



南开大学
Nankai University

《计算机网络》实验报告

(2022~2023 学年第一学期)

实验名称：基于套接字的网络程序设计

学 院：软件学院

姓 名：李鹏

学 号：2113850

指导老师：张圣林

2023 年 12 月 11 日

目录

1 实验 4.1 套接字基础与 UDP 通信	1
1.1 实验目的	1
1.2 实验内容	1
1.3 实验原理、方法和手段	1
1.4 实验步骤	3
1.5 实验结果	3
2 实验 4.2 TCP 通信与 Web 服务器	4
2.1 实验目的	4
2.2 实验内容	4
2.3 实验原理、方法和手段	4
2.4 实验步骤	6
2.5 实验结果	7
3 实验 4.3 SMTP 客户端实现	9
3.1 实验目的	9
3.2 实验内容	9
3.3 实验原理、方法和手段	9
3.4 实验步骤	10
3.5 实验结果	10
4 心得体会	12
5 附录	12
5.1 实验截图	12
5.2 源代码	15

实验四：基于套接字的网络程序设计

1 实验 4.1 套接字基础与 UDP 通信

1.1 实验目的

熟悉基于 Python 进行 UDP 套接字编程的基础知识，掌握使用 UDP 套接字发送和接收数据包，以 设置正确的套接字超时，了解 Ping 应用程序的基本概念，并理解其在简单判断网络状态，例如计算数据包丢失率等统计数据方面的意义。

1.2 实验内容

实验内容包括以下几个方面：

- 编写基于 UDP 的 Ping 客户端程序：

使用 Python 实现一个非标准 Ping 客户端，与标准 Ping 命令不同，采用 UDP 进行通信。实现要求：客户端发送 ping 报文至服务器，并接收对应的 pong 报文。计算从发送 ping 报文到接收 pong 报文为止的往返时延（Round-Trip Time, RTT）。

- 执行过程中的细节：

在一次执行中，客户端通过 UDP 向服务器发送 10 个 ping 报文。对于每个报文，当收到对应的 ping 报文时，确认并打印输出对应的 RTT 值。在整个执行过程中，客户端程序需考虑分组丢失情况，最长等待 1 秒，超时则打印丢失报文。

1.3 实验原理、方法和手段

UDP 作为一种传输层协议，提供无连接通信，且不对传送的数据包进行可靠性保证。因此，适合一次传输少量数据的应用场景。若需要保证可靠性，应由应用层负责。Ping 程序创建的正是一种不需要可靠性保证的程序，利用这种不可靠性测量网络的联通情况。

尽管 UDP 不保证通信的可靠性和包的顺序，也不提供流量控制，但由于 UDP 的控制选项较少，数据传输过程中延迟小、效率高。一些对可靠性要求不高但对性能敏感的应用层协

议选择基于 UDP 实现，如 TFTP、SNMP、NFS、DNS、BOOTP 等，通常使用 53 (DNS)、69 (TFTP)、161 (SNMP) 等端口。

基于 UDP 的无连接客户/服务器在 Python 实现中的工作流程如下：

1. 服务器端通过调用 socket 创建套接字启动服务器；
2. 服务器调用 bind 指定服务器的套接字地址，然后调用 recvfrom 等待接收数据；
3. 客户端调用 socket 创建套接字，然后调用 sendto 向服务器发送数据；
4. 服务器接收到客户端发来的数据后，调用 sendto 向客户端发送应答数据；
5. 客户调用 recvfrom 接收服务器发来的应答数据；
6. 数据传输结束后，服务器和客户通过调用 close 关闭套接字。

