Mariam Hamada Meky – 7072

Youssed Mohamed Ahmed – 7211

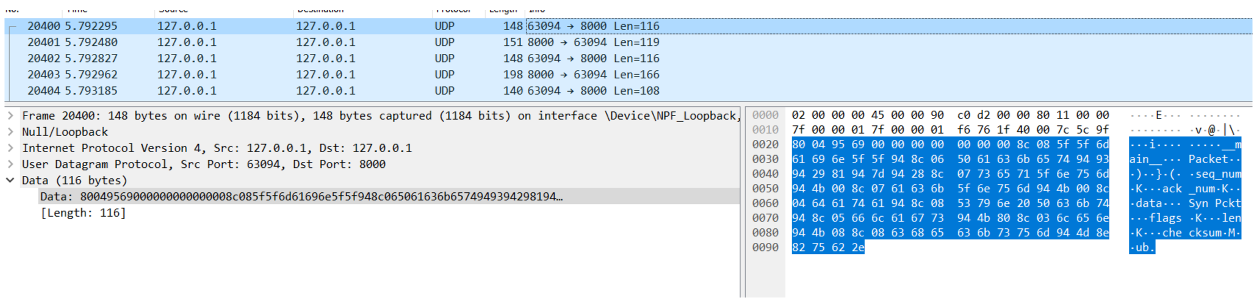
Computer Networks

FINAL LAB

IN the tcp-like stream:

1) The client starts an http connection with the server using the ***3-way handshake*** connection:

1- The client sends a SYN packet and seq number to the server to initiate the connection

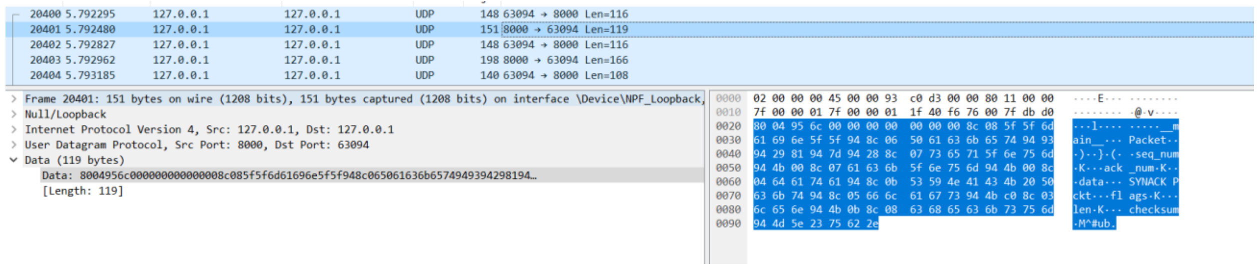


```
 20400 5.792295   127.0.0.1       127.0.0.1       UDP     148 63094 → 8000 Len=116
 20401 5.792480   127.0.0.1       127.0.0.1       UDP     151 8000 → 63094 Len=119
 20402 5.792827   127.0.0.1       127.0.0.1       UDP     148 63094 → 8000 Len=116
 20403 5.792962   127.0.0.1       127.0.0.1       UDP     198 8000 → 63094 Len=166
 20404 5.793185   127.0.0.1       127.0.0.1       UDP     140 63094 → 8000 Len=108

> Frame 20400: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 63094, Dst Port: 8000
v Data (116 bytes)
    Data: 800495690000000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 116]

0000  02 00 00 00 45 00 00 90  c0 d2 00 00 80 11 00 00   ····E··· ········
0010  7f 00 00 01 7f 00 00 01  f6 76 1f 40 00 7c 5c 9f   ········ ·v·@·|\·
0020  80 04 95 69 00 00 00 00  00 00 00 8c 08 5f 5f 6d   ···i···· ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 00 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 8c 08  53 79 6e 20 50 63 6b 74   ·data··· Syn Pckt
0070  94 8c 05 66 6c 61 67 73  94 4b 80 8c 03 6c 65 6e   ···flags ·K···len
0080  94 4b 08 8c 08 63 68 65  63 6b 73 75 6d 94 4d 8e   ·K···che cksum·M·
0090  82 75 62 2e                                        ·ub.
```

2- The server responds with a SYN-ACK packet to confirm receipt of the client's SYN packet and provide its own sequence number

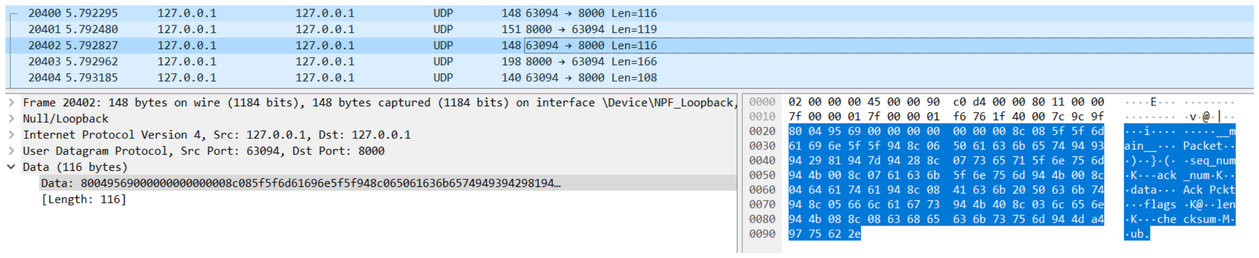


```
 20400 5.792295   127.0.0.1       127.0.0.1       UDP     148 63094 → 8000 Len=116
 20401 5.792480   127.0.0.1       127.0.0.1       UDP     151 8000 → 63094 Len=119
 20402 5.792827   127.0.0.1       127.0.0.1       UDP     148 63094 → 8000 Len=116
 20403 5.792962   127.0.0.1       127.0.0.1       UDP     198 8000 → 63094 Len=166
 20404 5.793185   127.0.0.1       127.0.0.1       UDP     140 63094 → 8000 Len=108

> Frame 20401: 151 bytes on wire (1208 bits), 151 bytes captured (1208 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 8000, Dst Port: 63094
v Data (119 bytes)
    Data: 8004956c000000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 119]

0000  02 00 00 00 45 00 00 93  c0 d3 00 00 80 11 00 00   ····E··· ········
0010  7f 00 00 01 7f 00 00 01  1f 40 f6 76 00 7f db d0   ········ ·@·v····
0020  80 04 95 6c 00 00 00 00  00 00 00 8c 08 5f 5f 6d   ···l···· ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 00 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 8c 0b  53 59 4e 41 43 4b 20 50   ·data··· SYNACK P
0070  63 6b 74 94 8c 05 66 6c  61 67 73 94 4b c0 8c 03   ckt···fl ags·K···
0080  6c 65 6e 94 4b 0b 8c 08  63 68 65 63 6b 73 75 6d   len·K··· checksum
0090  94 4d 5e 23 75 62 2e                               ·M^#ub.
```

3- The client sends an ACK (acknowledge) packet to confirm receipt of the server's SYN-ACK packet and complete the connection establishment.

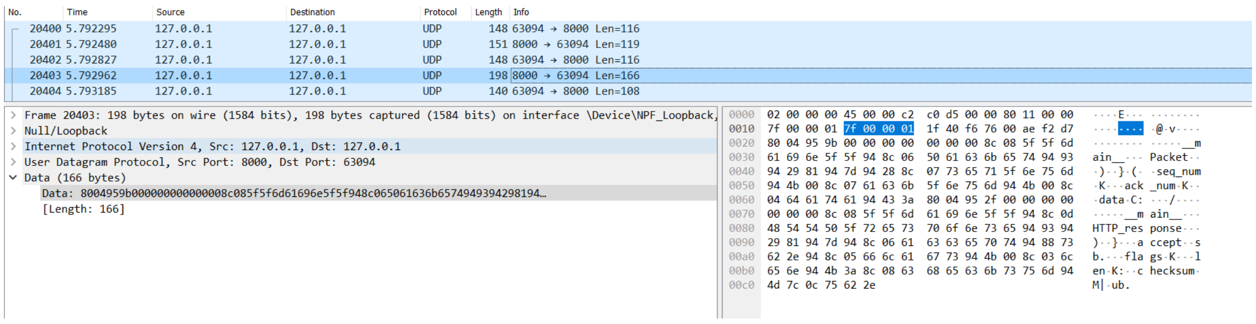


```
 20400 5.792295   127.0.0.1       127.0.0.1       UDP     148 63094 → 8000 Len=116
 20401 5.792480   127.0.0.1       127.0.0.1       UDP     151 8000 → 63094 Len=119
 20402 5.792827   127.0.0.1       127.0.0.1       UDP     148 63094 → 8000 Len=116
 20403 5.792962   127.0.0.1       127.0.0.1       UDP     198 8000 → 63094 Len=166
 20404 5.793185   127.0.0.1       127.0.0.1       UDP     140 63094 → 8000 Len=108

> Frame 20402: 148 bytes on wire (1184 bits), 148 bytes captured (1184 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 63094, Dst Port: 8000
v Data (116 bytes)
    Data: 800495690000000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 116]

0000  02 00 00 00 45 00 00 90  c0 d4 00 00 80 11 00 00   ····E··· ········
0010  7f 00 00 01 7f 00 00 01  f6 76 1f 40 00 7c 9c 9f   ········ ·v·@·|··
0020  80 04 95 69 00 00 00 00  00 00 00 8c 08 5f 5f 6d   ···i···· ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 00 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 8c 08  41 63 6b 20 50 63 6b 74   ·data··· Ack Pckt
0070  94 8c 05 66 6c 61 67 73  94 4b 40 8c 03 6c 65 6e   ···flags ·K@··len
0080  94 4b 08 8c 08 63 68 65  63 6b 73 75 6d 94 4d a4   ·K···che cksum·M·
0090  97 75 62 2e                                        ·ub.
```

2) The http server accepts connection notifying client:

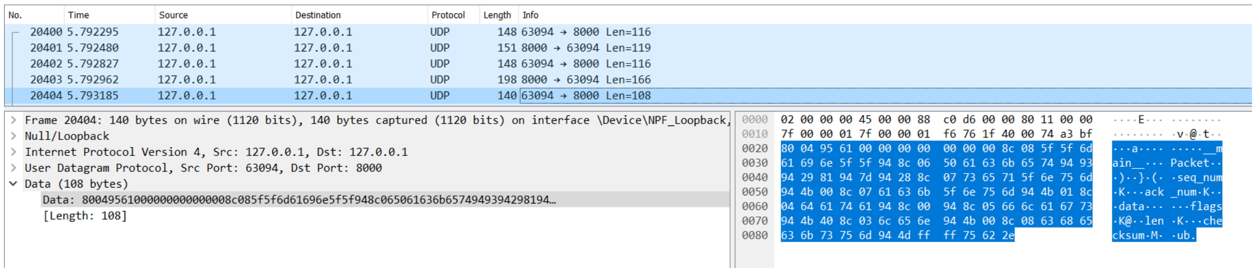


```
No.    Time       Source          Destination     Protocol Length Info
 20400 5.792295   127.0.0.1       127.0.0.1       UDP     148 63094 → 8000 Len=116
 20401 5.792480   127.0.0.1       127.0.0.1       UDP     151 8000 → 63094 Len=119
 20402 5.792827   127.0.0.1       127.0.0.1       UDP     148 63094 → 8000 Len=116
 20403 5.792962   127.0.0.1       127.0.0.1       UDP     198 8000 → 63094 Len=166
 20404 5.793185   127.0.0.1       127.0.0.1       UDP     140 63094 → 8000 Len=108

> Frame 20403: 198 bytes on wire (1584 bits), 198 bytes captured (1584 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 8000, Dst Port: 63094
v Data (166 bytes)
    Data: 8004959b000000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 166]

0000  02 00 00 00 45 00 00 c2  c0 d5 00 00 80 11 00 00   ····E··· ········
0010  7f 00 00 01 7f 00 00 01  1f 40 f6 76 00 ae f2 d7   ········ ·@·v····
0020  80 04 95 9b 00 00 00 00  00 00 00 8c 08 5f 5f 6d   ········ ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 00 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 43 3a  80 04 95 2f 00 00 00 00   ·data·C: ···/····
0070  00 00 00 8c 08 5f 5f 6d  61 69 6e 5f 5f 94 8c 0d   ·····__m ain__···
0080  48 54 54 50 5f 72 65 73  70 6f 6e 73 65 94 93 94   HTTP_res ponse···
0090  29 81 94 7d 94 8c 06 61  63 63 65 70 74 94 88 73   )··}···a ccept··s
00a0  62 2e 94 8c 05 66 6c 61  67 73 94 4b 00 8c 03 6c   b.···fla gs·K···l
00b0  65 6e 94 4b 3a 8c 08 63  68 65 63 6b 73 75 6d 94   en·K:··c hecksum·
00c0  4d 7c 0c 75 62 2e                                  M|·ub.
```

3) Clients sends an ack:

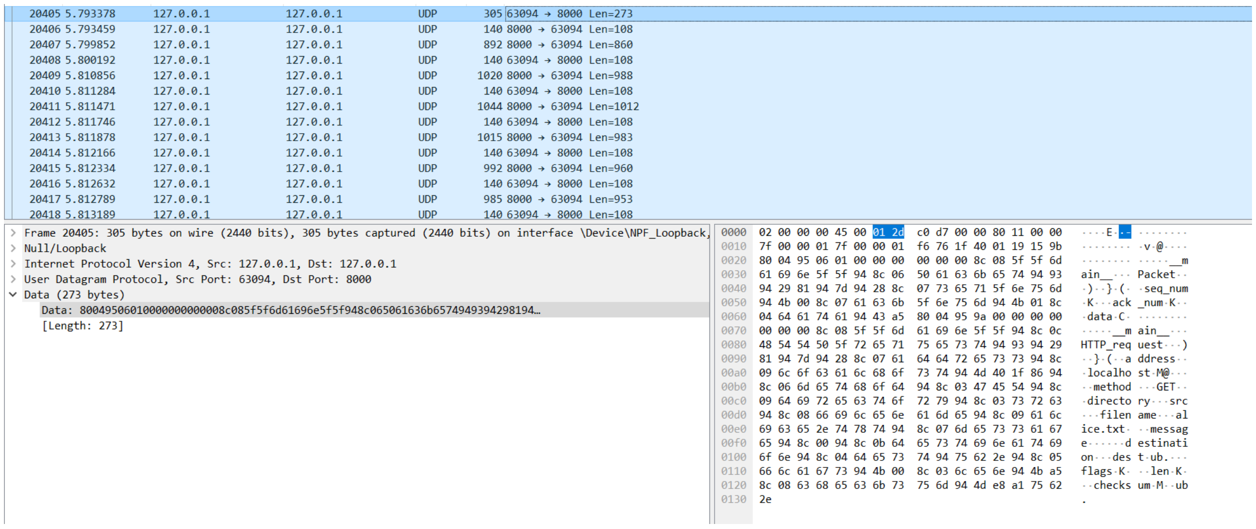


```
No.    Time      Source      Destination   Protocol  Length Info
20400 5.792295   127.0.0.1   127.0.0.1     UDP       148 63094 → 8000 Len=116
20401 5.792480   127.0.0.1   127.0.0.1     UDP       151 8000 → 63094 Len=119
20402 5.792827   127.0.0.1   127.0.0.1     UDP       148 63094 → 8000 Len=116
20403 5.792962   127.0.0.1   127.0.0.1     UDP       198 8000 → 63094 Len=166
20404 5.793185   127.0.0.1   127.0.0.1     UDP       140 63094 → 8000 Len=108
```

```
> Frame 20404: 140 bytes on wire (1120 bits), 140 bytes captured (1120 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 63094, Dst Port: 8000
v Data (108 bytes)
    Data: 8004956100000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 108]
```

```
0000  02 00 00 00 45 00 00 88  c0 d6 00 00 80 11 00 00   ····E··· ········
0010  7f 00 00 01 7f 00 00 01  f6 76 1f 40 00 74 a3 bf   ········ ·v·@·t··
0020  80 04 95 61 00 00 00 00  00 00 00 8c 08 5f 5f 6d   ···a···· ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 01 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 8c 00  94 8c 05 66 6c 61 67 73   ·data··· ···flags
0070  94 4b 40 8c 03 6c 65 6e  94 4b 00 8c 08 63 68 65   ·K@··len ·K···che
0080  63 6b 73 75 6d 94 4d ff  ff 75 62 2e               cksum·M· ·ub.
```

4) Client then sends an HTTP request message to requests file:

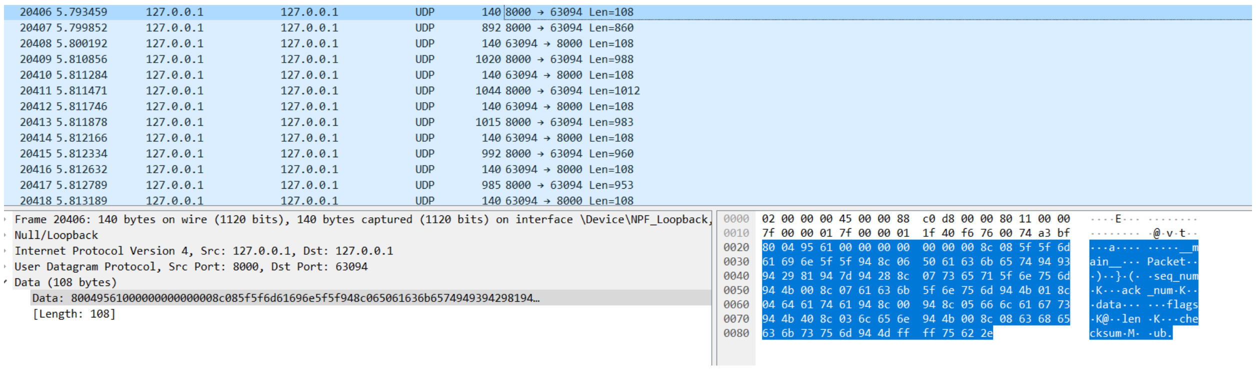


```
20405 5.793378   127.0.0.1   127.0.0.1   UDP    305 63094 → 8000 Len=273
20406 5.793459   127.0.0.1   127.0.0.1   UDP    140 8000 → 63094 Len=108
20407 5.799852   127.0.0.1   127.0.0.1   UDP    892 8000 → 63094 Len=860
20408 5.800192   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20409 5.810856   127.0.0.1   127.0.0.1   UDP   1020 8000 → 63094 Len=988
20410 5.811284   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20411 5.811471   127.0.0.1   127.0.0.1   UDP   1044 8000 → 63094 Len=1012
20412 5.811746   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20413 5.811878   127.0.0.1   127.0.0.1   UDP   1015 8000 → 63094 Len=983
20414 5.812166   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20415 5.812334   127.0.0.1   127.0.0.1   UDP    992 8000 → 63094 Len=960
20416 5.812632   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20417 5.812789   127.0.0.1   127.0.0.1   UDP    985 8000 → 63094 Len=953
20418 5.813189   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
```

```
> Frame 20405: 305 bytes on wire (2440 bits), 305 bytes captured (2440 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 63094, Dst Port: 8000
v Data (273 bytes)
    Data: 800495060100000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 273]
```

```
0000  02 00 00 00 45 00 01 2d  c0 d7 00 00 80 11 00 00   ····E·-- ········
0010  7f 00 00 01 7f 00 00 01  f6 76 1f 40 01 19 15 9b   ········ ·v·@····
0020  80 04 95 06 01 00 00 00  00 00 00 8c 08 5f 5f 6d   ········ ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 01 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 43 a5  80 04 95 9a 00 00 00 00   ·data·C· ········
0070  00 00 00 8c 08 5f 5f 6d  61 69 6e 5f 5f 94 8c 0c   ·····__m ain__···
0080  48 54 54 50 5f 72 65 71  75 65 73 74 94 93 94 29   HTTP_req uest···)
0090  81 94 7d 94 28 8c 07 61  64 64 72 65 73 73 94 8c   ··}·(··a ddress··
00a0  09 6c 6f 63 61 6c 68 6f  73 74 94 4d 40 1f 86 94   ·localho st·M@···
00b0  8c 06 6d 65 74 68 6f 64  94 8c 03 47 45 54 94 8c   ··method ···GET··
00c0  09 64 69 72 65 63 74 6f  72 79 94 8c 03 73 72 63   ·directo ry···src
00d0  94 8c 08 66 69 6c 65 6e  61 6d 65 94 8c 09 61 6c   ···filen ame···al
00e0  69 63 65 2e 74 78 74 94  8c 07 6d 65 73 73 61 67   ice.txt· ··messag
00f0  65 94 8c 00 94 8c 0b 64  65 73 74 69 6e 61 74 69   e······d estinati
0100  6f 6e 94 8c 04 64 65 73  74 94 75 62 2e 94 8c 05   on···des t·ub.···
0110  66 6c 61 67 73 94 4b 00  8c 03 6c 65 6e 94 4b a5   flags·K· ··len·K·
0120  8c 08 63 68 65 63 6b 73  75 6d 94 4d e8 a1 75 62   ··checks um·M··ub
0130  2e                                                 .
```

