Index

Index	**1**
Introduction	**2**
Client Program Description	**3**
Index Server Program Description	**6**
Results	**8**
Conclusion	**9**
Source Code Peer	**10**
Source Code Server	**21**

Introduction

The goal of this project is to implement a Peer-to-Peer application that will allow users to host and/or download different files through the use of UDP and TCP sockets. Sockets allow the communication between two devices by opening endpoints to send/receive information to and from. The two protocols to be utilized for socket programming in this project are UDP and TCP, the main difference between the two protocols is that TCP requires three verification requests before transmitting data while UDP enables communication with no regard for any availability requests. Due to this key difference TCP is the slower but more reliable protocol more suited for file transfer, while UDP is more suited for establishing quick connections. Within the application there are two different classifications of users that must be considered. A user that wishes to download a piece of content is known as a content client while a user that owns the file to be downloaded is known as a content server. An index server is to be developed to support the processes of connecting a content client to a content server. The communication between the index server and a peer is handled through the use of a UDP socket due to its quickness while content download is accomplished through TCP sockets due to more reliable transfers. The application must implement five functionalities, content registration, download, listing, de-registration, and quitting. Content registration is when a peer chooses to make a file they own available for download, while content de-registration is when a peer wishes to remove a file they own from the online server. Content download is when a user wishes to download a file from the available files on the online server. Finally, content listing allows a user to see the list of files available for download and if a user selects to quit their hosted content will be de-registered.

Client Program Description

The client program code implements all the functionalities available to a client user and communicates with the index server. All the communication is accomplished by sending PDU structs, which are custom data types containing two fields. The first data field in the PDU struct is a char referred to as “type” and the second is the char array “data”. The PDU type is used to identify the type of data being transmitted, while the data array holds the information to be transmitted. There are eight different PDU types, each representing a different function, refer to **Table 1** below for the classification of each type. Upon starting the client application, the user will be faced with an interactive text based menu promoting them to choose a function to execute, the functions correspond to PDU types ‘R’ , ‘D’, ‘S’, ‘T’ as well as a fifth option ‘Q' to quit.

Type	**Function**
R	Content Registration
D	Download Request
S	Search Request
T	De-register request
C	Content Data
O	List Registered Content Request
A	Acknowledgment
E	Error

Table 1: PDU Type classifications

If the user selects to register a piece of content by entering ‘R’ into the console then the corresponding registration function will execute. This function contains all the code dealing with

the content registration process. To register a file the peer will enter in the desired file name that exists locally on their computer along with a peer name. Upon verifying the existence of the desired file, the application will begin to compile an ‘R’ type PDU by completing the following steps. Ideally the port at which the file is registered at would be dynamically chosen using the getsockname() function and setting htons() parameter to 0. However, we were not able to implement this functionality properly, so instead the user is asked at which port they would like to host their file and this port is saved for download and hosting use. Next, the data array in the PDU will be populated with the name of the peer registering the content and the file name. The ‘R’ type PDU will then be sent to the index server using UDP, if this transmission is successful an ‘A’ type PDU will be returned, the piece of content has been successfully registered and global counter variables will be modified to reflect the new piece of registered content. If instead an ‘E’ type PDU is received then the registration process has failed. Once a peer successfully registers a piece of content, they are now considered a client server, thus they must be able to “host” a piece of content to be downloaded by other users. This is handled by the host function which runs as a background process to the main program execution. The host function opens a TCP socket at the port where the content is registered, it then listens for any download request, if one is received then the content data will be sent to the client who sent the download request.

