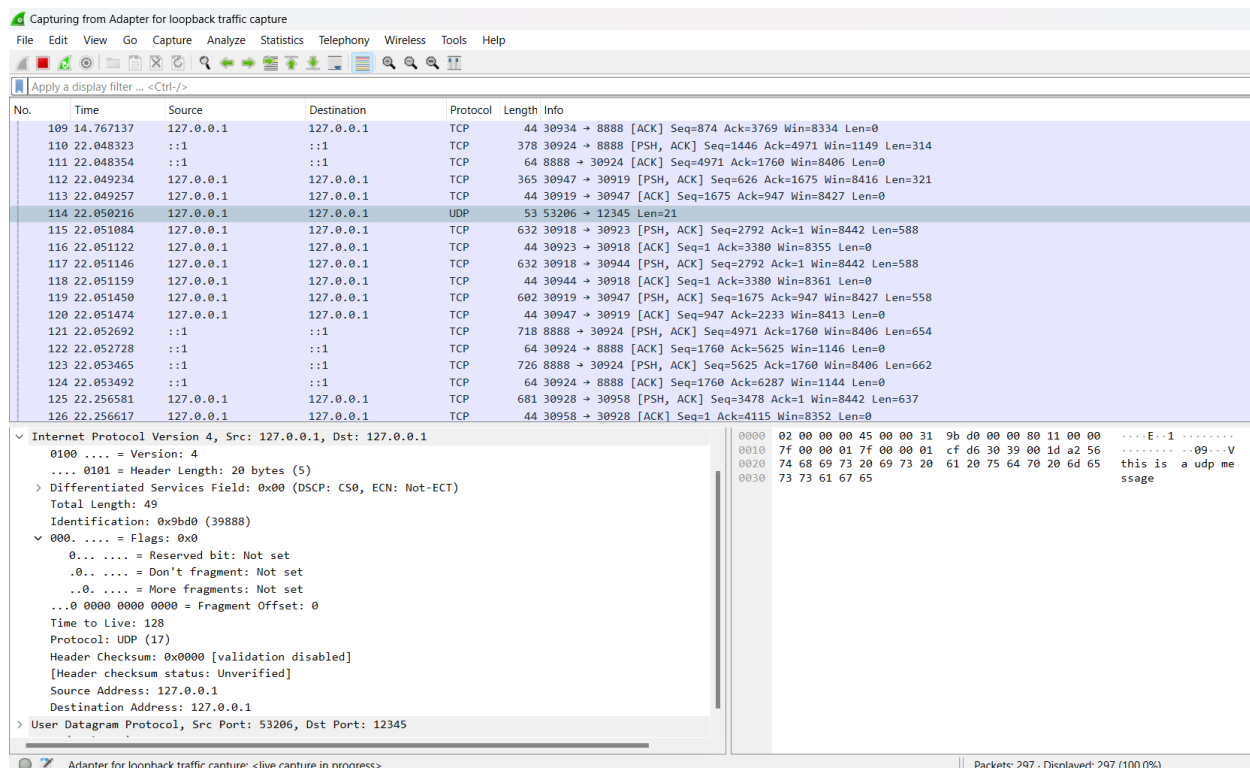
0000	02 00 00 00 45 00 00 31	9b d0 00 00 80 11 00 00
0010	7f 00 00 01 7f 00 00 01	cf d6 30 39 00 1d a2 56
0020	74 68 69 73 20 69 73 20	61 20 75 64 70 20 6d 65
0030	73 73 61 67 65	

### شکل ۴ – تحلیل مقادیر

خط اول، اطلاعات هدر را نشان می‌دهد و دو خط پایین، پیام است.

در ۱۰ بایت بعدی به ترتیب رنگ از سمت چپ این موارد مشخص شده اند:

✓ Source Address, Destination Address, Source Port, Destination Port, Length, Checksum



شکل ۳ – اطلاعات پیام در Wireshark

تحلیل بخش پایین سمت چپ در شکل ۳:

بخش Reserved Bit: به 0 ست شده که به این معنی است که این بیت رزرو نشده و معمولاً در هدر های UDP به این فیلد مقدار صفر داده میشود.