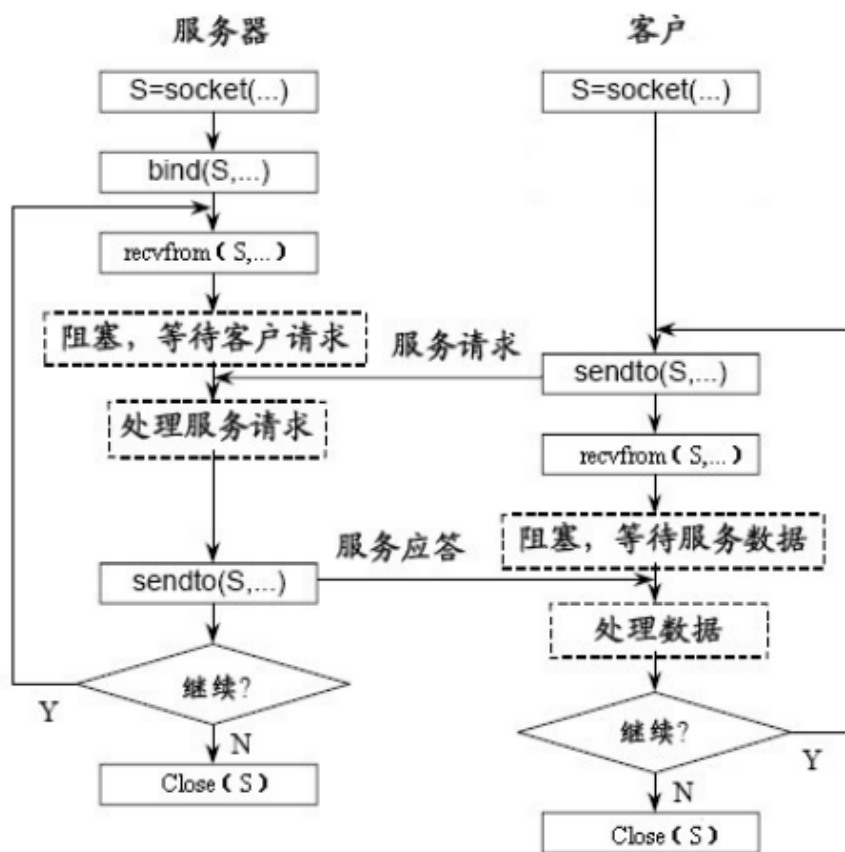


图 1: UDP 客户端/服务器

1.4 实验步骤

在实验中，按照以下要求完成 Ping 程序的客户端部分。先启动服务端，再运行客户端。

1. 使用 UDP 发送 Ping 消息：

- UDP 是无连接协议，无需建立连接。

2. 响应及 RTT 计算：

- 如果服务器在 1 秒内响应，打印该响应消息。
- 计算并打印每个数据包的往返时间 RTT（以秒为单位）。

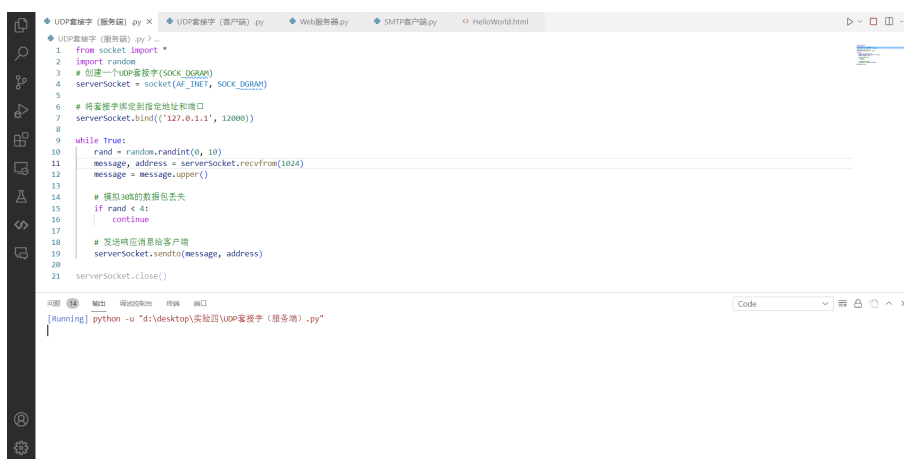
3. 请求超时处理：

- 如果服务器未在 1 秒内响应，打印“请求超时”。

完成编写后，使用客户端 Ping 程序通过 UDP 向目标服务器发送 10 个 Ping 报文。

1.5 实验结果

启动服务端：



```
UDP服务端 (服务端).py x
1 from socket import *
2 import random
3 # 创建一个UDP服务端(SOCK_DGRAM)
4 serverSocket = socket(AF_INET, SOCK_DGRAM)
5
6 # 将套接字绑定到指定地址和端口
7 serverSocket.bind(('127.0.0.1', 12000))
8
9 while True:
10     rand = random.randint(0, 10)
11     message, address = serverSocket.recvfrom(1024)
12     message = message.upper()
13
14     # 模拟30%的数据包丢失
15     if rand < 4:
16         continue
17
18     # 发送响应消息给客户端
19     serverSocket.sendto(message, address)
20
21 serverSocket.close()
```

问题 输出 调试控制台 终端 窗口

[Running] python -u "d:\desktop\实验四\UDP服务端 (服务端).py"

图 2: UDP 服务端

启动客户端：



```
1 from socket import *
2 import time
3
4 IP = '127.0.0.1'
5 PORT = 12000
6 TIMEOUT = 1 # 1秒超时
7
8 # 实例化一个socket对象, 指明UDP协议
9 dataSocket = socket(AF_INET, SOCK_DGRAM)
10
11 for i in range(10):
12     # 记录发送数据包之前的时间
13     start_time = time.time()
14
15     # 发送UDP数据, 将数据封装到数据包
16     dataSocket.sendto(f'Ping {i}'.encode(), (IP, PORT))
17
18     try:
19         # 设置接收响应的超时时间
20         dataSocket.settimeout(TIMEOUT)
21         # 接收服务器的消息
22         received, addr = dataSocket.recvfrom(1024)
```

```
PS D:\desktop\实验四> & D:/python/python.exe d:/desktop/实验四/UDP套接字(客户端).py
来自 ('127.0.0.1', 12000) 的响应: PING 0 - 往返时间: 0.000000 秒
来自 ('127.0.0.1', 12000) 的响应: PING 1 - 往返时间: 0.000000 秒
来自 ('127.0.0.1', 12000) 的响应: PING 2 - 往返时间: 0.007977 秒
来自 ('127.0.0.1', 12000) 的响应: PING 3 - 往返时间: 0.000000 秒
Ping 4 请求超时
Ping 5 请求超时
来自 ('127.0.0.1', 12000) 的响应: PING 6 - 往返时间: 0.000000 秒
来自 ('127.0.0.1', 12000) 的响应: PING 7 - 往返时间: 0.000000 秒
Ping 8 请求超时
来自 ('127.0.0.1', 12000) 的响应: PING 9 - 往返时间: 0.000000 秒
PS D:\desktop\实验四>
```

图 3: UDP 客户端

分析:

Ping 命令成功得到响应, 说明基于 UDP 的 Ping 客户端程序在局域网环境下运行正常。部分请求超时和非零的往返延时 RTT 可能是由于网络波动或其他因素引起的, 或者是计时精度不够。综合来看, 该 Ping 程序对于服务器的请求能够在大多数情况下得到迅速的响应, 但在部分情况下可能会遇到请求超时 (有时丢包率大于 30 %)。

2 实验 4.2 TCP 通信与 Web 服务器

2.1 实验目的

熟悉基于 Python 进行 TCP 套接字编程的基础知识, 理解 HTTP 报文格式, 能基于 Python 编写一个可以一次响应一个 HTTP 请求, 并返回静态文件的简单 Web 服务器。

2.2 实验内容

利用 Python 开发一个可以一次处理一个 HTTP 请求的 Web 服务器, 该服务器可以接受并解析 HTTP 请求, 然后从服务器的文件系统中读取被 HTTP 请求的文件, 并根据该文件是否存在而向客户端发送正确的响应消息

2.3 实验原理、方法和手段

在 Python 中, 基于 TCP 的面向客户端/服务器的工作流程如下:

1. 服务器端：

- 通过调用 `socket` 创建套接字来启动服务器；
- 调用 `bind` 绑定指定服务器的套接字地址（IP 地址 + 端口号）；
- 调用 `listen` 做好侦听准备，规定请求队列的长度；
- 服务器进入阻塞状态，等待客户的连接请求；
- 通过 `accept` 接收连接请求，并获得客户的 `socket` 地址。