5) Server sends ack to confirm it will start sending file:

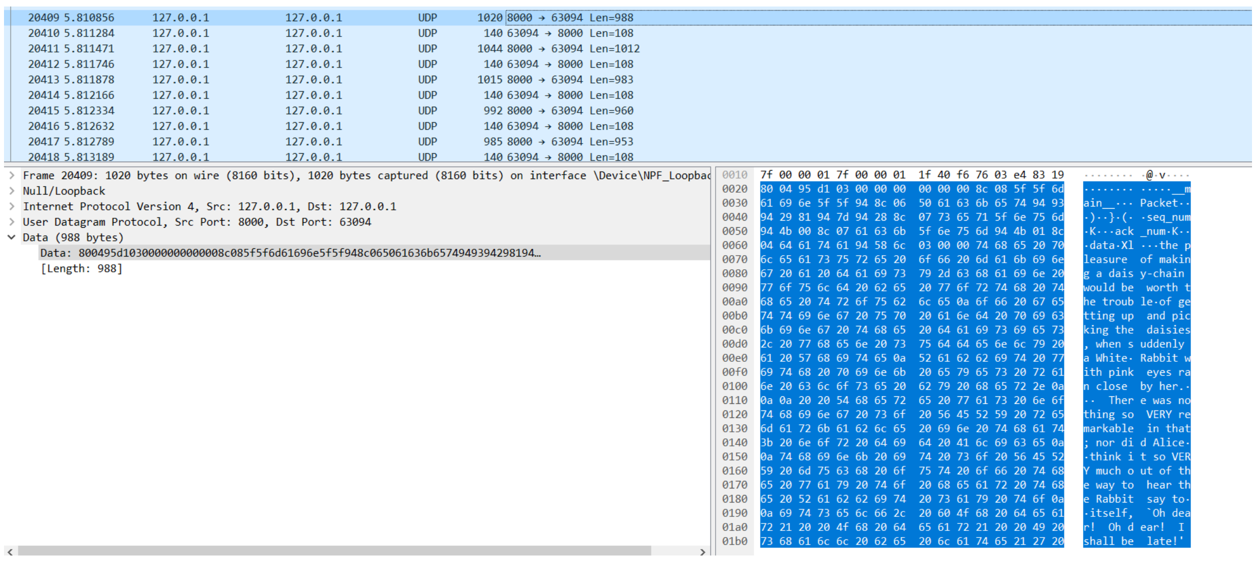


```
20406 5.793459   127.0.0.1   127.0.0.1   UDP    140 8000 → 63094 Len=108
20407 5.799852   127.0.0.1   127.0.0.1   UDP    892 8000 → 63094 Len=860
20408 5.800192   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20409 5.810856   127.0.0.1   127.0.0.1   UDP   1020 8000 → 63094 Len=988
20410 5.811284   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20411 5.811471   127.0.0.1   127.0.0.1   UDP   1044 8000 → 63094 Len=1012
20412 5.811746   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20413 5.811878   127.0.0.1   127.0.0.1   UDP   1015 8000 → 63094 Len=983
20414 5.812166   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20415 5.812334   127.0.0.1   127.0.0.1   UDP    992 8000 → 63094 Len=960
20416 5.812632   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20417 5.812789   127.0.0.1   127.0.0.1   UDP    985 8000 → 63094 Len=953
20418 5.813189   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
```

```
Frame 20406: 140 bytes on wire (1120 bits), 140 bytes captured (1120 bits) on interface \Device\NPF_Loopback,
Null/Loopback
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
User Datagram Protocol, Src Port: 8000, Dst Port: 63094
Data (108 bytes)
  Data: 8004956100000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
  [Length: 108]
```

```
0000  02 00 00 00 45 00 00 88  c0 d8 00 00 80 11 00 00   ····E··· ········
0010  7f 00 00 01 7f 00 00 01  1f 40 f6 76 00 74 a3 bf   ········ ·@·v·t··
0020  80 04 95 61 00 00 00 00  00 00 00 8c 08 5f 5f 6d   ···a···· ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 01 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 8c 00  94 8c 05 66 6c 61 67 73   ·data··· ···flags
0070  94 4b 40 8c 03 6c 65 6e  94 4b 00 8c 08 63 68 65   ·K@··len ·K···che
0080  63 6b 73 75 6d 94 4d ff  ff 75 62 2e               cksum·M· ·ub.
```

6) Server sends response message containing header and first part of the message that can fit in the available buffer space (1024) :

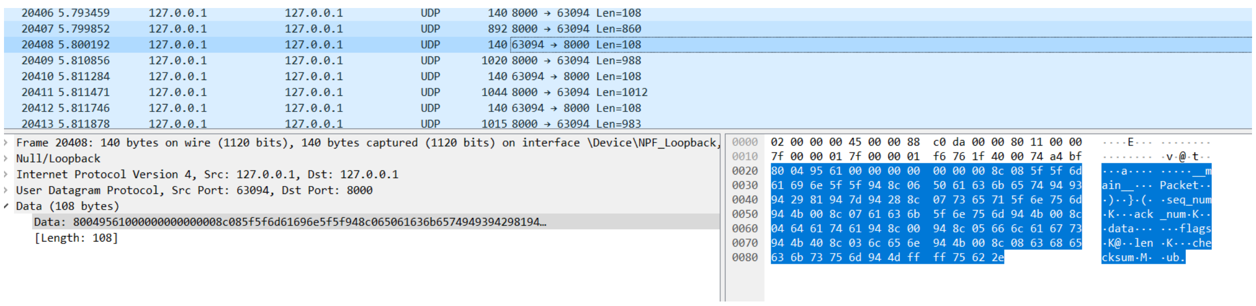


```
20409 5.810856   127.0.0.1   127.0.0.1   UDP   1020 8000 → 63094 Len=988
20410 5.811284   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20411 5.811471   127.0.0.1   127.0.0.1   UDP   1044 8000 → 63094 Len=1012
20412 5.811746   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20413 5.811878   127.0.0.1   127.0.0.1   UDP   1015 8000 → 63094 Len=983
20414 5.812166   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20415 5.812334   127.0.0.1   127.0.0.1   UDP    992 8000 → 63094 Len=960
20416 5.812632   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20417 5.812789   127.0.0.1   127.0.0.1   UDP    985 8000 → 63094 Len=953
20418 5.813189   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108

> Frame 20409: 1020 bytes on wire (8160 bits), 1020 bytes captured (8160 bits) on interface \Device\NPF_Loopbac
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 8000, Dst Port: 63094
v Data (988 bytes)
    Data: 800495d1030000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 988]

0010  7f 00 00 01 7f 00 00 01  1f 40 f6 76 03 e4 83 19   ········ ·@·v····
0020  80 04 95 d1 03 00 00 00  00 00 00 8c 08 5f 5f 6d   ········ ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 01 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 58 6c  03 00 00 74 68 65 20 70   ·data·Xl ···the p
0070  6c 65 61 73 75 72 65 20  6f 66 20 6d 61 6b 69 6e   leasure  of makin
0080  67 20 61 20 64 61 69 73  79 2d 63 68 61 69 6e 20   g a dais y-chain 
0090  77 6f 75 6c 64 20 62 65  20 77 6f 72 74 68 20 74   would be  worth t
00a0  68 65 20 74 72 6f 75 62  6c 65 0a 6f 66 20 67 65   he troub le·of ge
00b0  74 74 69 6e 67 20 75 70  20 61 6e 64 20 70 69 63   tting up  and pic
00c0  6b 69 6e 67 20 74 68 65  20 64 61 69 73 69 65 73   king the  daisies
00d0  2c 20 77 68 65 6e 20 73  75 64 64 65 6e 6c 79 20   , when s uddenly 
00e0  61 20 57 68 69 74 65 0a  52 61 62 62 69 74 20 77   a White· Rabbit w
00f0  69 74 68 20 70 69 6e 6b  20 65 79 65 73 20 72 61   ith pink  eyes ra
0100  6e 20 63 6c 6f 73 65 20  62 79 20 68 65 72 2e 0a   n close  by her.·
0110  0a 0a 20 20 54 68 65 72  65 20 77 61 73 20 6e 6f   ··  Ther e was no
0120  74 68 69 6e 67 20 73 6f  20 56 45 52 59 20 72 65   thing so  VERY re
0130  6d 61 72 6b 61 62 6c 65  20 69 6e 20 74 68 61 74   markable  in that
0140  3b 20 6e 6f 72 20 64 69  64 20 41 6c 69 63 65 0a   ; nor di d Alice·
0150  0a 74 68 69 6e 6b 20 69  74 20 73 6f 20 56 45 52   ·think i t so VER
0160  59 20 6d 75 63 68 20 6f  75 74 20 6f 66 20 74 68   Y much o ut of th
0170  65 20 77 61 79 20 74 6f  20 68 65 61 72 20 74 68   e way to  hear th
0180  65 20 52 61 62 62 69 74  20 73 61 79 20 74 6f 0a   e Rabbit  say to·
0190  0a 69 74 73 65 6c 66 2c  20 60 4f 68 20 64 65 61   ·itself,  `Oh dea
01a0  72 21 20 20 4f 68 20 64  65 61 72 21 20 20 49 20   r!  Oh d ear!  I 
01b0  73 68 61 6c 6c 20 62 65  20 6c 61 74 65 21 27 20   shall be  late!' 
```