بیت DF: به 0 ست شده، به این معنی که packet میتواند در صورت نیاز fragment شود. (در هدر IPv4، اجازه دسترسی router ها برای fragment کردن پکت را تعیین میکنند. (بیت Don't Fragment)

Checksum: مقدارش 0x0000 قرار داده شده و validation خاموش است. در UDP، checksum اختیاری است و مقدار صفر به این معنی است که استفاده نشده.

آدرس مبدا (Source Address): در اینجا 127.0.0.1، این یک آدرس loopback است و برای ارتباط در همان سیستم استفاده میشود.

آدرس مقصد (Destination Address): در اینجا 127.0.0.1 که loopback است و مشخص میکند که مقصد packet همان سیستم است.

## تحلیل و بررسی کد TCP با Wireshark

پس از اجرای کد TCP Server، برنامه TCP Client میتواند به سرور پیام بفرستد و زمانی که پیام "0" را ارسال کند، socket کلاینت بسته و سرور نیز shutdown خواهد شد.

## TCP Server

```
In [2]: import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

serverAddress = ('localhost', 12345)
sock.bind(serverAddress)
sock.listen(1)

while True:
    print('Waiting for a connection...')
    connection, clientAddress = sock.accept()

    print('Connection from', clientAddress)

    # For receiving data
    while True:
        data = connection.recv(16) # buffer size of 16
        print('\nReceived', data.decode())

        if data:
            print('Sending data back to the client.')
            connection.sendall(data)
            if data == '0'.encode(): # When the client enters 0, the server shuts down
                break
        else:
            print('No more data from', clientAddress)
            break
    connection.close()
    if data == '0'.encode():
        break

print('Shutting down the server.')
sock.close()
```

شکل ۵ – کد پایتون سرور TCP

```
break

print('Shutting down the server.')
sock.close()
```

Waiting for a connection...  
Connection from ('127.0.0.1', 23407)

Received Hello  
Sending data back to the client.

Received my name is parin  
Sending data back to the client.

Received az akef  
Sending data back to the client.

Received this is a tcp me  
Sending data back to the client.

Received ssage  
Sending data back to the client.

Received 0  
Sending data back to the client.  
Shutting down the server.

شکل ۶ – وضعیت اعلام شده توسط کد سرور TCP

## TCP Client

```
In [2]: import socket
import sys

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

serverAddress = ('localhost', 12345)
print('Connecting to', serverAddress[0], 'port', serverAddress[1])
sock.connect(serverAddress)

while True:

    # Sending data
    message = input('\nEnter your message:').encode()
    print('Sending', message)
    sock.sendall(message)

    # Looking for the response
    amount_received, amount_expected = 0, len(message)

    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        print('Received', data)

    if message == '0'.encode(): break

print('Closing socket')
sock.close()
```

شکل ۷ – کد پایتون کلاینت TCP

```
print('Closing socket')
sock.close()
```

Connecting to localhost port 12345

Enter your message:Hello  
Sending b'Hello'  
Received b'Hello'

Enter your message:my name is parinaz akef  
Sending b'my name is parinaz akef'  
Received b'my name is parin'  
Received b'az akef'

Enter your message:this is a tcp message  
Sending b'this is a tcp message'  
Received b'this is a tcp me'  
Received b'ssage'

Enter your message:0  
Sending b'0'  
Received b'0'  
Closing socket

شکل ۸ – وضعیت اعلام شده توسط کلاینت TCP

## بررسی روند انتقال packet و دریافت گواهی برای یک پیام:

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 12345

No.	Time	Source	Destination	Protocol	Length	Info
14376	878.317464	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=3 Ack=26 Win=2161152 Len=0
14377	878.317599	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=3 Ack=26 Win=2161152 Len=16
14378	878.317614	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=19 Win=2161152 Len=0
14379	878.317843	127.0.0.1	127.0.0.1	TCP	51	12345 → 23451 [PSH, ACK] Seq=19 Ack=26 Win=2161152 Len=7
14380	878.317859	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=26 Win=2161152 Len=0
14493	881.513761	127.0.0.1	127.0.0.1	TCP	65	23451 → 12345 [PSH, ACK] Seq=26 Ack=26 Win=2161152 Len=21
14494	881.513789	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=26 Ack=47 Win=2161152 Len=0
14495	881.513923	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=26 Ack=47 Win=2161152 Len=16
14496	881.513947	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=42 Win=2161152 Len=0
14497	881.514159	127.0.0.1	127.0.0.1	TCP	49	12345 → 23451 [PSH, ACK] Seq=42 Ack=47 Win=2161152 Len=5