2. 客户端：

- 通过调用 `socket` 创建套接字；
- 调用 `connect` 和服务器建立连接。

3. 连接成功后：

- 客户端和服务端之间通过 `recv` 和 `send` 来接收和发送数据。

4. 数据传输结束后：

- 服务器和客户各自通过调用 `close` 关闭套接字。

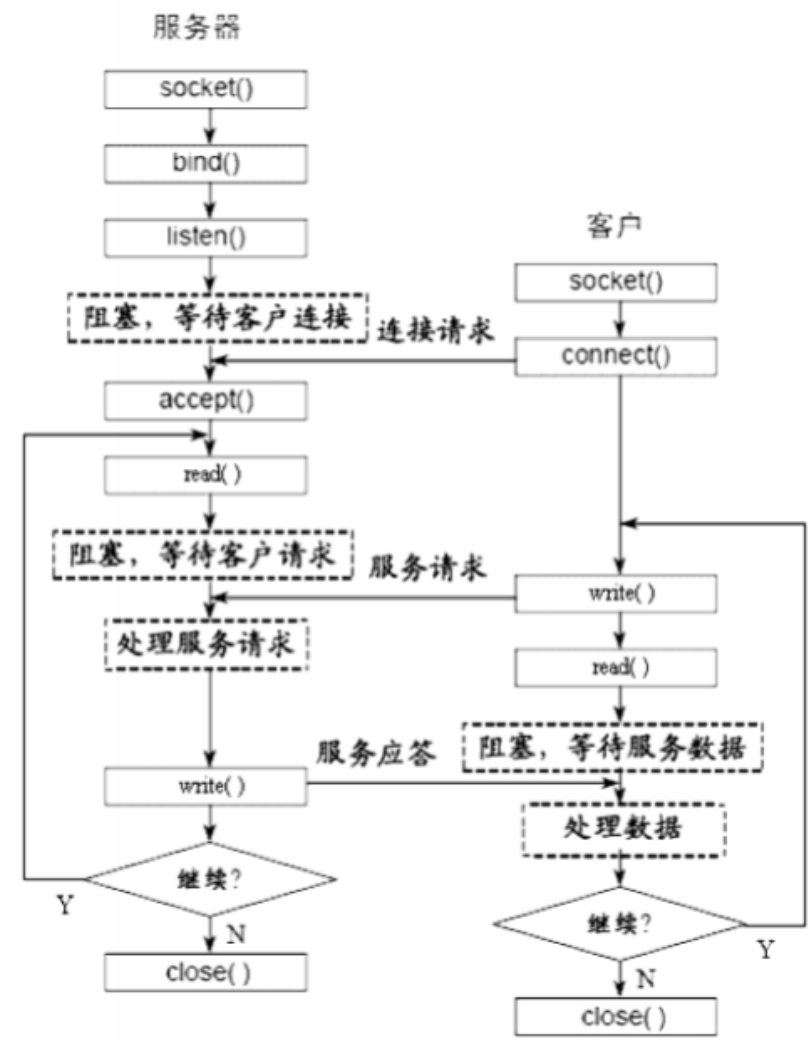


图 4: TCP 服务器/客户端

2.4 实验步骤

在实验附件一节中提供了一个不完整的 Web 服务器框架代码，学生需要逐步填充代码中不完善的部分，完成一个具有以下功能的简单 Web 服务器：

1. 创建 TCP 套接字：

- 服务器收到请求时能够创建一个 TCP 套接字。

2. 接收 HTTP 请求：

- 通过 TCP 套接字接收 HTTP 请求。

3. 解析 HTTP 请求：

- 解析 HTTP 请求并在操作系统中确定客户端所请求的特定文件。

4. 读取文件：

- 从服务器的文件系统读取客户端请求的文件。

5. 创建成功响应报文：

- 当被请求文件存在时，创建一个由被请求的文件组成的“请求成功”HTTP 响应报文。

6. 创建目标不存在响应报文：

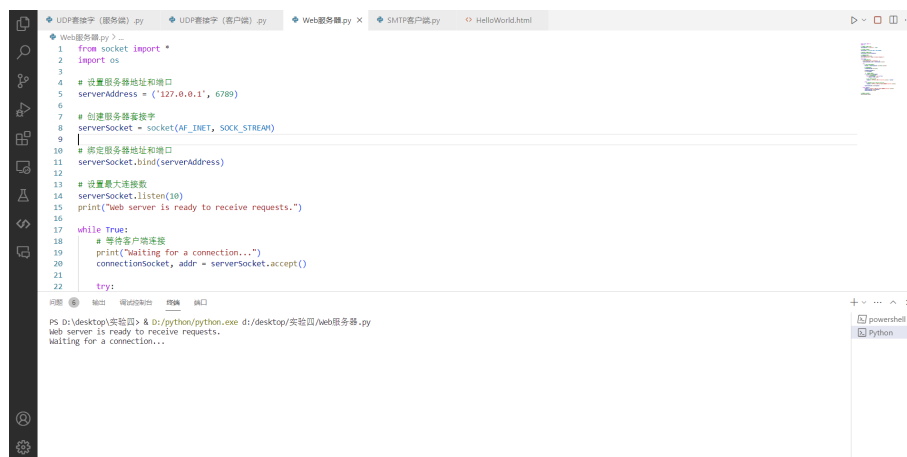
- 当被请求文件不存在时，创建“请求目标不存在”HTTP 响应报文。

7. 发送响应报文：

- 通过 TCP 连接将响应报文发回客户端。

2.5 实验结果

启动 web 服务器：

The image shows a Windows-style application window titled "Web服务器.py". Inside, there is a Python script for a simple web server. The script imports socket and os, sets a server address of ('127.0.0.1', 6789), creates a server socket, binds it, and starts listening. It then enters a loop to accept connections. The terminal output at the bottom shows the command to run the script and the message "web server is ready to receive requests. waiting for a connection...".

```
Web服务器.py > ...
1 from socket import *
2 import os
3
4 # 设置服务器地址和端口
5 serverAddress = ('127.0.0.1', 6789)
6
7 # 创建服务器套接字
8 serverSocket = socket(AF_INET, SOCK_STREAM)
9
10 # 绑定服务器地址和端口
11 serverSocket.bind(serverAddress)
12
13 # 设置最大连接数
14 serverSocket.listen(10)
15 print("web server is ready to receive requests.")
16
17 while True:
18     # 等待客户端连接
19     print("waiting for a connection...")
20     connectionSocket, addr = serverSocket.accept()
21
22     try:
```

PS D:\desktop\实验四> & D:\python\python.exe d:\desktop\实验四\Web服务器.py
web server is ready to receive requests.
waiting for a connection...

