



UNIVERSITY OF DHAKA

Department of Computer Science and Engineering

CSE-3111 : Computer Networking Lab

Lab Report 3 : Implementing File transfer using Socket
Programming and HTTP GET/POST requests

Submitted By:

Name: Md Shamsur Rahman Sami

Roll No : 57

Name: Md Rakib Hossain

Roll No : 55

Submitted On :

February 08, 2024

Submitted To :

Dr. Md. Abdur Razzaque

Contents

1	Introduction	2
1.1	Objectives	2
2	Theory of File Transfer Using Socket Programming and HTTP GET/POST Requests	2
2.1	Networking Basics	2
2.2	HTTP Protocol	3
2.3	Socket Programming Framework	3
2.4	Security Considerations	3
2.5	Optimization Techniques	4
3	Methodology: File Transfer Using Socket Programming and HTTP	4
3.1	Understanding Requirements	4
3.2	Designing the System Architecture	4
3.3	Implementing the File Server	4
3.4	Implementing the File Client	5
3.5	Testing and Validation	5
3.6	Performance Optimization	6
3.7	Documentation and Maintenance	6
4	Experimental Result	6
4.1	Task 1	6
4.2	Task 2	11
5	Experience	16

1 Introduction

1.1 Objectives

The primary objective of this lab experiment is to delve into the fundamentals of socket programming while emphasizing practical implementation. Through this exercise, participants aim to comprehend the intricacies of creating and managing network sockets, establishing robust connections between client and server applications, and orchestrating data transmission over a network.

A pivotal aspect of this experiment involves the implementation of a client-server communication model, whereby participants craft both client and server applications capable of seamless interaction. This entails mastering the intricacies of socket creation, binding, and listening on the server side, and connecting to the server from the client side.

Furthermore, the experiment delves into the intricacies of file transfer mechanisms, a quintessential aspect of network communication. Participants navigate the process of reading files from the server and transmitting them to clients, or receiving files from clients and storing them on the server. This involves handling binary data efficiently and ensuring data integrity throughout the transmission process.

An additional focus lies on error handling and robustness, where participants gain insights into implementing mechanisms to address potential issues that may arise during socket communication, such as network disruptions or file transfer errors. By instilling a proactive approach to error handling, participants enhance the reliability and resilience of their socket-based applications, thereby bolstering their proficiency in network programming.

2 Theory of File Transfer Using Socket Programming and HTTP GET/POST Requests

This section provides a theoretical overview of implementing file transfer using socket programming and HTTP GET/POST requests. It covers fundamental concepts, protocols, and considerations essential for understanding and implementing file transfer mechanisms.

2.1 Networking Basics

- **TCP/IP Protocol Suite:** TCP/IP protocols, including IP, TCP, and UDP, form the backbone of network communication. TCP pro-

vides reliable, stream-oriented data transfer, while UDP offers faster, datagram-based communication.

- **Sockets:** Sockets serve as endpoints for network communication, enabling processes to exchange data over a network. Server sockets listen for incoming connections, while client sockets initiate connections to servers.

2.2 HTTP Protocol

- **Fundamentals:** HTTP operates on a request-response model, with methods like GET and POST for retrieving and uploading files, respectively. Headers and status codes provide additional metadata and information about the request/response.
- **File Transfer with HTTP:** HTTP GET requests are commonly used to retrieve files from servers, while POST requests facilitate file uploads. Chunked Transfer Encoding allows efficient transfer of large files in smaller chunks.

2.3 Socket Programming Framework

- **Choice of Language:** Select a programming language with robust networking support, such as Python, Java, or C++.
- **Socket API:** Familiarize yourself with the language's socket API for creating, binding, listening, connecting, sending, and receiving data.
- **Error Handling:** Implement error handling mechanisms to handle connection failures, invalid requests, and other network issues gracefully.

2.4 Security Considerations

- **Authentication and Authorization:** Implement mechanisms to verify user identity and grant access to files based on permissions.
- **Encryption:** Encrypt data transfers to protect sensitive information, especially over public networks.
- **Vulnerability Management:** Stay updated on security vulnerabilities and take measures to mitigate risks.

2.5 Optimization Techniques

- **Buffering:** Use buffers to improve data transfer efficiency by reducing network calls.
- **Compression:** Compress files before transmission to reduce bandwidth usage.
- **Multithreading:** Implement thread-based concurrency for handling multiple file transfers simultaneously.