Next is the download functionality which occurs when a user enters ‘D’ into the main menu. The download process starts with the user entering the filename of their desired piece of content and their peer name. This information is sent to the index server as a part of an ‘S’ type PDU to search for the registered content. If the entered content exists, the index server will respond with a ‘D’ type PDU containing the TCP port of the socket where the content is available for download. If an error occurs during this process the server will instead respond with

an ‘E’ type PDU. The peer program will use the port address received from the ‘D’ type PDU to establish a TCP connection. Once the connection is successful, the peer will send a ‘D’ type PDU to the client server, contiang the filename to be downloaded. The client server responds with a series of ‘C’ type PDUs in which the data field is populated with the files contents. The client server may send multiple ‘C’ type PDUs depending on the size of the transmitted file. The data field contents will be copied into a file created by the peer program, thus successfully completing the file transfer to the content client. After the content client user downloads the file, they can now act as a content server for that file, thus a new registration request is made at a new port address for the file and the old content server is removed. Removing the old content server acts as a sort of load balancer as it ensures that content servers will not be dealing with more than one download request.

Third, a client can see all the available content to download by selecting option ‘O’ from the menu. This functionality works by sending the ‘O’ type PDU to the index server, and receiving back a PDU containing the filename of the hosted file. The client will continue to read the data from the PDU being sent until the type is changed to ‘F’, indicating this is the final filename that is being hosted.

Moving on, a client can also choose to de-register a piece of content that they have previously registered on the index server. To start this process the program first checks that the user requesting to de-register a file has in-fact registered a file previously. After verifying this, the program will prompt the user to enter in the desired filename they wish to have de-registered. Again, upon verifying this file exists by iterating through a local list of registered files, and verifying that the user requesting to de-register the content is the same user that registered the content, a ‘T’ type PDU will be sent to the index server. This PDU contains the filename to be

de-registered, upon successful de-registration, the index server will respond back with a ‘T’ type PDU confirming the file is removed, and the local file list will be updated to reflect the new content.

Finally, the last option available to a user is quitting the program which occurs when a user enters ‘Q’. When a user quits the program all of their registered content must be removed from the list of hosted content as they are no longer an active content client. This is accomplished by iterating through the local list kept of client registered content and sending a series of ‘T’ type PDUs where the data field is the filenames from the list. Lastly, the user’s hosting process is terminated because they are no longer a content client of any file.

Index Server Program Description

The index server program for this project handles the peer requests and connects the different peers on the network together. Unlike previous labs conducted for this course, the index server does not hold any of the registered content, it instead holds the information of the content server peer and provides this information to the content client peer upon request. During a file registration process the index server will receive an ‘R’ type PDU over UDP protocol. Upon receiving this PDU, the index server will check if another peer with the same name has registered the same piece of content. If this is the case then the server will return an ‘E’ type error PDU, if not the content has been successfully registered and an ‘A’ type PDU will be returned. Similar to the client program, the index keeps a local content list that holds information such as content name, peer name, and the port at which the content server exists.

The index server processes a content download request by first receiving the ‘S’ type PDU. This PDU contains the content name and tells the server to look for the port corresponding

to the received name. If the server is able to find the file, it will respond with a ‘D’ type PDU sending the corresponding port of the socket at which the content client is located. If the file cannot be found then the server responds with an ‘E’ type error PDU. The peer uses this information received from the server to connect to the content server and continue their download process. The server will register the peer that sent the download request as a new content server, as they now own the registered file, the local content list will be updated accordingly.

To process a content listing request the server receives an ‘O’ type PDU from the user. The index server then searches its local content list that holds all the registered content information and returns it to the user. The index server sends this information using a series of ‘C’ type PDUs containing the file information, the final transmission will be sent using an ‘F’ type indicating the final data transmission.

Finally, upon receiving a ‘T’ type PDU from the user, the server will begin the de-registration process. To de-register a piece of content the server extracts the file name and peer name from the ‘T’ PDU to ensure that the user de-registering the content owns that content. The verification process is done by comparing the sent data to the index server's local list of content. Once verified, the server removes the corresponding file and peer name from the local list so that the content is no longer registered and sends an ‘A’ type PDU to confirm the successful de-registration. If the deregistration process fails for any reason, then an ‘E’ type error PDU will be sent.