图 5: 启动 web 服务器

访问 <http://127.0.0.1:6789/HelloWorld.html>

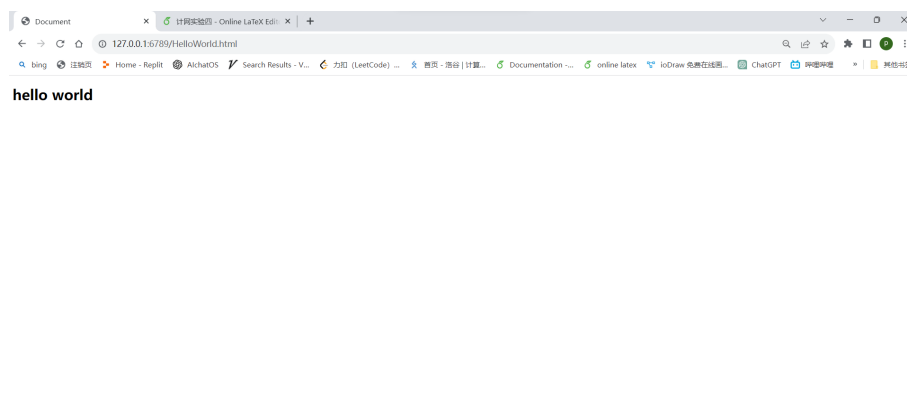


图 6: 访问 <http://127.0.0.1:6789/HelloWorld.html>

访问其他文件:



图 7: 访问其他文件

分析:

服务器运行主机的 IP 地址为 “127.0.0.1”，端口号为 6789，这是服务器监听的端口。服务器程序会自动从当前运行的路径开始查询文件。在给定的例子中，如果将 “HelloWorld.html” 文件放置在服务器程序文件存放目录中，服务器将从该路径开始查找。客户端发起请求，通过提供的 URL: “<http://127.0.0.1:6789/HelloWorld.html>”，可以成功获取到 “HelloWorld.html” 文件。若请求的 URL 为 “<http://127.0.0.1:6789/dfs.html>”，则请求的文件不存在，服务器将创建一个 “HTTP ERROR 404” 的响应报文，并将相应的信息发送给客户端。综合来看，服务器程序能够成功响应客户端的请求，根据请求的文件是否存在，分别生成相应的成功响应或目标不存在的响应。

3 实验 4.3 SMTP 客户端实现

3.1 实验目的

进一步理解和掌握基于 Python 进行 TCP 套接字编程的知识,理解 SMTP 报文格式,能基于 Python 编写一个简单的 SMTP 客户端程序。

3.2 实验内容

通过 Python 编写代码创建一个可以向标准电子邮件地址发送电子邮件的简单邮件客户端。该客户端可以与邮件服务器创建一个 TCP 连接,并基于 SMTP 协议与邮件服务器交互并发送邮件报文,完成邮件发送后关闭连接

3.3 实验原理、方法和手段

简单邮件传输协议 (Simple Mail Transfer Protocol, SMTP) 是实现电子邮件收发的主要应用层协议。以下是 SMTP 的一些关键特性和交互过程:

1. 协议基础:

- SMTP 基于 TCP 提供可靠数据传输连接。
- 工作在发送方邮件服务器上的 SMTP 客户端和接收方邮件服务器上的 SMTP 服务器共同参与邮件传输。

2. 协议历史:

- SMTP 是一种古老的应用层协议,于 1982 年首次在 RFC 文档 821 中定义,之后进行了两次更新 (RFC 2821 和 RFC 5321)。
- SMTP 拥有许多出色的特性,但也保留了一些陈旧特征,如限制邮件报文主体部分只能采用简单的 7 比特 ASCII 码表示。

3. 邮件传输过程:

- SMTP 客户端和 SMTP 服务器通过 TCP 连接进行实际通信。
- SMTP 客户端向 SMTP 服务器发送命令,如 HELLO、MAIL FROM、RCPT TO、DATA 等,每个命令用 ASCII 码表示,以及一个点 (句点) 表示消息内容结束。

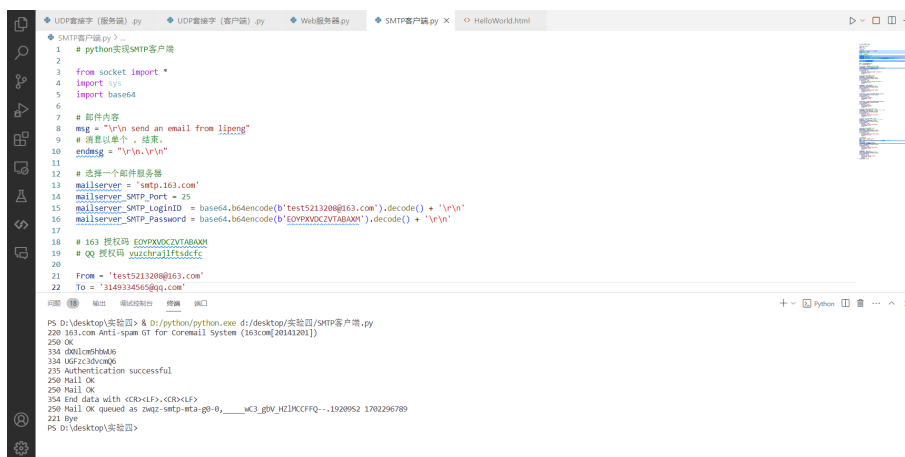
- 服务器对每个命令都回复一个回答码和相应的英文解释。
- SMTP 客户端可以通过一个连接发送多封邮件, 每封邮件用一个新的命令开始, 以点表示结束。

需要注意的是, SMTP 使用 TCP 连接, 可以在一个连接上复用多次发送邮件。在连接的最后, 通过 QUIT 命令结束与服务器的连接。

3.4 实验步骤

完成一个简单电子邮件客户端程序, 并通过向不同的账号发送电子邮件来测试程序。

3.5 实验结果



```
SMTP客户端.py > ...
1 # python实现SMTP客户端
2
3 from socket import *
4 import sys
5 import base64
6
7 # 邮件内容
8 msg = "\r\n send an email from lipeng"
9 # 消息以单个 . 结束。
10 endmsg = "\r\n.\r\n"
11
12 # 选择一个邮件服务器
13 mailserver = 'smtp.163.com'
14 mailserver SMTP_Port = 25
15 mailserver SMTP_LoginID = base64.b64encode(b'test5213208@163.com').decode() + '\r\n'
16 mailserver SMTP_Password = base64.b64encode(b'EQYPMQCVTABAMM').decode() + '\r\n'
17
18 # 163 授权码 EQYPMQCVTABAMM
19 # QQ 授权码 yuzchraji1ttdcfc
20
21 From = 'test5213208@163.com'
22 To = '3149334565@qq.com'
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
```