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 23451, Dst Port: 12345, Seq: 26, Ack: 26, Len: 21

Source Port: 23451

Destination Port: 12345

[Stream index: 28]

> [Conversation completeness: Complete, WITH\_DATA (31)]

[TCP Segment Len: 21]

Sequence Number: 26 (relative sequence number)

Sequence Number (raw): 557982978

[Next Sequence Number: 47 (relative sequence number)]

Acknowledgment Number: 26 (relative ack number)

Acknowledgment number (raw): 778096354

0101 ... = Header Length: 20 bytes (5)

> Flags: 0x018 (PSH, ACK)

Window: 8442

[Calculated window size: 2161152]

[Window size scaling factor: 256]

Checksum: 0x6315 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> [Timestamps]

> [SEQ/ACK analysis]

TCP payload (21 bytes)

> Data (21 bytes)

Data: 74686973206973206120746370206d657373616765

[Length: 21]

0000 02 00 00 00 45 00 00 3d 9a 45 40 00 00 06 00 00 ...E...-E@....  
0010 7f 00 00 01 7f 00 00 01 5b 9b 30 39 21 42 25 02 .....[.0918%  
0020 2e 60 ce e2 50 18 20 fa 63 15 00 00 74 68 69 73 ...P...c...this  
0030 20 69 73 20 61 20 74 63 70 20 6d 65 73 73 61 67 is a tc p messag  
0040 65

wireshark\_NPF\_LoopbackMY7CG2.pcapng

Packets: 17744 · Displayed: 94 (0.5%)

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 12345

No.	Time	Source	Destination	Protocol	Length	Info
14376	878.317464	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=3 Ack=26 Win=2161152 Len=0
14377	878.317599	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=3 Ack=26 Win=2161152 Len=16
14378	878.317614	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=19 Win=2161152 Len=0
14379	878.317843	127.0.0.1	127.0.0.1	TCP	51	12345 → 23451 [PSH, ACK] Seq=19 Ack=26 Win=2161152 Len=7
14380	878.317859	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=26 Win=2161152 Len=0
14493	881.513761	127.0.0.1	127.0.0.1	TCP	65	23451 → 12345 [PSH, ACK] Seq=26 Ack=26 Win=2161152 Len=21
14494	881.513789	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=26 Ack=47 Win=2161152 Len=0
14495	881.513923	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=26 Ack=47 Win=2161152 Len=16
14496	881.513947	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=42 Win=2161152 Len=0
14497	881.514159	127.0.0.1	127.0.0.1	TCP	49	12345 → 23451 [PSH, ACK] Seq=42 Ack=47 Win=2161152 Len=5

> Frame 14494: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF\_{Loopback}

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 12345, Dst Port: 23451, Seq: 26, Ack: 47, Len: 0

Source Port: 12345

Destination Port: 23451

[Stream index: 28]

> [Conversation completeness: Complete, WITH\_DATA (31)]

[TCP Segment Len: 0]

Sequence Number: 26 (relative sequence number)

Sequence Number (raw): 778096354

[Next Sequence Number: 26 (relative sequence number)]

Acknowledgment Number: 47 (relative ack number)

Acknowledgment number (raw): 557982999

0101 ... = Header Length: 20 bytes (5)

> Flags: 0x010 (ACK)

Window: 8442

[Calculated window size: 2161152]

[Window size scaling factor: 256]

Checksum: 0xc167 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> [Timestamps]

> [SEQ/ACK analysis]

0000 02 00 00 00 45 00 00 28 9a 46 40 00 00 06 00 00 ...E...(- F@....  
0010 7f 00 00 01 7f 00 00 01 30 39 5b 9b 2e 60 ce e2 .....09[....  
0020 21 42 25 17 50 10 20 fa c1 67 00 00 1b%P...g...

wireshark\_NPF\_LoopbackMY7CG2.pcapng

Packets: 18179 · Displayed: 94 (0.5%)