7) The client acks to confirm:

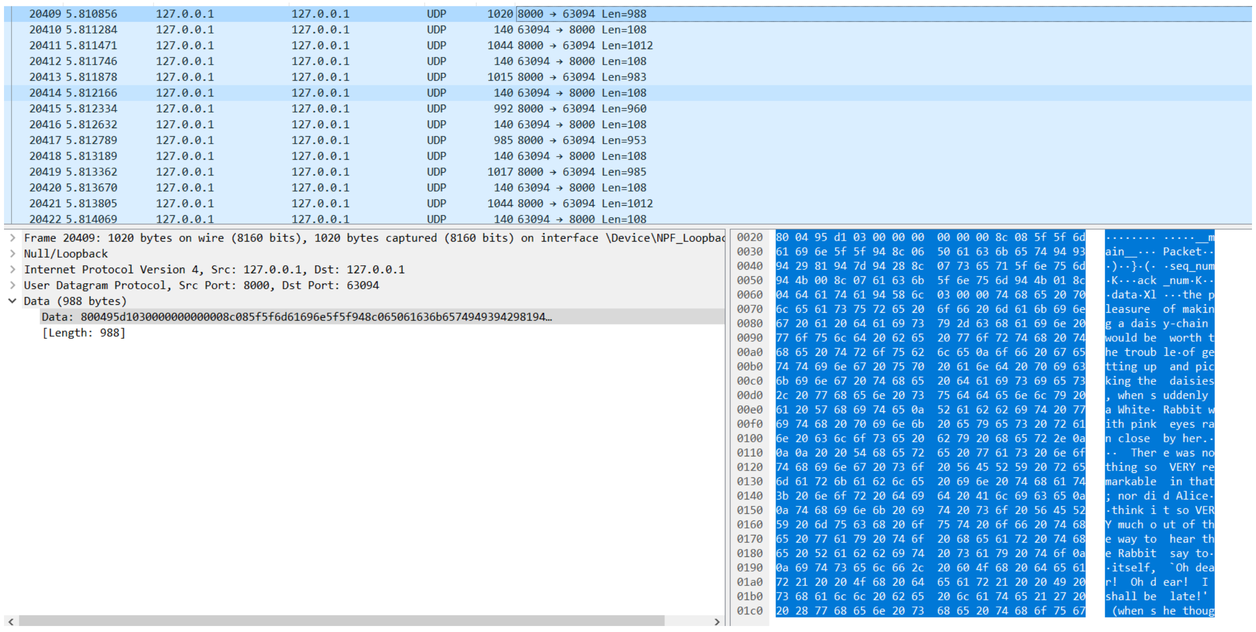


```
20406 5.793459   127.0.0.1   127.0.0.1   UDP    140 8000 → 63094 Len=108
20407 5.799852   127.0.0.1   127.0.0.1   UDP    892 8000 → 63094 Len=860
20408 5.800192   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20409 5.810856   127.0.0.1   127.0.0.1   UDP   1020 8000 → 63094 Len=988
20410 5.811284   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20411 5.811471   127.0.0.1   127.0.0.1   UDP   1044 8000 → 63094 Len=1012
20412 5.811746   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20413 5.811878   127.0.0.1   127.0.0.1   UDP   1015 8000 → 63094 Len=983

> Frame 20408: 140 bytes on wire (1120 bits), 140 bytes captured (1120 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 63094, Dst Port: 8000
v Data (108 bytes)
    Data: 80049561000000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 108]

0000  02 00 00 00 45 00 00 88  c0 da 00 00 80 11 00 00   ····E··· ········
0010  7f 00 00 01 7f 00 00 01  f6 76 1f 40 00 74 a4 bf   ········ ·v·@·t··
0020  80 04 95 61 00 00 00 00  00 00 00 8c 08 5f 5f 6d   ···a···· ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 00 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 8c 00  94 8c 05 66 6c 61 67 73   ·data··· ···flags
0070  94 4b 40 8c 03 6c 65 6e  94 4b 00 8c 08 63 68 65   ·K@··len ·K···che
0080  63 6b 73 75 6d 94 4d ff  ff 75 62 2e               cksum·M· ·ub.
```

8) server keeps sending the rest of the file in messages and client keeps on acking each packet to confirm it’s sent, until all the file is sent successfully:

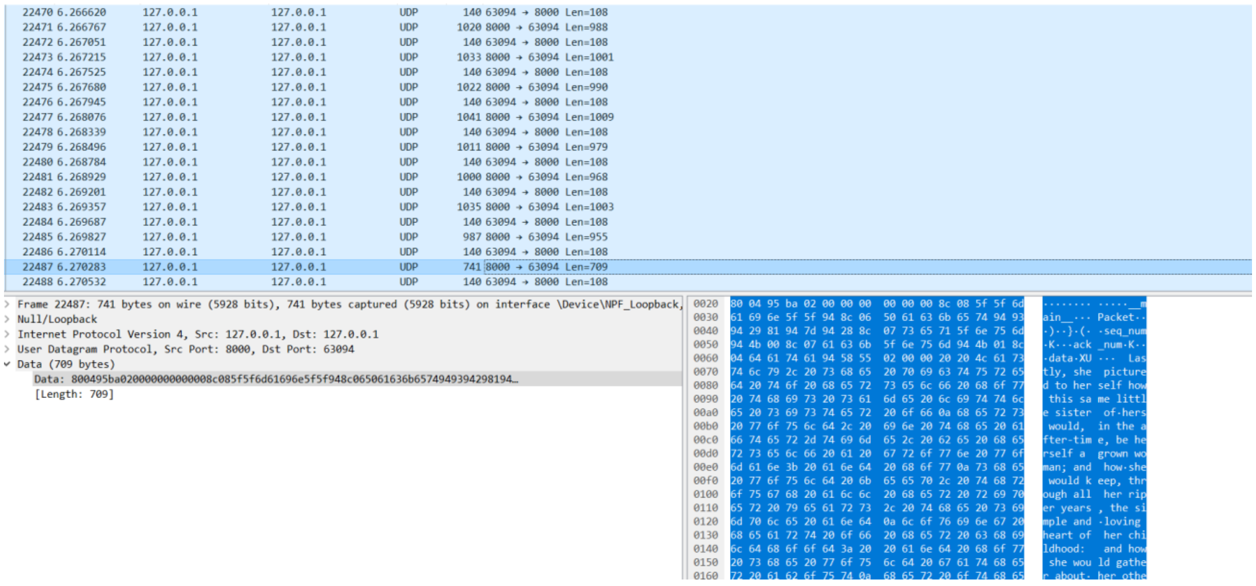


```
20409 5.810856   127.0.0.1   127.0.0.1   UDP   1020 8000 → 63094 Len=988
20410 5.811284   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20411 5.811471   127.0.0.1   127.0.0.1   UDP   1044 8000 → 63094 Len=1012
20412 5.811746   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20413 5.811878   127.0.0.1   127.0.0.1   UDP   1015 8000 → 63094 Len=983
20414 5.812166   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20415 5.812334   127.0.0.1   127.0.0.1   UDP    992 8000 → 63094 Len=960
20416 5.812632   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20417 5.812789   127.0.0.1   127.0.0.1   UDP    985 8000 → 63094 Len=953
20418 5.813189   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20419 5.813362   127.0.0.1   127.0.0.1   UDP   1017 8000 → 63094 Len=985
20420 5.813670   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108
20421 5.813805   127.0.0.1   127.0.0.1   UDP   1044 8000 → 63094 Len=1012
20422 5.814069   127.0.0.1   127.0.0.1   UDP    140 63094 → 8000 Len=108

> Frame 20409: 1020 bytes on wire (8160 bits), 1020 bytes captured (8160 bits) on interface \Device\NPF_Loopbac
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 8000, Dst Port: 63094
v Data (988 bytes)
    Data: 800495d1030000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 988]

0020  80 04 95 d1 03 00 00 00  00 00 00 8c 08 5f 5f 6d   ········ ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 01 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 58 6c  03 00 00 74 68 65 20 70   ·data·Xl ···the p
0070  6c 65 61 73 75 72 65 20  6f 66 20 6d 61 6b 69 6e   leasure  of makin
0080  67 20 61 20 64 61 69 73  79 2d 63 68 61 69 6e 20   g a dais y-chain 
0090  77 6f 75 6c 64 20 62 65  20 77 6f 72 74 68 20 74   would be  worth t
00a0  68 65 20 74 72 6f 75 62  6c 65 0a 6f 66 20 67 65   he troub le·of ge
00b0  74 74 69 6e 67 20 75 70  20 61 6e 64 20 70 69 63   tting up  and pic
00c0  6b 69 6e 67 20 74 68 65  20 64 61 69 73 69 65 73   king the  daisies
00d0  2c 20 77 68 65 6e 20 73  75 64 64 65 6e 6c 79 20   , when s uddenly 
00e0  61 20 57 68 69 74 65 0a  52 61 62 62 69 74 20 77   a White· Rabbit w
00f0  69 74 68 20 70 69 6e 6b  20 65 79 65 73 20 72 61   ith pink  eyes ra
0100  6e 20 63 6c 6f 73 65 20  62 79 20 68 65 72 2e 0a   n close  by her.·
0110  0a 0a 20 20 54 68 65 72  65 20 77 61 73 20 6e 6f   ··  Ther e was no
0120  74 68 69 6e 67 20 73 6f  20 56 45 52 59 20 72 65   thing so  VERY re
0130  6d 61 72 6b 61 62 6c 65  20 69 6e 20 74 68 61 74   markable  in that
0140  3b 20 6e 6f 72 20 64 69  64 20 41 6c 69 63 65 0a   ; nor di d Alice·
0150  0a 74 68 69 6e 6b 20 69  74 20 73 6f 20 56 45 52   ·think i t so VER
0160  59 20 6d 75 63 68 20 6f  75 74 20 6f 66 20 74 68   Y much o ut of th
0170  65 20 77 61 79 20 74 6f  20 68 65 61 72 20 74 68   e way to  hear th
0180  65 20 52 61 62 62 69 74  20 73 61 79 20 74 6f 0a   e Rabbit  say to·
0190  0a 69 74 73 65 6c 66 2c  20 60 4f 68 20 64 65 61   ·itself,  `Oh dea
01a0  72 21 20 20 4f 68 20 64  65 61 72 21 20 20 49 20   r!  Oh d ear!  I 
01b0  73 68 61 6c 6c 20 62 65  20 6c 61 74 65 21 27 20   shall be  late!' 
01c0  20 28 77 68 65 6e 20 73  68 65 20 74 68 6f 75 67    (when s he thoug
```