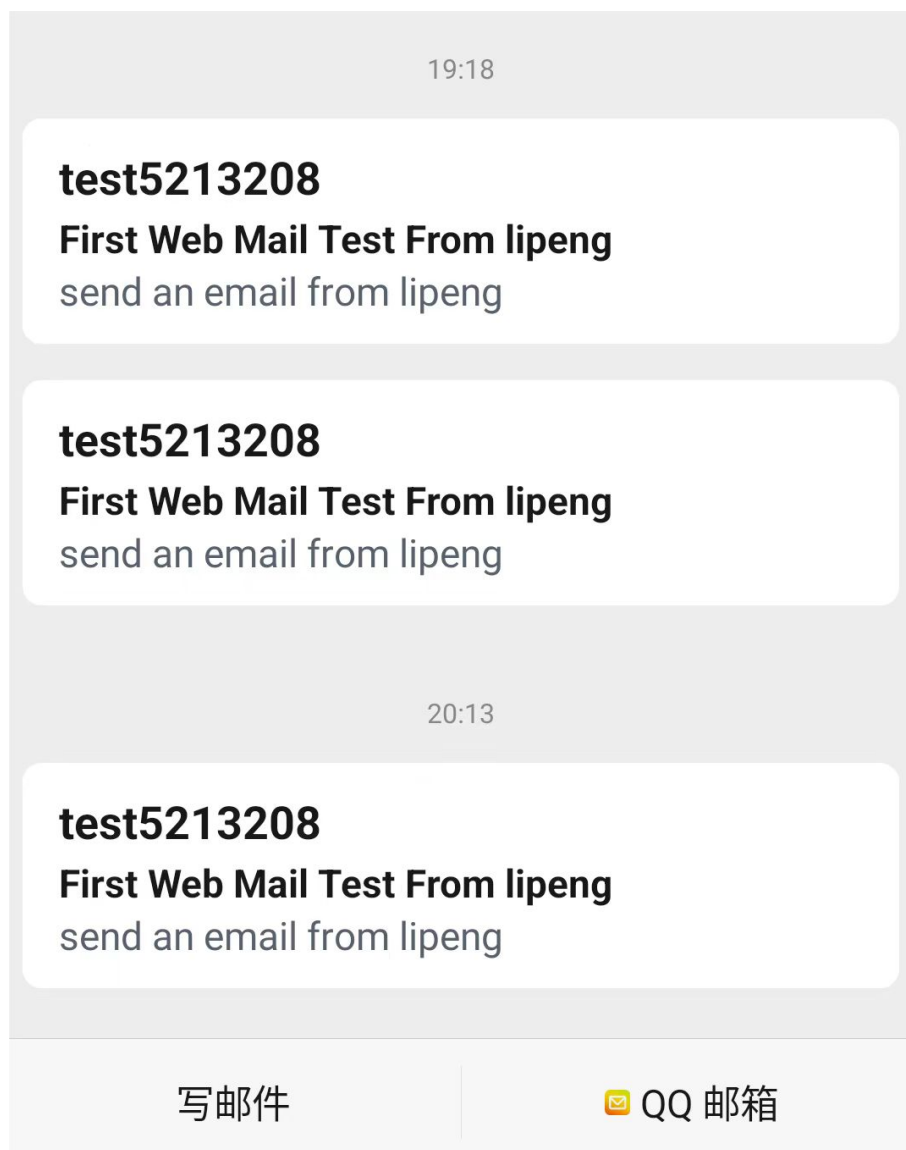


图 9: 收到邮件

分析:

1. SMTP 客户端首先与邮件服务器建立 TCP 连接，发送了 HELO 命令，成功收到服务器的响应。接着进行身份验证，使用 Base64 编码的登录 ID 和密码发送 AUTH LOGIN 命令，成功通过服务器的验证。
2. 客户端通过 MAIL FROM 命令指定发件人地址，成功收到服务器的确认。接着使用 RCPT TO 命令指定收件人地址，同样成功收到服务器的确认。之后，客户端发送 DATA 命令，开始传输邮件内容。
3. 邮件内容包括发件人、收件人、主题和正文信息。SMTP 客户端按照协议规定的格式发

送邮件内容，使用 Base64 编码保证信息的传输。在成功传输完邮件内容后，客户端收到服务器的确认。

- 最后，客户端发送 QUIT 命令结束与服务器的连接，成功收到服务器的结束确认。整个邮件发送过程完成。
- 通过 Python 实现的 SMTP 客户端程序成功地连接到邮件服务器，并完成了邮件的发送过程。

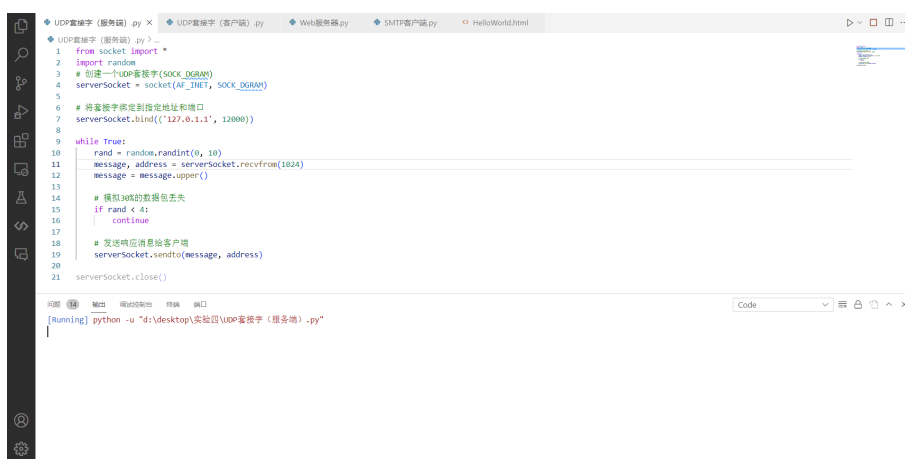
4 心得体会

通过本次实验，我学习了 UDP 和 TCP 通信的基础知识，掌握了套接字编程的实践技能。在实现 Ping 程序和搭建简单的 Web 服务器的过程中，我加深了对网络通信原理的理解，学会了解析 HTTP 请求、处理文件传输等操作。同时，通过 SMTP 客户端的实现，体会到了 SMTP 邮件传输协议的交互过程，成功地发送了电子邮件。