شکل ۸ و ۹ – TCP handshake و ACK آن

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 12345

No.	Time	Source	Destination	Protocol	Length	Info
14376	878.317464	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=3 Ack=26 Win=2161152 Len=0
14377	878.317599	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=3 Ack=26 Win=2161152 Len=16
14378	878.317614	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=19 Win=2161152 Len=0
14379	878.317843	127.0.0.1	127.0.0.1	TCP	51	12345 → 23451 [PSH, ACK] Seq=19 Ack=26 Win=2161152 Len=7
14380	878.317859	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=26 Win=2161152 Len=0
14493	881.513761	127.0.0.1	127.0.0.1	TCP	65	23451 → 12345 [PSH, ACK] Seq=26 Ack=26 Win=2161152 Len=21
14494	881.513789	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=26 Ack=47 Win=2161152 Len=0
14495	881.513923	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=26 Ack=47 Win=2161152 Len=16
14496	881.513947	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=42 Win=2161152 Len=0
14497	881.514159	127.0.0.1	127.0.0.1	TCP	49	12345 → 23451 [PSH, ACK] Seq=42 Ack=47 Win=2161152 Len=5

> Frame 14495: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface \Device\NPF\_{...}

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 12345, Dst Port: 23451, Seq: 26, Ack: 47, Len: 16

Source Port: 12345

Destination Port: 23451

[Stream index: 28]

> [Conversation completeness: Complete, WITH\_DATA (31)]

[TCP Segment Len: 16]

Sequence Number: 26 (relative sequence number)

Sequence Number (raw): 778096354

[Next Sequence Number: 42 (relative sequence number)]

Acknowledgment Number: 47 (relative ack number)

Acknowledgment number (raw): 557982999

0101 ... = Header Length: 20 bytes (5)

> Flags: 0x018 (PSH, ACK)

Window: 8442

[Calculated window size: 2161152]

[Window size scaling factor: 256]

Checksum: 0x9ce0 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> [Timestamps]

> [SEQ/ACK analysis]

TCP payload (16 bytes)

Data (16 bytes)

0000 02 00 00 00 45 00 00 38 9a 47 40 00 80 06 00 00 ....E..8.G@....

0010 7f 00 00 01 7f 00 00 01 30 39 5b 9b 2e 60 ce e2 .....09[....

0020 21 42 25 17 50 18 20 fa 9c e0 00 00 74 68 69 73 !B%P-...this

0030 20 69 73 20 61 20 74 63 70 20 6d 65 is a tc p me

Packets: 18495 · Displayed: 94 (0.5%)

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 12345

No.	Time	Source	Destination	Protocol	Length	Info
14377	878.317599	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=3 Ack=26 Win=2161152 Len=16
14378	878.317614	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=19 Win=2161152 Len=0
14379	878.317843	127.0.0.1	127.0.0.1	TCP	51	12345 → 23451 [PSH, ACK] Seq=19 Ack=26 Win=2161152 Len=7
14380	878.317859	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=26 Win=2161152 Len=0
14493	881.513761	127.0.0.1	127.0.0.1	TCP	65	23451 → 12345 [PSH, ACK] Seq=26 Ack=26 Win=2161152 Len=21
14494	881.513789	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=26 Ack=47 Win=2161152 Len=0
14495	881.513923	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=26 Ack=47 Win=2161152 Len=16
14496	881.513947	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=42 Win=2161152 Len=0
14497	881.514159	127.0.0.1	127.0.0.1	TCP	49	12345 → 23451 [PSH, ACK] Seq=42 Ack=47 Win=2161152 Len=5
14498	881.514179	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=47 Win=2161152 Len=0

> Frame 14496: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF\_{...}

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

Transmission Control Protocol, Src Port: 23451, Dst Port: 12345, Seq: 47, Ack: 42, Len: 0

Source Port: 23451

Destination Port: 12345

[Stream index: 28]

> [Conversation completeness: Complete, WITH\_DATA (31)]

[TCP Segment Len: 0]

Sequence Number: 47 (relative sequence number)

Sequence Number (raw): 557982999

[Next Sequence Number: 47 (relative sequence number)]

Acknowledgment Number: 42 (relative ack number)

Acknowledgment number (raw): 778096370

0101 ... = Header Length: 20 bytes (5)

> Flags: 0x010 (ACK)

Window: 8442

[Calculated window size: 2161152]

[Window size scaling factor: 256]

Checksum: 0xc157 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> [Timestamps]

> [SEQ/ACK analysis]

0000 02 00 00 00 45 00 00 28 9a 48 40 00 80 06 00 00 ....E..(..H@....