Results

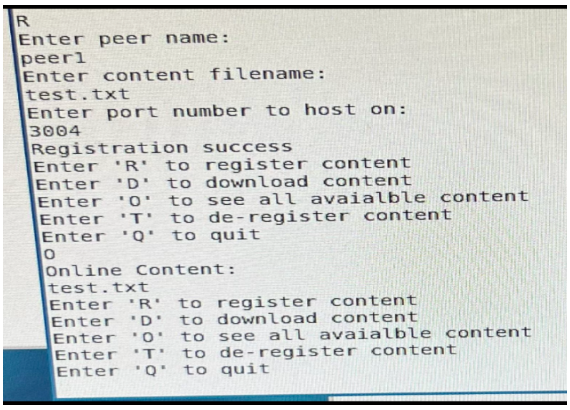

Figure1: Client

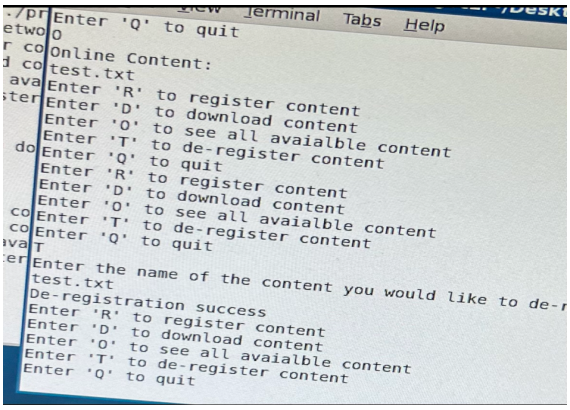

Figure 2: Client continued

Depicted above are the console outputs from running the peer program. As you can see in **figure 1** the user first enters in 'R' which will start the registration process. The corresponding prompts for registration are displayed in which the user enters the peer name, filename and port number. The program responds with the 'Registration success' message which was sent from the index server. Next the User enters 'O' to see the registered content, and the previously registered 'test.txt' file is displayed. Looking at **figure 2,** the user enters 'T' to start a de-registration request. After the correct name of the file is entered the user is met with the success message. The corresponding server console can be seen below in **figure 3**. This showcases examples of the functionalities available to the users of this program.

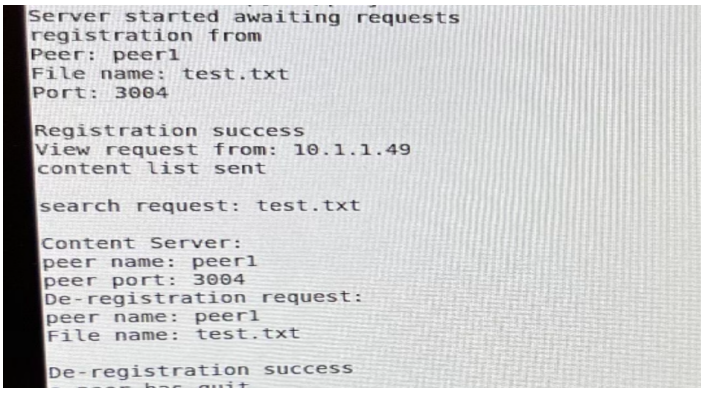

Figure 3: Server

Conclusion

In conclusion, all of the functionalities outlined in the lab report have been implemented, besides dynamically assigning TCP ports. The functions available to a user are register, de-register, view online content and quit. File transfer is accomplished using the reliable but slower TCP socket connection while communication between the index server and client is done through the quicker UDP connection. A custom data structure referred to as the PDU is the data being sent to and retrieved from the sockets. The PDU has two data fields, a character called ‘type’ to indicate the type of PDU being transmitted and a character array called ‘data’ holding the actual data being transmitted such as the filename, peer name or file data. All of this semester's work culminates into this final project, the building of a peer-to-peer network program for file sharing.