5 附录

5.1 实验截图

启动服务端：



```
UDP套接字 (服务端).py x
UDP套接字 (客户端).py
Web服务器.py
SMTP客户端.py
HelloWorld.html

1 from socket import *
2 import random
3 # 创建一个UDP套接字(SOCK_DGRAM)
4 serverSocket = socket(AF_INET, SOCK_DGRAM)
5
6 # 将套接字绑定到指定地址和端口
7 serverSocket.bind(('127.0.0.1', 12000))
8
9 while True:
10     rand = random.randint(0, 10)
11     message, address = serverSocket.recvfrom(1024)
12     message = message.upper()
13
14     # 模拟30%的数据包丢失
15     if rand < 4:
16         continue
17
18     # 发送响应消息给客户端
19     serverSocket.sendto(message, address)
20
21 serverSocket.close()

[Running] python -u "d:\desktop\实验四\UDP套接字 (服务端).py"
```

图 10: UDP 服务端

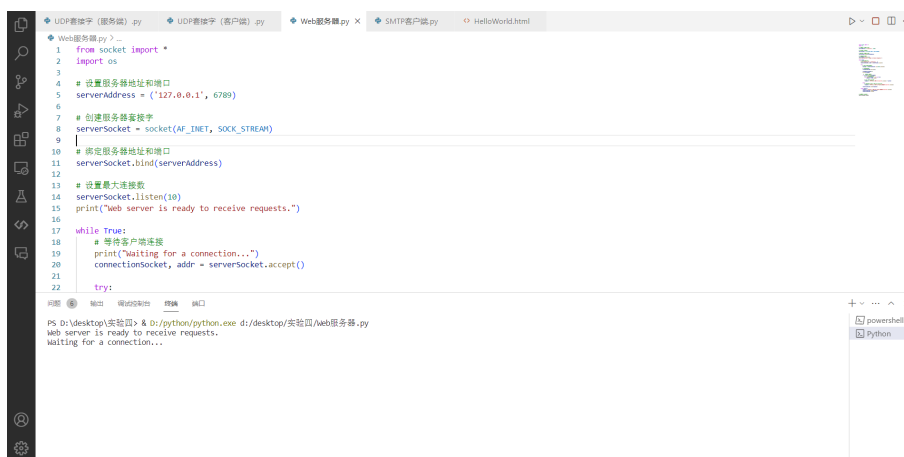
启动客户端：



```
1 from socket import *
2 import time
3
4 IP = '127.0.0.1'
5 PORT = 12000
6 TIMEOUT = 1 # 1秒超时
7
8 # 实例化一个socket对象, 指明UDP协议
9 dataSocket = socket(AF_INET, SOCK_DGRAM)
10
11 for i in range(10):
12     # 记录发送数据之前的时间
13     start_time = time.time()
14
15     # 发送udp数据, 将数据发送到服务器
16     dataSocket.sendto(f'Ping {i}'.encode(), (IP, PORT))
17
18     try:
19         # 设置接收响应的超时时间
20         dataSocket.settimeout(TIMEOUT)
21         # 接收服务器的消息
22         received, addr = dataSocket.recvfrom(1024)
```

PS D:\desktop\实验四> & D:/python/python.exe d:/desktop/实验四/UDP客户端.py
来自 ('127.0.0.1', 12000) 的响应: PING 0 - 往返时间: 0.000000 秒
来自 ('127.0.0.1', 12000) 的响应: PING 1 - 往返时间: 0.000000 秒
来自 ('127.0.0.1', 12000) 的响应: PING 2 - 往返时间: 0.007977 秒
来自 ('127.0.0.1', 12000) 的响应: PING 3 - 往返时间: 0.000000 秒
Ping 4 请求超时
来自 ('127.0.0.1', 12000) 的响应: PING 6 - 往返时间: 0.000000 秒
来自 ('127.0.0.1', 12000) 的响应: PING 7 - 往返时间: 0.000000 秒
Ping 8 请求超时
来自 ('127.0.0.1', 12000) 的响应: PING 9 - 往返时间: 0.000000 秒
PS D:\desktop\实验四>]

图 11: UDP 客户端



```
1 from socket import *
2 import os
3
4 # 设置服务器地址和端口
5 serverAddress = ('127.0.0.1', 6789)
6
7 # 创建服务器套接字
8 serverSocket = socket(AF_INET, SOCK_STREAM)
9
10 # 绑定服务器地址和端口
11 serverSocket.bind(serverAddress)
12
13 # 设置最大连接数
14 serverSocket.listen(10)
15 print("Web server is ready to receive requests.")
16
17 while True:
18     # 等待客户端连接
19     print("Waiting for a connection...")
20     connectionSocket, addr = serverSocket.accept()
21
22     try:
```

PS D:\desktop\实验四> & D:/python/python.exe d:/desktop/实验四/web服务器.py
Web server is ready to receive requests.
Waiting for a connection...

图 12: 启动 web 服务器

访问 <http://127.0.0.1:6789/HelloWorld.html>

图 13: 访问 <http://127.0.0.1:6789/HelloWorld.html>

访问其他文件:

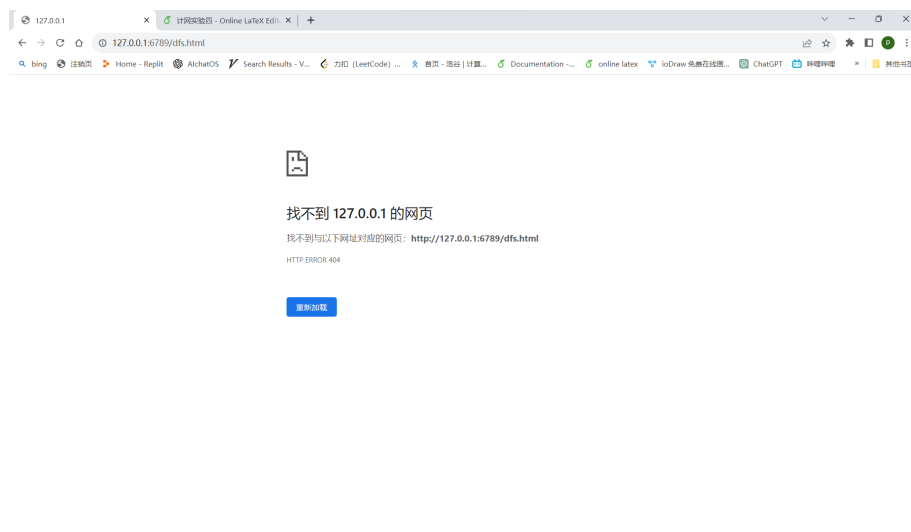


图 14: 访问其他文件

SMTP 客户端:

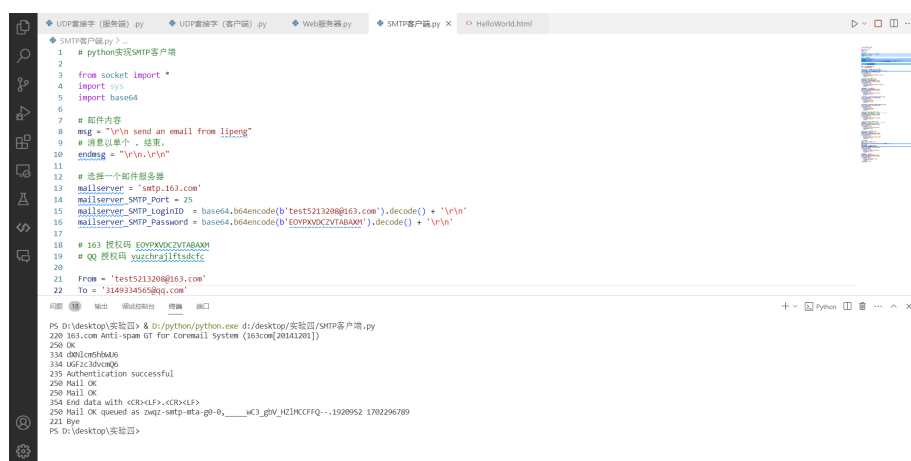


图 15: SMTP 发邮件

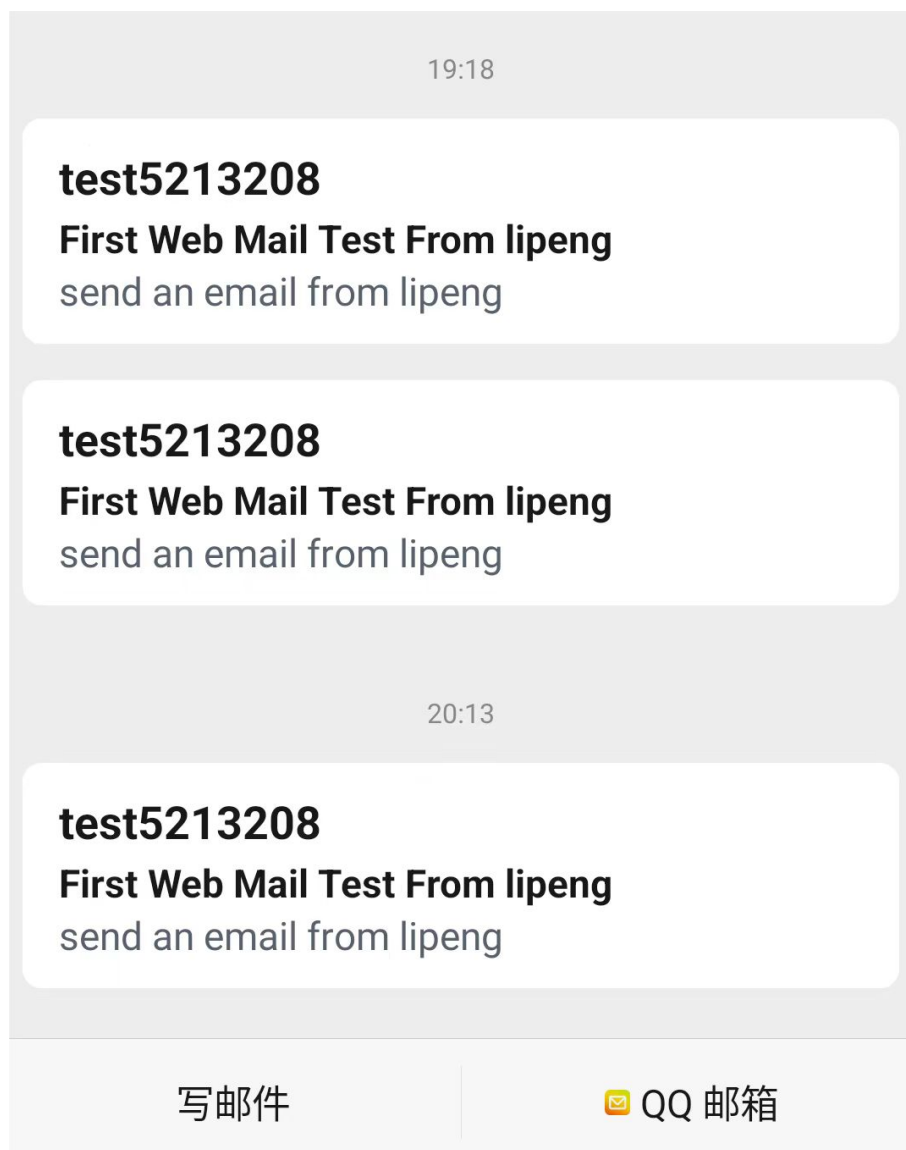


图 16: 收到邮件

5.2 源代码

UDP 套接字 (服务端)

```
1 from socket import *
2 import random
3 # 创建一个UDP套接字(SOCK_DGRAM)
4 serverSocket = socket(AF_INET, SOCK_DGRAM)
5
```

```
6 # 将套接字绑定到指定地址和端口
7 serverSocket.bind(('127.0.0.1', 12000))
8
9 while True:
10     rand = random.randint(0, 10)
11     message, address = serverSocket.recvfrom(1024)
12     message = message.upper()
13
14     # 模拟30%的数据包丢失
15     if rand < 4:
16         continue
17
18     # 发送响应消息给客户端
19     serverSocket.sendto(message, address)
20
21 serverSocket.close()
```

UDP 套接字 (客户端)

```
1 from socket import *
2 import time
3
4 IP = '127.0.0.1'
5 PORT = 12000
6 TIMEOUT = 1 # 1秒超时
7
8 # 实例化一个socket对象, 指明UDP协议
9 dataSocket = socket(AF_INET, SOCK_DGRAM)
10
11 for i in range(10):
12     # 记录发送数据包之前的时间
13     start_time = time.time()
14
15     # 发送UDP数据, 将数据发送到套接字
16     dataSocket.sendto(f'Ping {i}'.encode(), (IP, PORT))
```

```
17
18     try:
19         # 设置接收响应的超时时间
20         dataSocket.settimeout(TIMEOUT)
21         # 接收服务端的消息
22         recved, addr = dataSocket.recvfrom(1024)
23         # 如果返回空bytes, 表示对方关闭了连接
24         if not recved:
25             break
26         # 计算往返时间 (RTT)
27         rtt = time.time() - start_time
28         # 打印响应消息和RTT
29         print(f'来自 {addr} 的响应: {recved.decode()} - 往返时间: {rtt:.6f} ...
          秒')
30
31     except timeout:
32         # 处理超时
33         print(f'Ping {i} 请求超时')
34
35     dataSocket.close()
```