Process repeats as shown below and this is the last part of the file being sent:



```
22470 6.266620   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22471 6.266767   127.0.0.1   127.0.0.1   UDP  1020 8000 → 63094 Len=988
22472 6.267051   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22473 6.267215   127.0.0.1   127.0.0.1   UDP  1033 8000 → 63094 Len=1001
22474 6.267525   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22475 6.267680   127.0.0.1   127.0.0.1   UDP  1022 8000 → 63094 Len=990
22476 6.267945   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22477 6.268076   127.0.0.1   127.0.0.1   UDP  1041 8000 → 63094 Len=1009
22478 6.268339   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22479 6.268496   127.0.0.1   127.0.0.1   UDP  1011 8000 → 63094 Len=979
22480 6.268784   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22481 6.268929   127.0.0.1   127.0.0.1   UDP  1000 8000 → 63094 Len=968
22482 6.269201   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22483 6.269357   127.0.0.1   127.0.0.1   UDP  1035 8000 → 63094 Len=1003
22484 6.269687   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22485 6.269827   127.0.0.1   127.0.0.1   UDP   987 8000 → 63094 Len=955
22486 6.270114   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22487 6.270283   127.0.0.1   127.0.0.1   UDP   741 8000 → 63094 Len=709
22488 6.270532   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
```

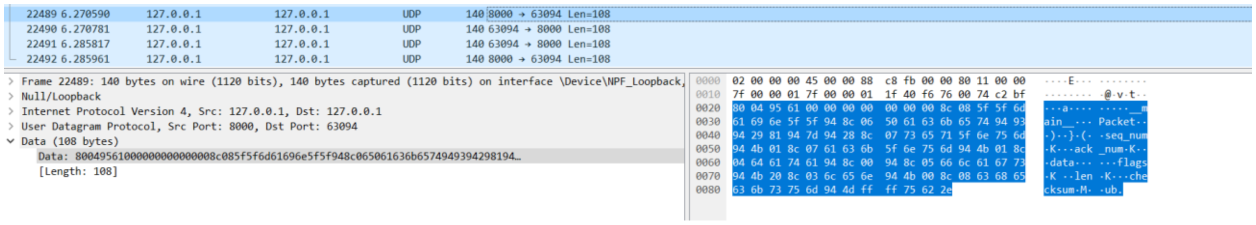
```
> Frame 22487: 741 bytes on wire (5928 bits), 741 bytes captured (5928 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 8000, Dst Port: 63094
v Data (709 bytes)
    Data: 800495ba020000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 709]
```

```
0020  80 04 95 ba 02 00 00 00  00 00 00 8c 08 5f 5f 6d   ········ ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 00 8c 07 61 63 6b  5f 6e 75 6d 94 4b 01 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 58 55  02 00 00 20 20 4c 61 73   ·data·XU ···  Las
0070  74 6c 79 2c 20 73 68 65  20 70 69 63 74 75 72 65   tly, she  picture
0080  64 20 74 6f 20 68 65 72  73 65 6c 66 20 68 6f 77   d to her self how
0090  20 74 68 69 73 20 73 61  6d 65 20 6c 69 74 74 6c    this sa me littl
00a0  65 20 73 69 73 74 65 72  20 6f 66 0a 68 65 72 73   e sister  of·hers
00b0  20 77 6f 75 6c 64 2c 20  69 6e 20 74 68 65 20 61    would,  in the a
00c0  66 74 65 72 2d 74 69 6d  65 2c 20 62 65 20 68 65   fter-tim e, be he
00d0  72 73 65 6c 66 20 61 20  67 72 6f 77 6e 20 77 6f   rself a  grown wo
00e0  6d 61 6e 3b 20 61 6e 64  20 68 6f 77 0a 73 68 65   man; and  how·she
00f0  20 77 6f 75 6c 64 20 6b  65 65 70 2c 20 74 68 72    would k eep, thr
0100  6f 75 67 68 20 61 6c 6c  20 68 65 72 20 72 69 70   ough all  her rip
0110  65 72 20 79 65 61 72 73  2c 20 74 68 65 20 73 69   er years , the si
0120  6d 70 6c 65 20 61 6e 64  0a 6c 6f 76 69 6e 67 20   mple and ·loving
0130  68 65 61 72 74 20 6f 66  20 68 65 72 20 63 68 69   heart of  her chi
0140  6c 64 68 6f 6f 64 3a 20  20 61 6e 64 20 68 6f 77   ldhood:   and how
0150  20 73 68 65 20 77 6f 75  6c 64 20 67 61 74 68 65    she wou ld gathe
0160  72 20 61 62 6f 75 74 0a  68 65 72 20 6f 74 68 65   r about· her othe
```

9)Last part:

- The connection is closed when the server sends a FIN packet
- The client sends a FIN-ACK
- Then client send a FIN-Packet
- Finally, server sends a FIN0-ACK:

Shown below in the last 4 packets:



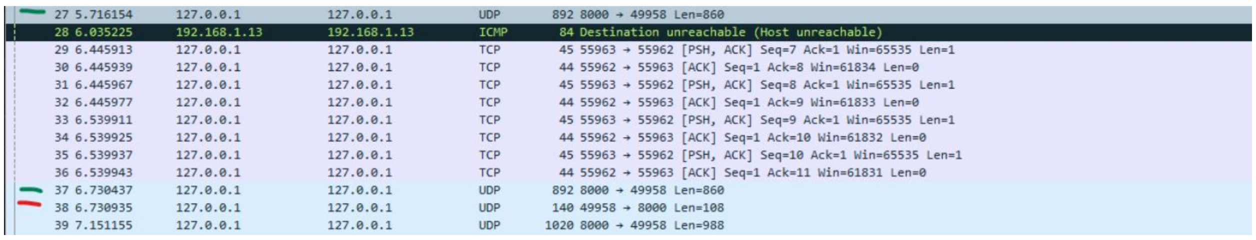
```
22489 6.270590   127.0.0.1   127.0.0.1   UDP   140 8000 → 63094 Len=108
22490 6.270781   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22491 6.285817   127.0.0.1   127.0.0.1   UDP   140 63094 → 8000 Len=108
22492 6.285961   127.0.0.1   127.0.0.1   UDP   140 8000 → 63094 Len=108
```

```
> Frame 22489: 140 bytes on wire (1120 bits), 140 bytes captured (1120 bits) on interface \Device\NPF_Loopback,
> Null/Loopback
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> User Datagram Protocol, Src Port: 8000, Dst Port: 63094
v Data (108 bytes)
    Data: 80049561000000000000008c085f5f6d61696e5f5f948c065061636b6574949394298194…
    [Length: 108]
```

```
0000  02 00 00 00 45 00 00 88  c8 fb 00 00 80 11 00 00   ····E··· ········
0010  7f 00 00 01 7f 00 00 01  1f 40 f6 76 00 74 c2 bf   ········ ·@·v·t··
0020  80 04 95 61 00 00 00 00  00 00 00 8c 08 5f 5f 6d   ···a···· ·····__m
0030  61 69 6e 5f 5f 94 8c 06  50 61 63 6b 65 74 94 93   ain__··· Packet··
0040  94 29 81 94 7d 94 28 8c  07 73 65 71 5f 6e 75 6d   ·)··}·(· ·seq_num
0050  94 4b 01 8c 07 61 63 6b  5f 6e 75 6d 94 4b 01 8c   ·K···ack _num·K··
0060  04 64 61 74 61 94 8c 00  94 8c 05 66 6c 61 67 73   ·data··· ···flags
0070  94 4b 20 8c 03 6c 65 6e  94 4b 00 8c 08 63 68 65   ·K ··len ·K···che
0080  63 6b 73 75 6d 94 4d ff  ff 75 62 2e               cksum·M· ·ub.
```

Another Scenario: Ack corrupted:

>> File packets are resent again as shown

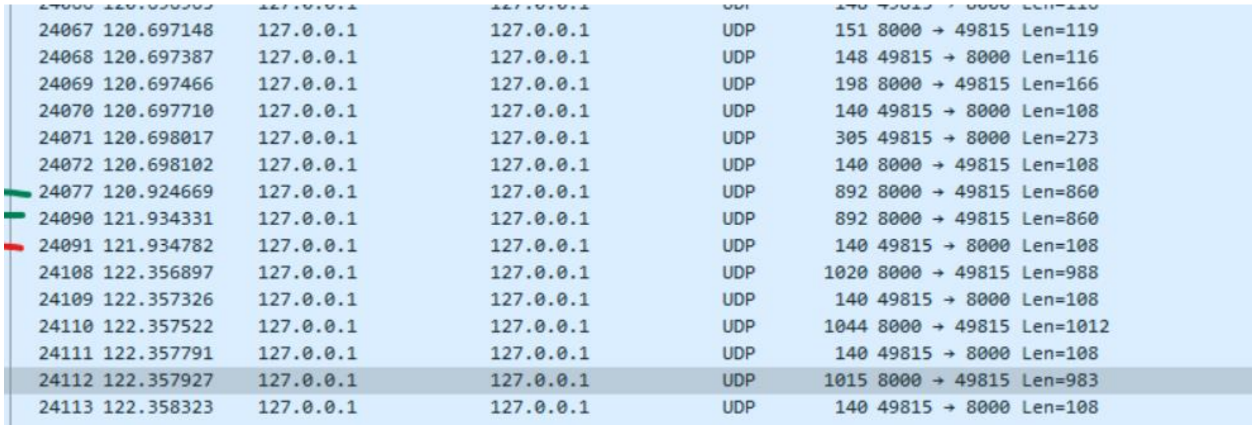


```
27 5.716154   127.0.0.1     127.0.0.1     UDP   892 8000 → 49958 Len=860
28 6.035225   192.168.1.13  192.168.1.13  ICMP   84 Destination unreachable (Host unreachable)
29 6.445913   127.0.0.1     127.0.0.1     TCP    45 55963 → 55962 [PSH, ACK] Seq=7 Ack=1 Win=65535 Len=1
30 6.445939   127.0.0.1     127.0.0.1     TCP    44 55962 → 55963 [ACK] Seq=1 Ack=8 Win=61834 Len=0
31 6.445967   127.0.0.1     127.0.0.1     TCP    45 55963 → 55962 [PSH, ACK] Seq=8 Ack=1 Win=65535 Len=1
32 6.445977   127.0.0.1     127.0.0.1     TCP    44 55962 → 55963 [ACK] Seq=1 Ack=9 Win=61833 Len=0
33 6.539911   127.0.0.1     127.0.0.1     TCP    45 55963 → 55962 [PSH, ACK] Seq=9 Ack=1 Win=65535 Len=1
34 6.539925   127.0.0.1     127.0.0.1     TCP    44 55962 → 55963 [ACK] Seq=1 Ack=10 Win=61832 Len=0
35 6.539937   127.0.0.1     127.0.0.1     TCP    45 55963 → 55962 [PSH, ACK] Seq=10 Ack=1 Win=65535 Len=1
36 6.539943   127.0.0.1     127.0.0.1     TCP    44 55962 → 55963 [ACK] Seq=1 Ack=11 Win=61831 Len=0
37 6.730437   127.0.0.1     127.0.0.1     UDP   892 8000 → 49958 Len=860
38 6.730935   127.0.0.1     127.0.0.1     UDP   140 49958 → 8000 Len=108
39 7.151155   127.0.0.1     127.0.0.1     UDP  1020 8000 → 49958 Len=988
```