0010 7f 00 00 01 7f 00 00 01 5b 9b 30 39 21 42 25 17 .....[..09!B%

0020 2e 60 ce f2 50 10 20 fa c1 57 00 00 ...P-...W...

Packets: 18579 · Displayed: 94 (0.5%)

شکل ۱۰ و ۱۱ – فرستادن ۱۶ بایت اول پیام و دریافت ACK آن

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 12345

No.	Time	Source	Destination	Protocol	Length	Info
14377	878.317599	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=3 Ack=26 Win=2161152 Len=16
14378	878.317614	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=19 Win=2161152 Len=0
14379	878.317843	127.0.0.1	127.0.0.1	TCP	51	12345 → 23451 [PSH, ACK] Seq=19 Ack=26 Win=2161152 Len=7
14380	878.317859	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=26 Ack=26 Win=2161152 Len=0
14493	881.513761	127.0.0.1	127.0.0.1	TCP	65	23451 → 12345 [PSH, ACK] Seq=26 Ack=26 Win=2161152 Len=21
14494	881.513789	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=26 Ack=47 Win=2161152 Len=0
14495	881.513923	127.0.0.1	127.0.0.1	TCP	60	12345 → 23451 [PSH, ACK] Seq=26 Ack=47 Win=2161152 Len=16
14496	881.513947	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=42 Win=2161152 Len=0
14497	881.514159	127.0.0.1	127.0.0.1	TCP	49	12345 → 23451 [PSH, ACK] Seq=42 Ack=47 Win=2161152 Len=5
14498	881.514179	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=47 Win=2161152 Len=0

> Frame 14497: 49 bytes on wire (392 bits), 49 bytes captured (392 bits) on interface \Device\NPF\_{...}

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 12345, Dst Port: 23451, Seq: 42, Ack: 47, Len: 5

Source Port: 12345

Destination Port: 23451

[Stream index: 28]

> [Conversation completeness: Complete, WITH\_DATA (31)]

[TCP Segment Len: 5]

Sequence Number: 42 (relative sequence number)

Sequence Number (raw): 778096370

[Next Sequence Number: 47 (relative sequence number)]

Acknowledgment Number: 47 (relative ack number)

Acknowledgment number (raw): 557982999

0101 .... = Header Length: 20 bytes (5)

> Flags: 0x018 (PSH, ACK)

Window: 8442

[Calculated window size: 2161152]

[Window size scaling factor: 256]

Checksum: 0x876f [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> [Timestamps]

> [SEQ/ACK analysis]

TCP payload (5 bytes)

> Data (5 bytes)

0000 02 00 00 00 45 00 00 2d 9a 49 40 00 80 06 00 00 ....E---.I@....

0010 7f 00 00 01 7f 00 00 01 30 39 5b 9b 2e 60 ce f2 .....09[...'

0020 21 42 25 17 50 18 20 fa 87 6f 00 00 73 73 61 67 1B%-P-...o-ssag

0030 65

wireshark\_NPF\_LoopbackMY7CG2.pcapng

Packets: 18634 · Displayed: 94 (0.5%)

\*Adapter for loopback traffic capture

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.port == 12345

No.	Time	Source	Destination	Protocol	Length	Info
14496	881.513947	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=42 Win=2161152 Len=0
14497	881.514159	127.0.0.1	127.0.0.1	TCP	49	12345 → 23451 [PSH, ACK] Seq=42 Ack=47 Win=2161152 Len=5
14498	881.514179	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=47 Ack=47 Win=2161152 Len=0
14529	883.241446	127.0.0.1	127.0.0.1	TCP	47	23451 → 12345 [PSH, ACK] Seq=47 Ack=47 Win=2161152 Len=3
14530	883.241470	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=47 Ack=50 Win=2161152 Len=0
14531	883.241630	127.0.0.1	127.0.0.1	TCP	47	12345 → 23451 [PSH, ACK] Seq=47 Ack=50 Win=2161152 Len=3
14532	883.241651	127.0.0.1	127.0.0.1	TCP	44	23451 → 12345 [ACK] Seq=50 Ack=50 Win=2161152 Len=0
14559	884.916297	127.0.0.1	127.0.0.1	TCP	45	23451 → 12345 [PSH, ACK] Seq=50 Ack=50 Win=2161152 Len=1
14560	884.916320	127.0.0.1	127.0.0.1	TCP	44	12345 → 23451 [ACK] Seq=50 Ack=51 Win=2161152 Len=0
14561	884.916492	127.0.0.1	127.0.0.1	TCP	45	12345 → 23451 [PSH, ACK] Seq=50 Ack=51 Win=2161152 Len=1