Web 服务器

```
1 from socket import *
2 import os
3
4 # 设置服务器地址和端口
5 serverAddress = ('127.0.0.1', 6789)
6
7 # 创建服务器套接字
8 serverSocket = socket(AF_INET, SOCK_STREAM)
9
10 # 绑定服务器地址和端口
11 serverSocket.bind(serverAddress)
12
```

```
13 # 设置最大连接数
14 serverSocket.listen(10)
15 print("Web server is ready to receive requests.")
16
17 while True:
18     # 等待客户端连接
19     print("Waiting for a connection...")
20     connectionSocket, addr = serverSocket.accept()
21
22     try:
23         # 接收客户端请求数据
24         message = connectionSocket.recv(1024).decode()
25
26         # 打印请求报文
27         filename=message.split()[1]
28
29         filename=filename[1:]
30         print(filename)
31
32         # 检查文件是否存在
33         if os.path.isfile(filename):
34             # 打开并读取文件内容
35             with open(filename, 'rb') as file:
36                 content = file.read()
37
38             # 构建 HTTP 响应报文
39             response = "HTTP/1.1 200 OK \r\n\r\n".encode() + content
40
41         else:
42             # 文件不存在, 返回 404 Not Found 响应
43             response = "HTTP/1.1 404 Not Found 请求失败\r\n\r\n".encode()
44
45         connectionSocket.send(response)
46
47     except IOError:
48         response = "HTTP/1.1 404 Not Found 请求失败\r\n\r\n".encode()
49         connectionSocket.send(response)
```

```
50         connectionSocket.close()
51
52
53 # 关闭服务器套接字
54 server_socket.close()
```

SMTP 客户端

```
1  from socket import *
2  import sys
3  import base64
4
5  # 邮件内容
6  msg = "\r\n send an email from lipeng"
7  # 消息以单个 . 结束。
8  endmsg = "\r\n.\r\n"
9
10 # 选择一个邮件服务器
11 mailserver = 'smtp.163.com'
12 mailserver_SMTP_Port = 25
13 mailserver_SMTP_LoginID = base64.b64encode(b'*****@163.com').decode() ...
    + '\r\n'
14 mailserver_SMTP_Password = base64.b64encode(b'*****').decode() + '\r\n'
15
16 # 163 授权码 EOYPXVDCZVTABAXM
17 # QQ 授权码 vuzchrajlfstdcfc
18
19 From = '*****@163.com'
20 To = '*****@qq.com'
21
22 # 创建 socket 和邮件服务器建立 TCP 连接
23 clientSocket = socket(AF_INET,SOCK_STREAM)
24 clientSocket.connect((mailserver,mailserver_SMTP_Port))
25 recv = clientSocket.recv(1024).decode()
26 print(recv,end = '')
```

```
27 if recv[:3] != '220':
28     print('220 reply not received from server.')
29     clientSocket.close()
30     exit(0)
31
32 # 发送 HELO 命令，打印服务器响应
33 heloCommand = 'HELO 163.com\r\n'
34 clientSocket.send(heloCommand.encode())
35 recv1 = clientSocket.recv(1024).decode()
36 print(recv1,end = '')
37 if recv1[:3] != '250':
38     print('250 reply not received from server.')
39     clientSocket.close()
40     exit(0)
41
42 logCommand = 'AUTH LOGIN\r\n'
43 clientSocket.send(logCommand.encode())
44 recv2 = clientSocket.recv(1024).decode()
45 print(recv2,end = '')
46 if recv2[:3] != '334':
47     print('334 login server goes wrong')
48     clientSocket.close()
49     exit(0)
50
51 clientSocket.send(mailserver_SMTP_LoginID.encode())
52 recv3 = clientSocket.recv(1024).decode()
53 print(recv3,end = '')
54 if recv3[:3] == '535':
55     print('Login ID wrong')
56     clientSocket.close()
57     exit(0)
58
59 clientSocket.send(mailserver_SMTP_Password.encode())
60 recv4 = clientSocket.recv(1024).decode()
61 print(recv4,end = '')
62 if recv4[:3] == '535':
63     print('Password wrong')
```

```
64     clientSocket.close()
65     exit(0)
66
67 # 发送 MAIL FROM 命令，打印服务器响应
68 fromCommand = 'MAIL FROM ' + '<' + From + '>' + '\r\n'
69 clientSocket.send(fromCommand.encode())
70 recv = clientSocket.recv(2048).decode()
71 print(recv,end = '')
72 if recv[:3] != '250':
73     print('Mail From server goes wrong')
74     clientSocket.close()
75     exit(0)
76
77 # 发送 RCPT TO 命令，打印服务器响应
78 toCommand = 'RCPT TO: ' + '<' + To + '>' + '\r\n'
79 clientSocket.send(toCommand.encode())
80 recv6 = clientSocket.recv(1024).decode()
81 print(recv6,end = '')
82 if recv6[:3] != '250':
83     print('Mail to server goes wrong')
84     clientSocket.close()
85     exit(0)
86
87 # 发送 DATA 命令，打印服务器响应
88 beginCommand = 'DATA\r\n'
89 clientSocket.send(beginCommand.encode())
90 recv7 = clientSocket.recv(1024).decode()
91 print(recv7,end = '')
92 if recv7[:3] != '354':
93     print('Data Begin server goes wrong')
94     clientSocket.close()
95     exit(0)
96
97 # 发送邮件内容
98 send = "From: " + From + '\r\n'
99 send += "To: " + To + '\r\n'
100 send += "Subject: " + "First Web Mail Test From lipeng" + '\r\n'
```

```
101 send += msg
102 clientSocket.send(send.encode())
103 clientSocket.send(endmsg.encode())
104 recv8 = clientSocket.recv(1024).decode()
105 print(recv8, end='')
106 if recv8[:3] != '250':
107     print('Data Transport goes wrong')
108     clientSocket.close()
109     exit(0)
110
111 endCommand = 'QUIT\r\n'
112 clientSocket.send(endCommand.encode())
113 recv9 = clientSocket.recv(1024).decode()
114 print(recv9, end='')
115 if recv9[:3] != '221':
116     print('server end goes wrong')
117     clientSocket.close()
118     exit(0)
119 clientSocket.close()
```