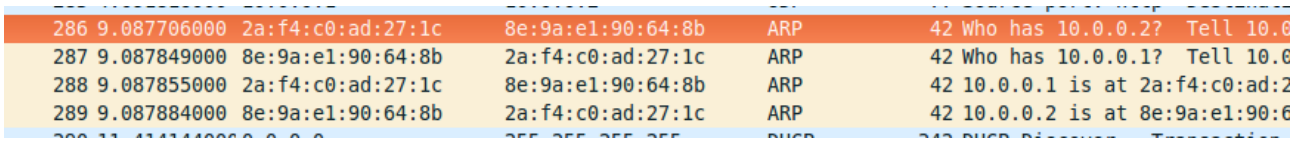
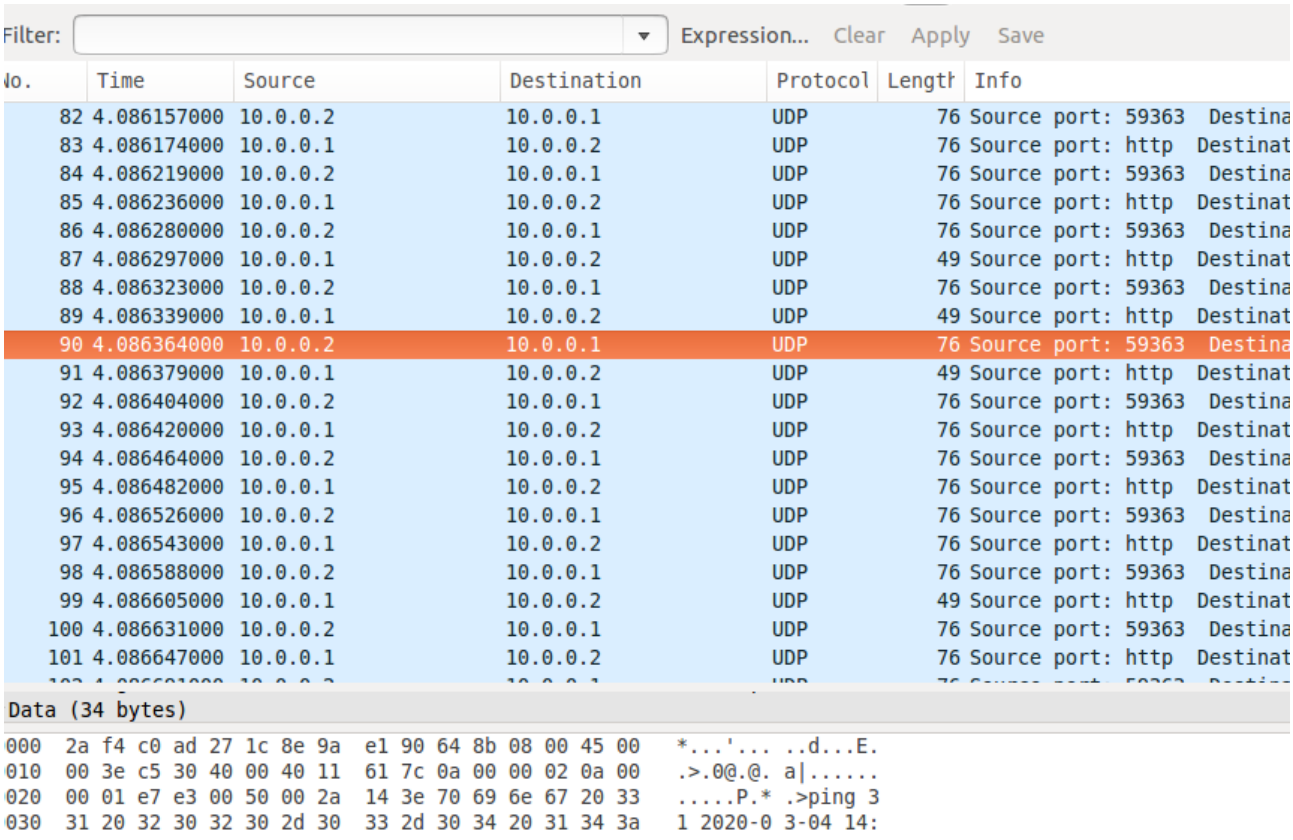
Eric Wang/Xuhui Wang CSC361 P2 documentation

ARP packets:



286	9.087706000	2a:f4:c0:ad:27:1c	8e:9a:e1:90:64:8b	ARP	42 Who has 10.0.0.2? Tell 10.0
287	9.087849000	8e:9a:e1:90:64:8b	2a:f4:c0:ad:27:1c	ARP	42 Who has 10.0.0.1? Tell 10.0
288	9.087855000	2a:f4:c0:ad:27:1c	8e:9a:e1:90:64:8b	ARP	42 10.0.0.1 is at 2a:f4:c0:ad:2
289	9.087884000	8e:9a:e1:90:64:8b	2a:f4:c0:ad:27:1c	ARP	42 10.0.0.2 is at 8e:9a:e1:90:6

UDP packets:



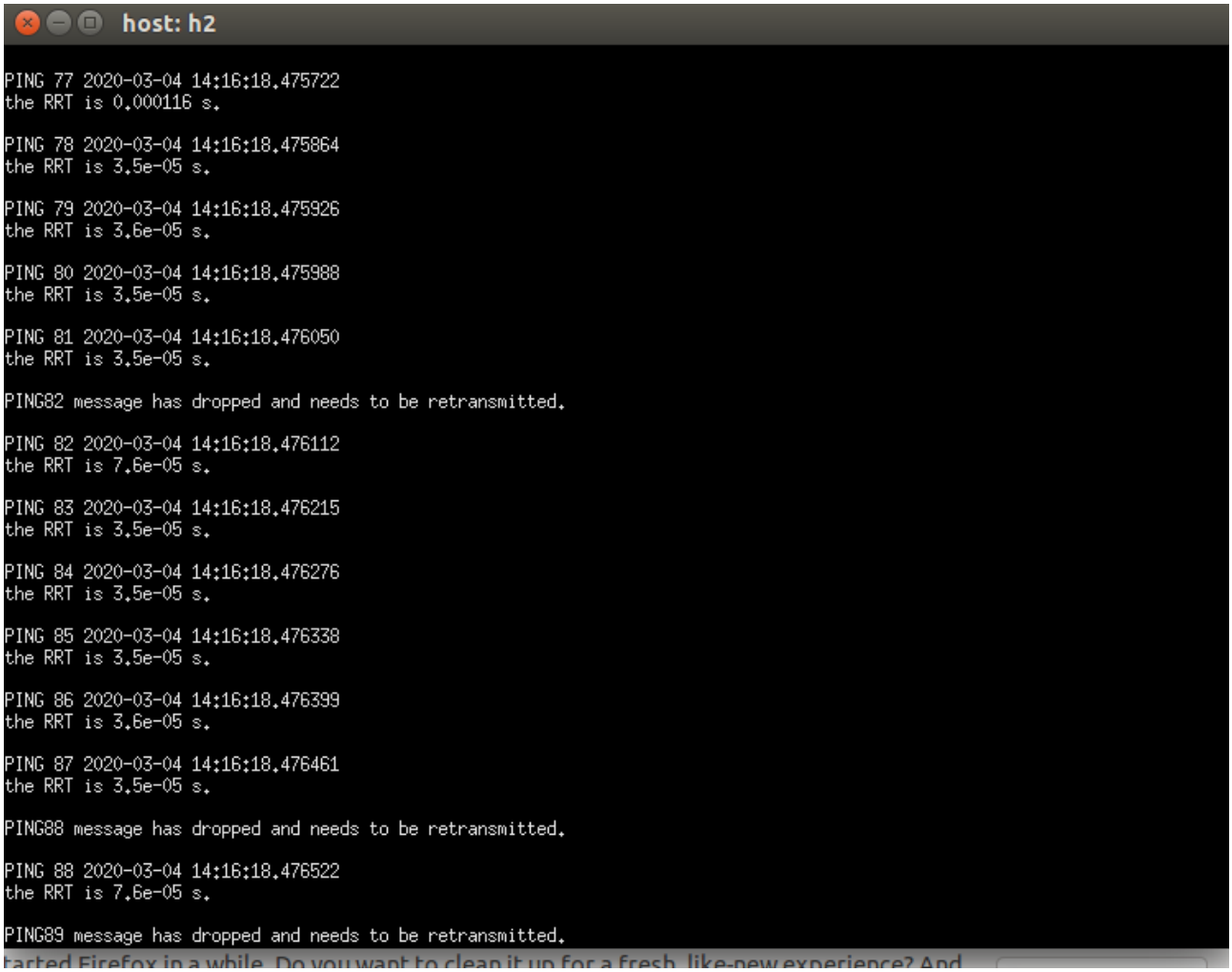
Filter: Expression... Clear Apply Save

No.	Time	Source	Destination	Protocol	Length	Info
82	4.086157000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
83	4.086174000	10.0.0.1	10.0.0.2	UDP	76	Source port: http Destinat
84	4.086219000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
85	4.086236000	10.0.0.1	10.0.0.2	UDP	76	Source port: http Destinat
86	4.086280000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
87	4.086297000	10.0.0.1	10.0.0.2	UDP	49	Source port: http Destinat
88	4.086323000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
89	4.086339000	10.0.0.1	10.0.0.2	UDP	49	Source port: http Destinat
90	4.086364000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
91	4.086379000	10.0.0.1	10.0.0.2	UDP	49	Source port: http Destinat
92	4.086404000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
93	4.086420000	10.0.0.1	10.0.0.2	UDP	76	Source port: http Destinat
94	4.086464000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
95	4.086482000	10.0.0.1	10.0.0.2	UDP	76	Source port: http Destinat
96	4.086526000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
97	4.086543000	10.0.0.1	10.0.0.2	UDP	76	Source port: http Destinat
98	4.086588000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
99	4.086605000	10.0.0.1	10.0.0.2	UDP	49	Source port: http Destinat
100	4.086631000	10.0.0.2	10.0.0.1	UDP	76	Source port: 59363 Destina
101	4.086647000	10.0.0.1	10.0.0.2	UDP	76	Source port: http Destinat

Data (34 bytes)

```
000  2a f4 c0 ad 27 1c 8e 9a  e1 90 64 8b 08 00 45 00   *...'... ..d...E.
010  00 3e c5 30 40 00 40 11  61 7c 0a 00 00 02 0a 00   .>.0@.@. a|......
020  00 01 e7 e3 00 50 00 2a  14 3e 70 69 6e 67 20 33   .....P.* .>ping 3
030  31 20 32 30 32 30 2d 30  33 2d 30 34 20 31 34 3a   1 2020-0 3-04 14:
```

Client terminal:



Retransmission:

```
# Assign IP address and port number to socket

serverSocket.bind((sys.argv[1], int(sys.argv[2])))
print("The server is ready to receive!")

while True:
        # Receive the client packet along with the address it is coming from
        msg, clientAddress = serverSocket.recvfrom(2048)