3 Methodology: File Transfer Using Socket Programming and HTTP

3.1 Understanding Requirements

- Define the requirements for the file transfer system, including supported file types, expected file sizes, security considerations, and performance requirements.

3.2 Designing the System Architecture

- Determine the overall architecture of the file transfer system, including client-server communication protocols, data formats, and error handling mechanisms.
- Choose between socket programming and HTTP for file transfer, considering factors such as network environment, scalability, and compatibility.

3.3 Implementing the File Server

- Develop the file server component responsible for handling incoming file transfer requests.
- For socket programming:
 - Create a server socket to listen for incoming connections.
 - Accept client connections and handle file transfer requests.
 - Read files from the server's file system and send them to clients over the network.

- For HTTP:
 - Set up an HTTP server using frameworks like Flask or Django.
 - Implement endpoints for handling file upload (POST) and download (GET) requests.
 - Receive files uploaded by clients and store them in a designated directory.
 - Serve files to clients upon receiving download requests.

3.4 Implementing the File Client

- Develop the file client component responsible for initiating file transfer requests and handling server responses.
- For socket programming:
 - Create a client socket to establish connections with the file server.
 - Send file transfer requests to the server and receive file contents in response.
 - Save received files to the client's file system.
- For HTTP:
 - Use HTTP client libraries (e.g., requests in Python) to send file upload and download requests to the server.
 - Handle server responses and process downloaded files as needed.

3.5 Testing and Validation

- Test the file transfer system under various scenarios, including different file sizes, concurrent transfers, and network conditions.
- Verify that file uploads and downloads are performed correctly and efficiently.
- Conduct security testing to ensure data integrity and confidentiality during file transfer operations.

3.6 Performance Optimization

- Identify performance bottlenecks and optimize the file transfer system for speed and efficiency.
- Implement techniques such as parallel processing, data compression, and caching to improve transfer speeds and reduce network latency.

3.7 Documentation and Maintenance

- Document the system architecture, implementation details, and usage instructions for future reference.
- Establish a maintenance plan to monitor and update the file transfer system as needed, including security patches and performance enhancements.

4 Experimental Result

4.1 Task 1

- **File Transfer via Socket Programming handling multiple clients**

```
1      SERVER SIDE CODE
2
3  import socket
4  import threading
5  import os
6
7  def handle_client(client_socket,address):
8
9      msg = client_socket.recv(1024).decode()
10     print(msg)
11     response = "This file are avaiable : pdf1.pdf,pdf.pdf,sami.txt"
12     client_socket.send(response.encode())
13     msg1 = client_socket.recv(1024)
14     msg1 = msg1.decode()
15     print(msg1)
16     file_path = os.path.join(r"F:\Python\Python\Networking Lab\Lab 3\Task 1", ms
17
```

```

18
19     try:
20         with open(file_path, 'rb') as f:
21             while True:
22                 data = f.read(1024)
23                 if not data:
24                     break
25                 client_socket.sendall(data)
26             f.close()
27         # except FileNotFoundError:
28         #     print("File not found!")
29     except Exception as e:
30         print("Error occurred during file transmission:", e)
31     finally:
32         client_socket.close()
33
34
35
36
37
38     # with open(file_path, 'rb') as file:
39     # # Your file handling code here
40
41
42     # # file =open("sami.txt", "rb")
43     #     dd=file.read()
44     #     client_socket.sendall(dd)
45     #     file.close()
46     # client_socket.close()
47
48
49
50 def server():
51     #server_address = (socket.gethostname(), 1234)
52     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
53     s.bind((socket.gethostname(), 1234))
54     s.listen()
55     print("Server is listening for incoming connections...")
56
57     while True:

```



```

58
59         client_socket, address = s.accept()
60         print("connection Successfull")
61         client_thread = threading.Thread(target=handle_client, args=(client_socket, address))
62         client_thread.start()
63
64     if __name__ == "__main__":
65         server()
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

```

```

31         file.close()
32     s.close()
33
34
35 num_clients = 1
36
37 # Create and start threads for each client
38 threads = []
39 for _ in range(num_clients):
40     t = threading.Thread(target=client)
41     threads.append(t)
42     t.start()
43
44
45
46 for t in threads:
47     t.join()
48