> Frame 14498: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface \Device\NPF\_{...}

> Null/Loopback

> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1

> Transmission Control Protocol, Src Port: 23451, Dst Port: 12345, Seq: 47, Ack: 47, Len: 0

Source Port: 23451

Destination Port: 12345

[Stream index: 28]

> [Conversation completeness: Complete, WITH\_DATA (31)]

[TCP Segment Len: 0]

Sequence Number: 47 (relative sequence number)

Sequence Number (raw): 557982999

[Next Sequence Number: 47 (relative sequence number)]

Acknowledgment Number: 47 (relative ack number)

Acknowledgment number (raw): 778096375

0101 .... = Header Length: 20 bytes (5)

> Flags: 0x010 (ACK)

Window: 8442

[Calculated window size: 2161152]

[Window size scaling factor: 256]

Checksum: 0xc152 [unverified]

[Checksum Status: Unverified]

Urgent Pointer: 0

> [Timestamps]

> [SEQ/ACK analysis]

0000 02 00 00 00 45 00 00 28 9a 4a 40 00 80 06 00 00 ....E-({.J@....

0010 7f 00 00 01 7f 00 00 01 5b 9b 30 39 21 42 25 17 .....[.091B%

0020 2e 60 ce f7 50 10 20 fa c1 52 00 00 .....P-...R...

wireshark\_NPF\_LoopbackMY7CG2.pcapng

Packets: 18667 · Displayed: 94 (0.5%)

شکل ۱۲ و ۱۳ – فرستادن ۵ بایت بعدی پیام و دریافت ACK آن



طبق عکس ها مشاهده میشود به دلیل گذاشتن محدودیت ۱۶ بیت برای پیام ها، پیامی که ۲۱ بایت داشت به دو پیام ۱۶ و ۵ بایتی تقسیم شد که از فیلد len پیام آنها میتوان دید. پس از handshake و اطمینان از برقرار بودن ارتباط، (زیرا بر خلاف UDP، TCP این مورد را بررسی میکند) ۱۶ بایت فرستاده شد و پس از دریافت آن توسط سرور، یک ACK برای گواهی گرفتن این بسته ارسال میشود. این عمل برای ۵ بایت بعدی نیز انجام میشود.

در بررسی فیلد ها برای مقال در مرحله ای که ۵ بایت ارسال میشود میتوان به این شکل تحلیل را انجام داد (در قسمت Transmission Control Protocol):

Source Port و Destination Port به ترتیب 12345 و 23451 می باشند. اندازه TCP Segment نیز ۵ بایت مشخص شده است. به این بسته Sequence Number ۴۲ تعلق گرفته و همچنین عدد sequence بعدی، ۴۷ است. Flag های Ack و Push، ۱ گذاشته شده اند. همچنین مقدار Checksum به 0x876f گذاشته شده و صفر نبودن آن، نیاز ما به استفاده از آن در ارتباط TCP را نشان میدهد.

0000	02 00 00 00 45 00 00 2d	9a 49 40 00 80 06 00 00	....E-- -I@.....
0010	7f 00 00 01 7f 00 00 01	30 39 5b 9b 2e 60 ce f2	.....09[.~..
0020	21 42 25 17 50 18 20 fa	87 6f 00 00 73 73 61 67	!B%-P. .o..ssag
0030	65		e

شکل ۱۴ – بررسی بایت های پیام TCP

در عکس ۱۴ به ترتیب از چپ به راست داریم:

✓ Source Address, Destination Address, Source Port, Destination Port, Sequence Number

در خط اول اطلاعات مربوط به هدر (مانند Family, Protocol, Length و checksum) تعیین شده اند و خطوط پایین حاوی پیام میباشند.