Another Scenario: Packet Corrupted:

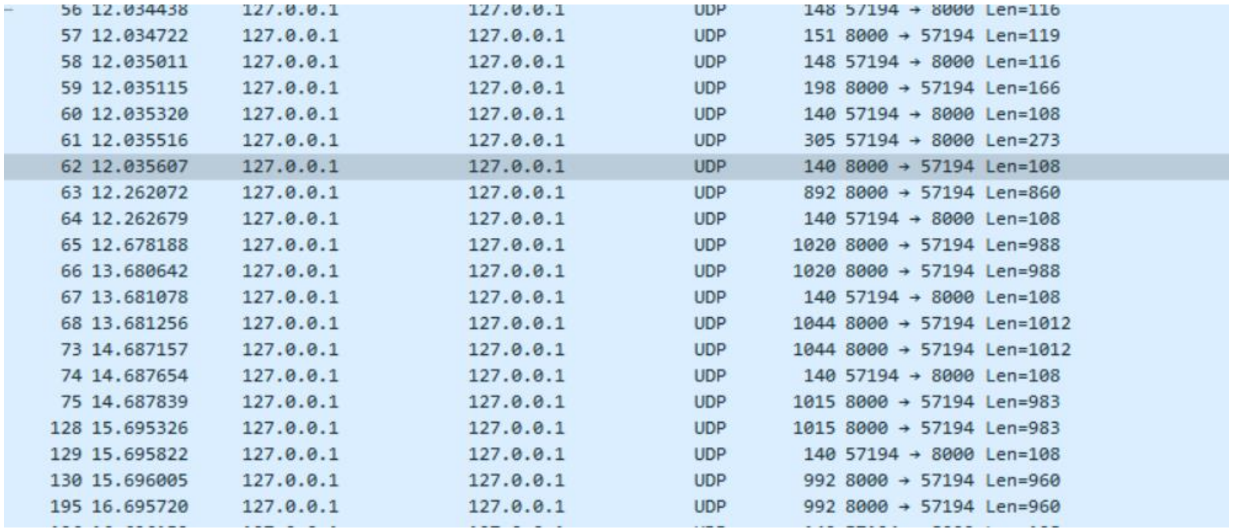
>> Client ignores the corrupted Packet

>> Waits for the following packet and then sends an ACK



```
24067 120.697148  127.0.0.1  127.0.0.1  UDP   151 8000 → 49815 Len=119
24068 120.697387  127.0.0.1  127.0.0.1  UDP   148 49815 → 8000 Len=116
24069 120.697466  127.0.0.1  127.0.0.1  UDP   198 8000 → 49815 Len=166
24070 120.697710  127.0.0.1  127.0.0.1  UDP   140 49815 → 8000 Len=108
24071 120.698017  127.0.0.1  127.0.0.1  UDP   305 49815 → 8000 Len=273
24072 120.698102  127.0.0.1  127.0.0.1  UDP   140 8000 → 49815 Len=108
24077 120.924669  127.0.0.1  127.0.0.1  UDP   892 8000 → 49815 Len=860
24090 121.934331  127.0.0.1  127.0.0.1  UDP   892 8000 → 49815 Len=860
24091 121.934782  127.0.0.1  127.0.0.1  UDP   140 49815 → 8000 Len=108
24108 122.356897  127.0.0.1  127.0.0.1  UDP  1020 8000 → 49815 Len=988
24109 122.357326  127.0.0.1  127.0.0.1  UDP   140 49815 → 8000 Len=108
24110 122.357522  127.0.0.1  127.0.0.1  UDP  1044 8000 → 49815 Len=1012
24111 122.357791  127.0.0.1  127.0.0.1  UDP   140 49815 → 8000 Len=108
24112 122.357927  127.0.0.1  127.0.0.1  UDP  1015 8000 → 49815 Len=983
24113 122.358323  127.0.0.1  127.0.0.1  UDP   140 49815 → 8000 Len=108
```

Another Scenario: each packet is lost the first time:



```
 56 12.034438  127.0.0.1  127.0.0.1  UDP   148 57194 → 8000 Len=116
 57 12.034722  127.0.0.1  127.0.0.1  UDP   151 8000 → 57194 Len=119
 58 12.035011  127.0.0.1  127.0.0.1  UDP   148 57194 → 8000 Len=116
 59 12.035115  127.0.0.1  127.0.0.1  UDP   198 8000 → 57194 Len=166
 60 12.035320  127.0.0.1  127.0.0.1  UDP   140 57194 → 8000 Len=108
 61 12.035516  127.0.0.1  127.0.0.1  UDP   305 57194 → 8000 Len=273
 62 12.035607  127.0.0.1  127.0.0.1  UDP   140 8000 → 57194 Len=108
 63 12.262072  127.0.0.1  127.0.0.1  UDP   892 8000 → 57194 Len=860
 64 12.262679  127.0.0.1  127.0.0.1  UDP   140 57194 → 8000 Len=108
 65 12.678188  127.0.0.1  127.0.0.1  UDP  1020 8000 → 57194 Len=988
 66 13.680642  127.0.0.1  127.0.0.1  UDP  1020 8000 → 57194 Len=988
 67 13.681078  127.0.0.1  127.0.0.1  UDP   140 57194 → 8000 Len=108
 68 13.681256  127.0.0.1  127.0.0.1  UDP  1044 8000 → 57194 Len=1012
 73 14.687157  127.0.0.1  127.0.0.1  UDP  1044 8000 → 57194 Len=1012
 74 14.687654  127.0.0.1  127.0.0.1  UDP   140 57194 → 8000 Len=108
 75 14.687839  127.0.0.1  127.0.0.1  UDP  1015 8000 → 57194 Len=983
128 15.695326  127.0.0.1  127.0.0.1  UDP  1015 8000 → 57194 Len=983
129 15.695822  127.0.0.1  127.0.0.1  UDP   140 57194 → 8000 Len=108
130 15.696005  127.0.0.1  127.0.0.1  UDP   992 8000 → 57194 Len=960
195 16.695720  127.0.0.1  127.0.0.1  UDP   992 8000 → 57194 Len=960
```