```

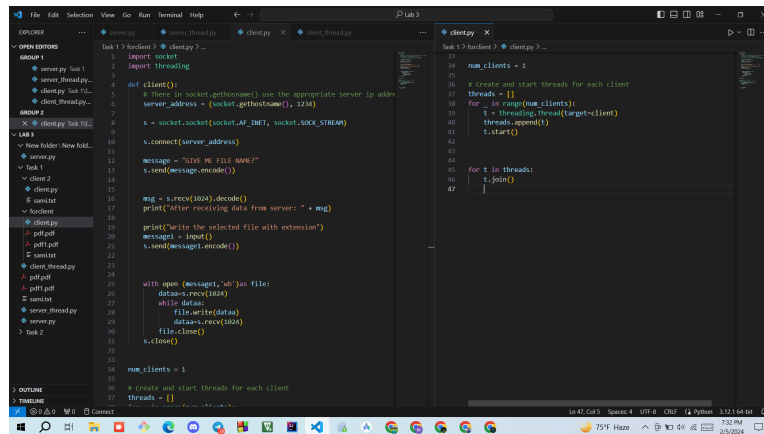


Figure 1: Client side code

```

1 import socket
2 import threading
3 import os
4
5 def handle_client(client_socket,address):
6
7     msg = client_socket.recv(1024).decode()
8     print(msg)
9     response = "this file are available : pdf1.pdf,pdf2.pdf,smi.txt"
10    client_socket.send(response.encode())
11    msg1 = client_socket.recv(1024)
12    msg1 = msg1.decode()
13    print(msg1)
14    file_path = os.path.join(r"Python\Python\Networking Lab\Lab 3\")
15
16    try:
17        with open(file_path, 'rb') as f:
18            while True:
19                data = f.read(1024)
20                if not data:
21                    break
22                client_socket.sendall(data)
23                f.close()
24            # except FileNotFoundError:
25            #     print("file not found")
26    except Exception as e:
27        print("error occurred during file transmission:", e)
28    finally:
29        client_socket.close()
30
31 # with open(file_path, 'rb') as file:
32 #     # your file handling code here
33
34
35
36
37
38
39 # # file open("smi.txt","rb")
40 # # do file read()
41 # # client_socket.sendall(data)
42 # # file.close()
43 # # client_socket.close()
44
45
46
47
48 def server():
49     server_address = (socket.gethostname(), 1234)
50     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
51     s.bind((socket.gethostname(), 1234))
52     s.listen()
53     print("Server is listening for incoming connections...")
54     while True:
55         client_socket, address = s.accept()
56         print("connection successful")
57         client_thread = threading.Thread(target=handle_client, args=(
58             client_socket, address))
59         client_thread.start()
60     if __name__ == "__main__":
61         server()
62

```

Figure 2: Server Side Code

```

C:\Windows\System32\cmd.exe
Python\Python\Networking Lab\Lab 3\Task 1\forclientpython -> client.py
After receiving data from server: this file are available : pdf1.pdf,pdf2.pdf,smi.txt
write the selected file with extension
smi.txt
Python\Python\Networking Lab\Lab 3\Task 1\forclientpython