        rand = random.randint(1, 10)

        if rand < 4:
                reMsg = "dropped"
                serverSocket.sendto(reMsg.encode(), clientAddress)
                continue

        modifiedMessage = msg.decode().upper()
        serverSocket.sendto(modifiedMessage.encode(), clientAddress)
```

I implemented retransmission on the server side as indicated in the code above:

If the server decides to drop the packet, it sends a string ”dropped” to indicate the client that the packet last sent by the client was dropped.

```
dropped = False
msg = "ping " + str(i)  + " " +  datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f"
start = datetime.datetime.now()
clientSocket.sendto(msg.encode(), (serverName, serverPort))
modifiedMsg, serverAddress = clientSocket.recvfrom(2048)

while modifiedMsg.decode() == "dropped":
        dropped = True
        print("PING" + str(i) + " message has dropped and needs to be retransmitted.")
        print()
        clientSocket.sendto(msg.encode(), (serverName, serverPort))
        modifiedMsg, serverAddress = clientSocket.recvfrom(2048)

end = datetime.datetime.now()
print(modifiedMsg.decode())
print("the RRT is " + str((end - start).total_seconds()) + " s.")
print()
#time.sleep(0.3)
```

I implemented retransmission on the client side as indicated in the code above:

If the client receives the packet replied by the server containing the string “dropped”, it resends the same packet to the server until it gets the right replied packet.

RRT:

```
msg = "ping " + str(i)  + " " +  datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")
start = datetime.datetime.now()
clientSocket.sendto(msg.encode(), (serverName, serverPort))
modifiedMsg, serverAddress = clientSocket.recvfrom(2048)

while modifiedMsg.decode() == "dropped":
        dropped = True
        print("PING" + str(i) + " message has dropped and needs to be retransmitted.")
        print()
        clientSocket.sendto(msg.encode(), (serverName, serverPort))
        modifiedMsg, serverAddress = clientSocket.recvfrom(2048)

end = datetime.datetime.now()
print(modifiedMsg.decode())
print("the RRT is " + str((end - start).total_seconds()) + " s.")
print()
```

I implemented the RRT calculation as indicated in the code above:

Before I send the packet to the server, I get the start timestamp.

I get the end timestamp after I have received the replied packet correctly (including the retransmission time if need).

Then I get the time difference between start and end, which is the RRT.