Server Code:

```
import pickle
import socket
import os
import sys


class HTTP_response:
    accept = False
    status = 0
    message = ""

    def __init__(self, port):
        con = Connection()
        con.connect(port)
        self.accept = True
        con.send_pkt(to_bytes(self))
        con.recv_pkt()
        req = from_bytes(con.received_pkts.pop())
        if req.method == "POST":
            con.received_pkts.append(req.message)
            con.recv_pkts()
            if os.path.isdir(req.destination):
                self.status = 200
                con.send_pkt(to_bytes(self))
                con.close()
            else:
                self.status = 404
                con.send_pkt(to_bytes(self))
                con.close()
                con.recv_pkt()
```

```
                with open(req.destination + "/" + req.filename, 'a') as f:
                    for line in con.received_pkts:
                        f.write(line + "\n")
            else:
                self.status = 404
                con.send_pkt(to_bytes(self))
        else:
            if os.path.exists(req.destination + "/" + req.filename):
                self.status = 200
                lines = []
                with open(req.destination + "/" + req.filename, 'r') as f:
                    for line in f:
                        lines.append(line)
                i = 0
                while i < len(lines) and sys.getsizeof(self.message +
lines[i]) < 750:
                    self.message += lines[i]
                    i += 1
                con.send_pkt(to_bytes(self))
                lines = lines[i:]
                result = combine_strings(lines)
                con.send_file(result)
                con.close()
                con.recv_pkt()
```

```
class HTTP_request:

    def __init__(self, address, method, directory, filename, message, destination):
        self.address = address
        self.method = method
        self.directory = directory
        self.filename = filename
        self.message = message
        self.destination = destination


class Connection:
    received_pkts = []
    address = 0
    send = True
    receive = True

    def __init__(self):
        self.received_pkts = []
        self.seq_num = 0
        self.ack_num = 0
        self.sock = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
        #self.sock.settimeout(1)  # set timeout to 1 second

    def connect(self, port):
        server_address = ('localhost', port)
```

```
        self.sock.bind(server_address)
        while True:
            syn_packet, address = self.sock.recvfrom(1024)
            syn_packet = from_bytes(syn_packet)
            self.address = address
            if not syn_packet.is_corrupt():
                if syn_packet.flags & 0xC0 == 0b10000000:
                    self.sock.settimeout(1)
                    print(syn_packet.data)
                synack_packet = Packet(0, 0, 'SYNACK Pckt',
0b11000000)
                passes = 0
                while passes < 3:
                    try:
                        self.sock.sendto(to_bytes(synack_packet),
self.address)
                        ack_packet, address = self.sock.recvfrom(1024)
                        ack_packet = from_bytes(ack_packet)
                        if not ack_packet.is_corrupt():
                            if ack_packet.flags & 0xC0 == 0b01000000:
                                print(ack_packet.data)
                                break
                            else:
                                raise Exception("Wrong connection")
                        else:
                            raise Exception("Wrong connection")
                    except socket.timeout:
                        passes += 1
                        pass
                break
            else:
                raise Exception("Wrong connection")
        else:
            raise Exception("Wrong connection")

    def send_pkt(self, data):
        if self.send:
            self.sock.settimeout(1)
            packet = Packet(self.seq_num, self.ack_num, data, 0)
            passes = 0
            while True:
                try:
                    self.sock.sendto(to_bytes(packet), self.address)
                    ack_packet, address = self.sock.recvfrom(1024)
                    ack_packet = from_bytes(ack_packet)
                    print("seq", self.seq_num)
                    if not ack_packet.is_corrupt():
                        if ack_packet.ack_num == (self.seq_num + 1) % 2:
                            self.seq_num += 1
                            self.seq_num %= 2
                            # Acknowledgment received, move on to the next
packet
                            break
                        else:
                            pass
                    else:
                        raise Exception("Corrupt Packet")
                except socket.timeout:
                    if passes < 3:
                        passes += 1
                        pass
                    else:
                        raise Exception("No ACK received")

    def recv_pkts(self):
        if self.receive:
            self.sock.settimeout(3)
            passes = 0
            while True:
                try:
                    packet, address = self.sock.recvfrom(1024)
                    packet = from_bytes(packet)
                    if not packet.is_corrupt():
                        if packet.seq_num == self.ack_num and packet.flags
& 0xE0 == 0b00100000:
                            finack_pkt = Packet(self.seq_num, self.ack_num, '',
0b01100000)
                            self.sock.sendto(to_bytes(finack_pkt), self.address)
                            self.receive = False
                            self.handle_close()
                            break
                        elif packet.seq_num == self.ack_num:
                            self.ack_num += 1
                            self.ack_num = self.ack_num % 2
                            print(self.ack_num)
                            ack_packet = Packet(0, self.ack_num, '',
0b01000000)
                            self.sock.sendto(to_bytes(ack_packet),
self.address)
                            self.received_pkts.append(packet.data)
                            pass
                        else:
                            ack_packet = Packet(0, (packet.seq_num + 1) % 2,
'', 0b01000000)
                            self.sock.sendto(to_bytes(ack_packet),
self.address)
                except socket.timeout:
                    if passes < 3:
                        passes += 1
                    else:
                        raise Exception("Timeout")

    def recv_pkt(self):
        if self.receive:
            self.sock.settimeout(3)
            passes = 0
            while True:
                try:
                    packet, address = self.sock.recvfrom(1024)
                    packet = from_bytes(packet)
                    if not packet.is_corrupt():
                        if packet.seq_num == self.ack_num and packet.flags
& 0xE0 == 0b00100000:
                            finack_pkt = Packet(self.seq_num, self.ack_num, '',
0b01100000)
                            self.sock.sendto(to_bytes(finack_pkt), self.address)
                            self.receive = False
                            self.handle_close()
                            break
                        elif packet.seq_num == self.ack_num:
                            self.ack_num += 1
                            self.ack_num = self.ack_num % 2
                            print(self.ack_num)
                            ack_packet = Packet(0, self.ack_num, '',
0b01000000)
                            self.sock.sendto(to_bytes(ack_packet),
self.address)
                            self.received_pkts.append(packet.data)
                            break
                        else:
```

```
                    ack_packet = Packet(0, (packet.seq_num + 1) % 2,
'', 0b01000000)
                    self.sock.sendto(to_bytes(ack_packet),
self.address)
            except socket.timeout:
                if passes < 3:
                    passes += 1
                else:
                    raise Exception("Timeout")

    def send_file(self, lines):
        if self.send:
            for line in lines:
                self.send_pkt(line)

    def close(self):
        self.sock.settimeout(3)
        # send FIN packet
        fin_pkt = Packet(self.seq_num, self.ack_num, '', 0b00100000)
        passes = 0
        while True:
            try:
                self.sock.sendto(to_bytes(fin_pkt), self.address)
                finack_packet, address = self.sock.recvfrom(1024)
                finack_packet = from_bytes(finack_packet)
                if not finack_packet.is_corrupt():
                    if finack_packet.flags & 0xE0 == 0b01100000:
                        # FIN-ACK packet received
                        self.send = False
                        self.handle_close()
                        self.recv_pkts()
                        break
                    else:
                        raise Exception("Wrong connection")
                else:
                    raise Exception("Corrupt Packet")
            except socket.timeout:
                if passes < 3:
                    passes += 1
                    pass
                else:
                    raise Exception("No ACK received")

    def handle_close(self):
        if not self.send and not self.receive:
            try:
                self.sock.close()
            except Exception as e:
                pass
            print("Connection closed")

    def pack_crrpt(self, packet):
        packet.checksum += 3
        packet.checksum = (packet.checksum & 0xffff) +
(packet.checksum >> 16)

    def lose_one_ack(self, data):
        if self.send:
            self.sock.settimeout(1)
            packet = Packet(self.seq_num, self.ack_num, data, 0)
            passes = 0
            while True:
                try:
                    self.sock.sendto(to_bytes(packet), self.address)
                    if passes == 0:
                        _, _ = self.sock.recvfrom(1024)
                    ack_packet, address = self.sock.recvfrom(1024)
                    ack_packet = from_bytes(ack_packet)
                    print("seq", self.seq_num)
                    if not ack_packet.is_corrupt():
                        if ack_packet.ack_num == (self.seq_num + 1) % 2:
                            self.seq_num += 1
                            self.seq_num %= 2
                            # Acknowledgment received, move on to the next
packet
                            break
                        else:
                            pass
                    else:
                        raise Exception("Corrupt Packet")
                except socket.timeout:
                    if passes < 3:
                        passes += 1
                        pass
                    else:
                        raise Exception("No ACK received")

    def lose_one_pack(self):
        if self.receive:
            self.sock.settimeout(3)
            passes = 0
            while True:
                try:
                    packet, address = self.sock.recvfrom(1024)
                    packet = from_bytes(packet)
                    if not packet.is_corrupt():
                        if packet.seq_num == self.ack_num and packet.flags
& 0xE0 == 0b00100000:
                            finack_pkt = Packet(self.seq_num, self.ack_num, '',
0b01100000)
                            self.sock.sendto(to_bytes(finack_pkt), self.address)
                            self.receive = False
                            self.handle_close()
                            break
                        elif packet.seq_num == self.ack_num:
                            self.ack_num += 1
                            self.ack_num = self.ack_num % 2
                            print(self.ack_num)
                            ack_packet = Packet(0, self.ack_num, '',
0b01000000)
                            self.sock.sendto(to_bytes(ack_packet),
self.address)
                            self.received_pkts.append(packet.data)
                            break
                        else:
                            ack_packet = Packet(0, (packet.seq_num + 1) % 2,
'', 0b01000000)
                            self.sock.sendto(to_bytes(ack_packet),
self.address)
                except socket.timeout:
                    if passes < 3:
                        passes += 1
        else:
            raise Exception("Timeout")


class Packet:
    def __init__(self, seq_num, ack_num, data, flags):
        self.seq_num = seq_num
        self.ack_num = ack_num
        self.data = data
        self.flags = flags
        self.len = len(data)
        self.checksum = self.calculate_checksum(data)

    def calculate_checksum(self, data):
        if isinstance(data, str):
            data = data.encode()
        if len(data) % 2 == 1:
            data += b'\x00'  # append null byte to make even length
        checksum = 0
        for i in range(0, len(data), 2):
            chunk = (data[i] << 8) + data[i + 1]
            checksum += chunk
            checksum = (checksum & 0xffff) + (checksum >> 16)
        return ~checksum & 0xffff

    def is_corrupt(self):
        data = self.data
        if isinstance(data, str):
            data = data.encode()
        if len(data) % 2 == 1:
            data += b'\x00'  # append null byte to make even length
        checksum = 0
        for i in range(0, len(data), 2):
            chunk = (data[i] << 8) + data[i + 1]
            checksum += chunk
            checksum = (checksum & 0xffff) + (checksum >> 16)
        checksum = ~checksum & 0xffff
        return not checksum == self.checksum


def to_bytes(obj):
    return pickle.dumps(obj)


def from_bytes(bytes_packet):
    return pickle.loads(bytes_packet)


def combine_strings(strings):
    result = []
    current = ""
    for s in strings:
        if len(current.encode()) + len(s.encode()) > 900:
            result.append(current)
            current = ""
        current += s
    if current:
        result.append(current)
    return result


serve = HTTP_response(8000)
```