## بخش دوم:

### ایجاد یک پیام DNS Query و بازگرداندن IP گوگل

در کل پایتون نشان داده شده در شکل ۱۵، در تابع structDNSQuery، با دریافت یک میزبان (domain name) اقدام به ساخت یک DNS query در فرمت مشخص شده را میکند. (فیلد های مختلف را با استفاده از struct.pack، pack میکند)

در قسمت main برنامه، پس از ساخت یک UDP socket و دریافت DNS query برای 'google.com'، با استفاده از socket این query را به DNS server با IP 8.8.8.8 و port 53 میفرستد. (مخصوص گوگل)

پس از دریافت response و ذخیره آن در result، بخشی که مربوط به دیتا و اطلاعات غیر هدر است را جدا میکنیم. (با [0] و رد کردن ۲۸ بایت اول که مربوط به هدر می باشد)

۴ بایتی که مربوط به شماره پورت هستند را با struct.unpack از result میگیریم و نشان می دهیم.

```

In [1]: import socket
import struct

In [2]: # Code to send a DNS query to a specified DNS server
# DNS server: Google's public DNS server at IP address 8.8.8.8 and port 53

def structDNSQuery(host):
    # Fields in DNS query header
    q = struct.pack("!HHHHH", 0x1234, 0x0100, 0x0001, 0x0000, 0x0000, 0x0000)
    hostParts = host.split('.')
    for part in hostParts:
        q += struct.pack('B', len(part))
        q += part.encode('utf-8')
    q += struct.pack('B', 0) # Append a null byte
    # Query type and query class (type A and IN class)
    q += struct.pack('!HH', 0x0001, 0x0001)
    return q

# Creates a UDP socket (DNS typically uses UDP for communication)
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Generates DNS query and sends it
dnsQ = structDNSQuery('google.com')
sock.sendto(dnsQ, ('8.8.8.8', 53)) # DNS server and DNS Port
# [0] extracts the recieved data and [28:] is for skipping the header
result = sock.recvfrom(1024)[0][28:]
if len(result) >= 16:
    address = struct.unpack('BBBB', result[12:16])
    finalResult = '.'.join(map(str, address)) # Final IP address
    print(finalResult)
else:
    print('Length exceeded.')
print("Closing socket")
sock.close()

216.239.38.120
Closing socket

```

شکل ۱۵ – کد پایتون برای ایجاد DNS Query و گرفتن IP گوگل

### بررسی پیام ارسال شده و response با Wireshark

همانطور که در شکل ۱۶ مشاهده میشود، Transaction ID و Desntation Port و ۱۶ مشاهده میشود، همانطور که مقادیر آنها در کد مشخص شده بود به ترتیب 8.8.8.8 و 0x1234 هستند.

مقدار Questions (تعداد سئوالات)، ۱ قرار داده شده و بیت Recursion Desired یک گذاشته شده که به معنی انجام کوئری به شکل بازگشتی است.

در قسمت Queries می‌توان query نوشته شده google.com با تایپ A و کلاس IN را مشاهده کرد.

همچنین در خط پایین نوشته شده که response به این پیام در No. 2012 قرار دارد.



بررسی شکل ۱۷ (response برای درخواست آدرس IP):

از port 8.8.8.8 (سمت گوگل) به port 192.168.1.202 (سمت سیستم) فرستاده شده. مقدار Transmission ID مقدار 0x1234 که تعیین شده بود می‌باشد.

مقدار قسمت Flags برابر 0x8180 است که با بررسی مقادیر آن، بیت response که پاسخ بودن این پیام را مشخص می‌کند ۱ گذاشته شده. بیت‌های مربوط به بازگشتی بودن query نیز ۱ هستند.

تعداد پاسخ‌ها (Answer RRs) و تعداد سئوالات (Questions) ۱ هستند.

در قسمت Answer، مشاهده می‌کنیم:

google.com: type A, class IN, addr **216.239.38.120**

در این بخش آدرس IP گوگل مشخص شده و همان آدرسی می‌باشد که توانستیم با کد بدست آوریم.