```

Figure 3: Client Side result

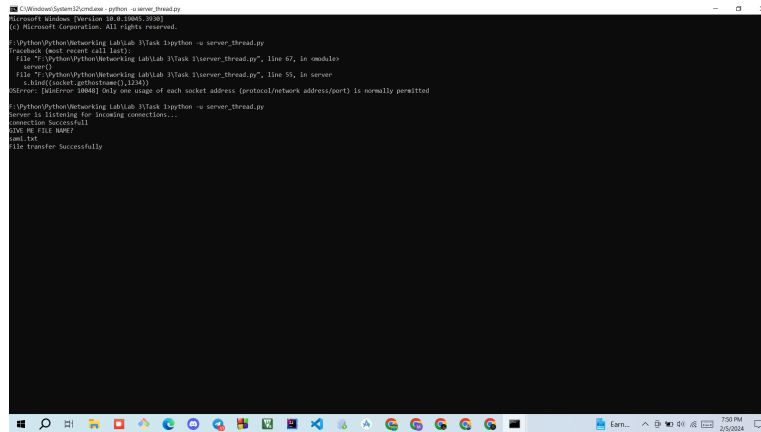


Figure 4: Server Side result

4.2 Task 2

- File Transfer via HTTP handling multiple clients simultaneously.

```

1     SERVER SIDE CODE
2     from http.server import BaseHTTPRequestHandler, HTTPServer
3     import os
4
5     class FileServerHandler(BaseHTTPRequestHandler):
6         def do_GET(self):
7             try:
8
9                 if self.path == '/':
10
11                     file_path = 'index.html'
12                 else:
13
14                     file_path = self.path[1:]
15
16
17                 if os.path.exists(file_path) and os.path.isfile(file_path):
18                     with open(file_path, 'rb') as file:
19                         self.send_response(200)
20                         self.send_header('Content-type', 'application/octet-stream')

```

```

21         self.send_header('Content-Disposition', f'attachment; filename={filename}')
22         self.end_headers()
23         self.wfile.write(file.read())
24         print("Client Get Request ")
25
26     else:
27         self.send_response(404)
28         self.end_headers()
29         self.wfile.write(b'File not found')
30
31     except Exception as e:
32         self.send_response(500)
33         self.end_headers()
34         self.wfile.write(str(e).encode())
35
36 def do_POST(self):
37     try:
38         content_length = int(self.headers['Content-Length'])
39         file_content = self.rfile.read(content_length)
40
41
42         unique_filename = f'received_file_{content_length}.txt'
43
44         with open(unique_filename, 'wb') as file:
45             file.write(file_content)
46
47         self.send_response(200)
48         self.end_headers()
49         print("Client Post Request ")
50         print("Save it in server disk ")
51         self.wfile.write(b'File saved successfully')
52
53     except Exception as e:
54         self.send_response(500)
55         self.end_headers()
56         self.wfile.write(str(e).encode())
57
58 def run_server(port=8080):
59     try:
60         server_address = ('', port)

```

```

61         httpd = HTTPServer(server_address, FileServerHandler)
62         print(f'Starting server on port {port}')
63         httpd.serve_forever()
64
65     except KeyboardInterrupt:
66         print('Server stopped')
67
68 if __name__ == '__main__':
69     run_server(port=8080)
70
71
72

```



```

1     CLIENT SIDE CODE
2     import requests
3
4     def download_file(url, save_path):
5         response = requests.get(url)
6
7         if response.status_code == 200:
8             with open(save_path, 'wb') as file:
9                 file.write(response.content)
10            print(f'Downloaded file: {save_path}')
11        else:
12            print(f'Error downloading file. Status code: {response.status_code}')
13
14    def upload_file(url, file_path):
15        with open(file_path, 'rb') as file:
16            files = {'file': (file_path, file)}
17            response = requests.post(url, files=files)
18
19        if response.status_code == 200:
20            print(f'File uploaded successfully to {url}')
21        else:
22            print(f'Failed to upload file. Status code: {response.status_code}')
23
24    if __name__ == '__main__':
25        server_url = 'http://localhost:8080'
26
27        # Example: Downloading a file

```

```

28     download_file(f'{server_url}/request_download.txt', 'downloaded_file.txt')
29
30     # Example: Uploading a file
31     upload_file(server_url, 'upload_from_client.txt')
32