Client Code:

```
import pickle
import socket
import sys


class HTTP_response:
   accept = False
   status = 0
   message = ""


class HTTP_request:

   def __init__(self, address, method, directory, filename,
message, destination):
      self.address = address
      self.method = method
      self.directory = directory
      self.filename = filename
      self.message = message
      self.destination = destination
      self.request()

   def request(self):
      con = Connection(self.address)
      con.recv_pkt()
      res = from_bytes(con.received_pkts.pop())
      if res.accept:
         if self.method == "POST":
            lines = []
            with open(self.directory + "/" + self.filename, 'r') as f:
               for line in f:
                  lines.append(line)
            i = 0
            while i < len(lines) and sys.getsizeof(self.message +
lines[i]) < 750:
               self.message += lines[i]
               i += 1
            con.send_pkt(to_bytes(self))
            lines = lines[i:]
            result = combine_strings(lines)
            con.send_file(result)
            con.close()
            con.recv_pkt()
            stat = from_bytes(con.received_pkts.pop())
            if stat.status == 200:
               print("Status 200 OK")
            else:
               print("Status 404 NOT FOUND")
            con.recv_pkt()
         else:
            con.send_pkt(to_bytes(self))
            con.recv_pkt()
            stat = from_bytes(con.received_pkts.pop())
            if stat.status == 200:
               print("Status 200 OK")
               con.recv_pkts()
               with open(self.directory + "/" + self.filename, 'a') as f:
                  for line in con.received_pkts:
                     f.write(line+"\n")

            else:
               print("Status 404 NOT FOUND")
            con.close()
      else:
         print("Connection Not Accepted")


class Connection:
   received_pkts = []
   send = True
   receive = True

   def __init__(self, address):
      self.received_pkts = []
      self.address = address
      self.seq_num = 0
      self.ack_num = 0
      self.sock = socket.socket(socket.AF_INET,
socket.SOCK_DGRAM)
      self.sock.settimeout(1)  # set timeout to 1 second
      self._connect()

   def _connect(self):
      # send SYN packet
      syn_pkt = Packet(self.seq_num, self.ack_num, 'Syn Pckt',
0b10000000)
      self.sock.sendto(to_bytes(syn_pkt), self.address)
      while True:
         synack_packet, address = self.sock.recvfrom(1024)
         synack_packet = from_bytes(synack_packet)
         if not synack_packet.is_corrupt():
            if synack_packet.flags & 0xC0 == 0b11000000:
               print(synack_packet.data)
               ack_packet = Packet(0, 0, 'Ack Pckt', 0b01000000)
               self.sock.sendto(to_bytes(ack_packet), self.address)
               break
            else:
               raise Exception("Wrong connection")
         else:
            raise Exception("Wrong connection")

   def send_pkt(self, data):
      if self.send:
         self.sock.settimeout(1)
         packet = Packet(self.seq_num, self.ack_num, data, 0)
         passes = 0
         while True:
            try:
               self.sock.sendto(to_bytes(packet), self.address)
               ack_packet, address = self.sock.recvfrom(1024)
               ack_packet = from_bytes(ack_packet)
               print("seq", self.seq_num)
               if not ack_packet.is_corrupt():
                  if ack_packet.ack_num == (self.seq_num + 1) % 2:
                     self.seq_num += 1
                     self.seq_num %= 2
                     # Acknowledgment received, move on to the next
packet
```

```
                    break
                else:
                    pass
            else:
                raise Exception("Corrupt Packet")
        except socket.timeout:
            if passes < 3:
                passes += 1
                pass
            else:
                raise Exception("No ACK received")

    def recv_pkts(self):
        if self.receive:
            self.sock.settimeout(3)
            while True:
                packet, address = self.sock.recvfrom(1024)
                packet = from_bytes(packet)
                if not packet.is_corrupt():
                    if packet.seq_num == self.ack_num and packet.flags &
0xE0 == 0b00100000:
                        finack_pkt = Packet(self.seq_num, self.ack_num, '',
0b01100000)
                        self.sock.sendto(to_bytes(finack_pkt), self.address)
                        self.receive = False
                        self.handle_close()
                        break
                    elif packet.seq_num == self.ack_num:
                        self.ack_num += 1
                        self.ack_num = self.ack_num % 2
                        print(self.ack_num)
                        ack_packet = Packet(0, self.ack_num, '', 0b01000000)
                        self.sock.sendto(to_bytes(ack_packet), self.address)
                        self.received_pkts.append(packet.data)
                        pass
                    else:
                        ack_packet = Packet(0, (packet.seq_num + 1) % 2, '',
0b01000000)
                        self.sock.sendto(to_bytes(ack_packet), self.address)


    def recv_pkt(self):
        if self.receive:
            self.sock.settimeout(3)
            while True:
                packet, address = self.sock.recvfrom(1024)
                packet = from_bytes(packet)
                if not packet.is_corrupt():
                    if packet.seq_num == self.ack_num and packet.flags &
0xE0 == 0b00100000:
                        finack_pkt = Packet(self.seq_num, self.ack_num, '',
0b01100000)
                        self.sock.sendto(to_bytes(finack_pkt), self.address)
                        self.receive = False
                        self.handle_close()
                        break
                    elif packet.seq_num == self.ack_num:
                        self.ack_num += 1
                        self.ack_num = self.ack_num % 2
                        print(self.ack_num)
                        ack_packet = Packet(0, self.ack_num, '', 0b01000000)
                        self.sock.sendto(to_bytes(ack_packet), self.address)
                        self.received_pkts.append(packet.data)
                        break
                    else:
                        ack_packet = Packet(0, (packet.seq_num + 1) % 2, '',
0b01000000)
                        self.sock.sendto(to_bytes(ack_packet), self.address)

    def send_file(self, lines):
        if self.send:
            for line in lines:
                self.send_pkt(line)

    def close(self):
        self.sock.settimeout(3)
        # send FIN packet
        fin_pkt = Packet(self.seq_num, self.ack_num, '', 0b00100000)
        passes = 0
        while True:
            try:
                self.sock.sendto(to_bytes(fin_pkt), self.address)
                finack_packet, address = self.sock.recvfrom(1024)
                finack_packet = from_bytes(finack_packet)
                if not finack_packet.is_corrupt():
                    if finack_packet.flags & 0xE0 == 0b01100000:
                        # FIN-ACK packet received
                        self.send = False
                        self.handle_close()
                        self.recv_pkts()
                        break
                    else:
                        raise Exception("Wrong connection")
                else:
                    raise Exception("Corrupt Packet")
            except socket.timeout:
                if passes < 3:
                    passes += 1
                    pass
                else:
                    raise Exception("No ACK received")

    def handle_close(self):
        if not self.send and not self.receive:
            try:
                self.sock.close()
            except Exception as e:
                pass
            print("Connection closed")

    def pack_crrpt(self, packet):
        packet.checksum += 3
        packet.checksum = (packet.checksum & 0xffff) +
(packet.checksum >> 16)

    def lose_one_ack(self, data):
        if self.send:
            self.sock.settimeout(1)
            packet = Packet(self.seq_num, self.ack_num, data, 0)
            passes = 0
            while True:
                try:
```

```
                self.sock.sendto(to_bytes(packet), self.address)
                if passes == 0:
                    _, _ = self.sock.recvfrom(1024)
                ack_packet, address = self.sock.recvfrom(1024)
                ack_packet = from_bytes(ack_packet)
                print("seq", self.seq_num)
                if not ack_packet.is_corrupt():
                    if ack_packet.ack_num == (self.seq_num + 1) % 2:
                        self.seq_num += 1
                        self.seq_num %= 2
                        # Acknowledgment received, move on to the next
packet
                        break
                    else:
                        pass
                else:
                    raise Exception("Corrupt Packet")
            except socket.timeout:
                if passes < 3:
                    passes += 1
                    pass
                else:
                    raise Exception("No ACK received")

    def crrpt_one_pack(self):
        if self.receive:
            self.sock.settimeout(1.5)
            passes = 0
            while True:
                packet, address = self.sock.recvfrom(1024)
                packet = from_bytes(packet)
                if passes == 0:
                    self.pack_crrpt(packet)
                    passes += 1
                if not packet.is_corrupt():
                    if packet.seq_num == self.ack_num and packet.flags &
0xE0 == 0b00100000:
                        finack_pkt = Packet(self.seq_num, self.ack_num, '',
0b01100000)
                        self.sock.sendto(to_bytes(finack_pkt), self.address)
                        self.receive = False
                        self.handle_close()
                        break
                    elif packet.seq_num == self.ack_num:
                        self.ack_num += 1
                        self.ack_num = self.ack_num % 2
                        print(self.ack_num)
                        ack_packet = Packet(0, self.ack_num, '', 0b01000000)
                        self.sock.sendto(to_bytes(ack_packet), self.address)
                        self.received_pkts.append(packet.data)
                        break
                    else:
                        ack_packet = Packet(0, (packet.seq_num + 1) % 2, '',
0b01000000)
                        self.sock.sendto(to_bytes(ack_packet), self.address)

    def lose_first_pack(self):
        if self.receive:
            self.sock.settimeout(1.5)
            while True:
                _, _ = self.sock.recvfrom(1024)
                packet, address = self.sock.recvfrom(1024)
                packet = from_bytes(packet)
                if not packet.is_corrupt():
                    if packet.seq_num == self.ack_num and packet.flags &
0xE0 == 0b00100000:
                        finack_pkt = Packet(self.seq_num, self.ack_num, '',
0b01100000)
                        self.sock.sendto(to_bytes(finack_pkt), self.address)
                        self.receive = False
                        self.handle_close()
                        break
                    elif packet.seq_num == self.ack_num:
                        self.ack_num += 1
                        self.ack_num = self.ack_num % 2
                        print(self.ack_num)
                        ack_packet = Packet(0, self.ack_num, '', 0b01000000)
                        self.sock.sendto(to_bytes(ack_packet), self.address)
                        self.received_pkts.append(packet.data)
                        pass
                    else:
                        ack_packet = Packet(0, (packet.seq_num + 1) % 2, '',
0b01000000)
                        self.sock.sendto(to_bytes(ack_packet), self.address)


class Packet:
    def __init__(self, seq_num, ack_num, data, flags):
        self.seq_num = seq_num
        self.ack_num = ack_num
        self.data = data
        self.flags = flags
        self.len = len(data)
        self.checksum = self.calculate_checksum(data)

    def calculate_checksum(self, data):
        if isinstance(data, str):
            data = data.encode()
        if len(data) % 2 == 1:
            data += b'\x00'  # append null byte to make even length
        checksum = 0
        for i in range(0, len(data), 2):
            chunk = (data[i] << 8) + data[i + 1]
            checksum += chunk
            checksum = (checksum & 0xffff) + (checksum >> 16)
        return ~checksum & 0xffff

    def is_corrupt(self):
        data = self.data
        if isinstance(data, str):
            data = data.encode()
        if len(data) % 2 == 1:
            data += b'\x00'  # append null byte to make even length
        checksum = 0
        for i in range(0, len(data), 2):
            chunk = (data[i] << 8) + data[i + 1]
            checksum += chunk
            checksum = (checksum & 0xffff) + (checksum >> 16)
        checksum = ~checksum & 0xffff
        return not checksum == self.checksum
```

```
def to_bytes(obj):
    return pickle.dumps(obj)


def from_bytes(bytes_packet):
    return pickle.loads(bytes_packet)


def combine_strings(strings):
    result = []
    current = ""
    for s in strings:
        if len(current.encode()) + len(s.encode()) > 900:
            result.append(current)
            current = ""
        current += s
    if current:
        result.append(current)
    return result


HTTP_request(('localhost', 8000), "GET", "src", "alice.txt", "",
"dest")
```