```

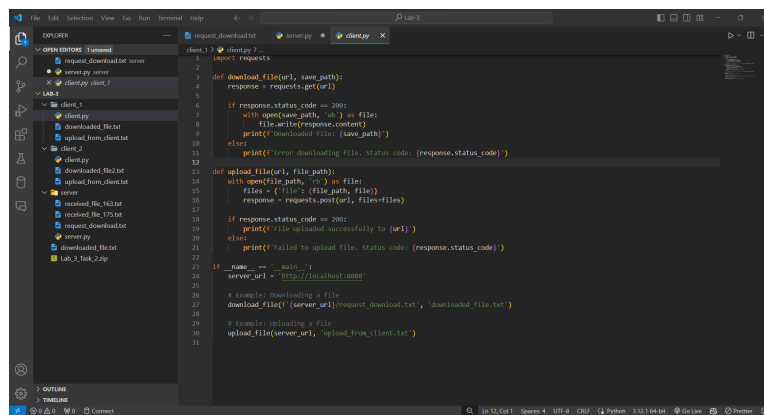


Figure 5: Client side code

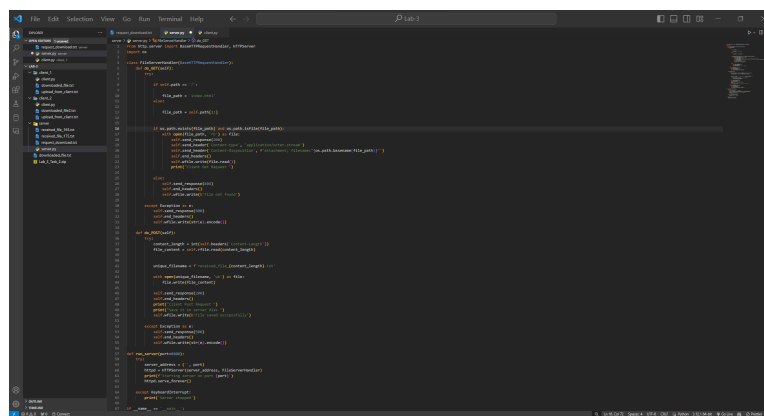


Figure 6: Server Side Code

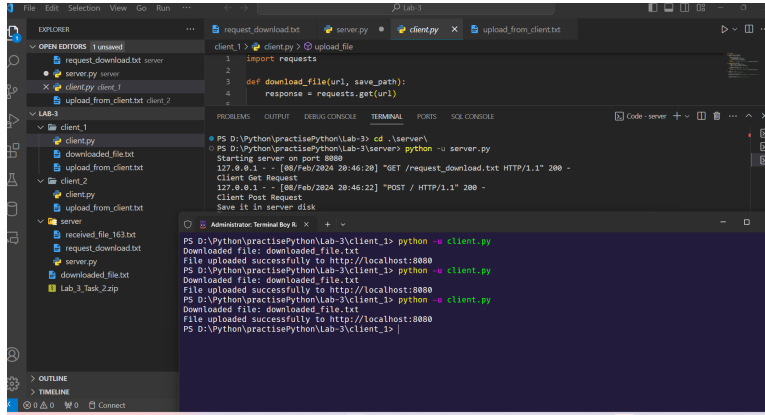


Figure 7: Result for Single client

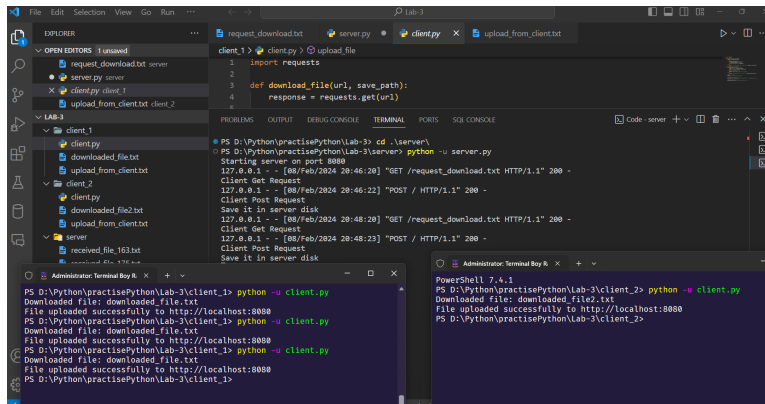


Figure 8: Result for multiple client

5 Experience

1. Ensuring that the server and client handle user inputs safely and validate file paths and names.
2. We have create multiple clients , thats why we needs more than one terminal for clients side and one for server sides
3. First we run server side program and then client request server for specific task
4. Server handle multiple clients request simultaneously, using treads programming
5. Handling different HTTP methods: Supporting both GET and POST methods for file transfer.
6. Identifying and resolving issues related to socket communication, file handling, or HTTP protocol implementation.

References

- [1] Socket Programming in Python: <https://www.geeksforgeeks.org/socket-programming-python/>
- [2] Socket Programming in Python: <https://realpython.com/python-sockets/>
- [3] File Transfer via HTTP: <https://www.freecodecamp.org/news/simplehttpserver-explained-how-to-send-files-using-python/#:~:text=Open%20browser%20and%20type%20in,settings%20on%20the%20first%20computer.>
- [4] File Transfer via HTTP: <https://www.youtube.com/watch?v=DeFST8tvтуI&t=630s>