COP 5570 – Concurrent, Parallel and Distributed Programming

# 

# Department of Computer Science

# Final Report

# Project Title:

# Peer-to-peer on-line gaming server

## Submitted by:

## Vaibhav Soni, Ratnesh Patidar, Gaurav Sinha

**1. Introduction**

In contrast with a client-server network model which is centralized considering the functionality of the server, a peer-to-peer network is a distributed and decentralized network where in distinct peers in the topology perform as either resource producers or resource consumers. In a peer-to-peer network the clients are distributed and connected in a manner that promotes decentralization by acting as servers to serve other clients which may be connected to them, thus relieving the central server off maximum load and processing. Hence, the entity to entity service granting is now distributed. The multi-client network may be categorized as authoritative referring to a central entity as a server and a non-authoritative referring to a multiplayer network topology under which all clients would maintain their own game states.

Our project relates to the area of Developing Gaming Infrastructure over a Peer-to-peer network. The project is an expansion of the tic-tac-toe gaming project completed at a prior time. The previous project involved a client-server centralized model that showcased numerous clients contacting the server with the intention of playing a tic-tac-toe game among them. In the previous scenario, scalability was the major drawback as the server was presented with heavy load due to sharing of its commodities with all the clients and responding to the requests received in extremely short spans of time. Therefore, the current project focuses on distributing the load from the central server to the clients wherein in a pair of clients involved, one would possess the responsibility of the server, sharing its resources and one would function as the client, requesting and consuming the resources. Several such clients would exist in the gaming network.

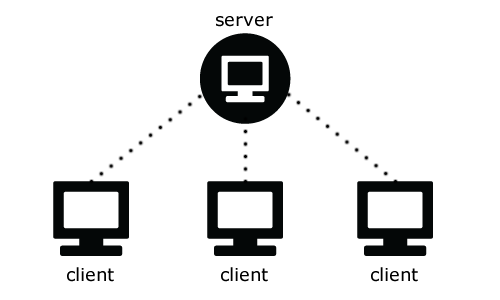
**2. Requirements and Terminology**

|  |  |
| --- | --- |
| **Host** | A different term for a system. |
| **Node** | A networked device (generally a host). |
| **Peer-to-peer Network** | A network of systems (*peers*) in which each can communicate with the other, generate and respond to requests for data and access to shard resources.  (Sometimes referred to as *p2p* networks.) |
| **Client-Server** | Sometimes referred to as *server based* networking, the more common and known genre of network in the current technological era. |
| **Client** | A node that transmits requests to servers.  The term could also refer to any software or user that generates requests to a server and it does not necessarily have be a separate hardware entity. |
| **Server** | A node that exists to respond to and serve the clients.  (A server does not necessarily have to be a separate hardware component.) |
| **TCP** | (Transmission Control Protocol)  A layer 4 protocol that provides a reliable connection-oriented service. |

**Socket** It refers to the endpoint of the flow of inter-process communication over a network of computer systems

**Server-clients** Clients which act as a server when another client is connected with it.

**3. Classic Approach**

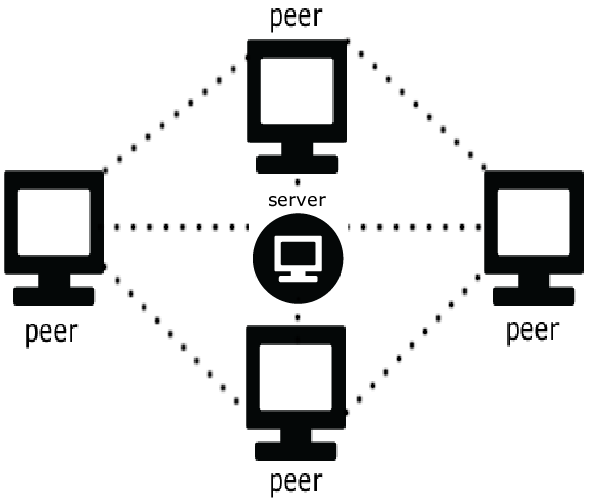


An authoritative or classic gaming network consists of every node under the topology assuming two roles, the server which shares its resources or the client which initializes a contact with the former in order to request to utilize the corresponding resources. But the applications of a client-server model reach out to a certain category of computer networks, namely the World Wide Web, Electronic mail (E-mail) and network printing. Communication takes place via message passing between the clients and the server. Thus, this inter application communication results in the requesting of resources by the clients and the server granting the resources via responses.

However, the server receives requests from countless number of clients and that too in a short span of time. As there exist constraints on a system’s performance limit, the server applies scheduling and prioritization to accommodate, respond and serve all the request it receives. To maximize the throughput over the network, the server as it possesses authority applies constraints and restrictions to the methodology in which the clients may use its resources. Even after applying all the restrictions, the network may suffer from denial of services attacks or processing deficiencies in relation to the server. Therefore it becomes obvious that not all network topologies are not suited to a client-server authoritative model.

Some functionalities like printing, electronic mailing and using the internet and thus require data to be stored at a centralized location for access and availability criteria and hence storage of the information or data occurs at central location, namely the server. In such a scenario, the client is not going to send requests to the server in short intervals as compared to what happens over a gaming network. So, the client-server network topology is not apt for all categories of distributed systems.

**4. P2P Approach for Multiplayer Gaming (Current Project)**

****

Initially the clients are separate individual standalone processes. They request access to the gaming network via connection to the server. The connection is established if certain validations for login names and passwords are met or if the user has registered for the first time. After, this stage, the server would establish a connection among the corresponding peers or clients who wish to play a game of tic-tac-toe or want to mail, chat or create a conference chat. The clients could also perform all the activities in a parallel manner.

Previously, the client-server model project involved the server storing the states of clients in its domain. In the current methodology one of the clients involved in a game would act as a server and would store the state of the clients under its domain. The server is offloaded but the connection with the clients is not broken. Whenever a game is completed, a client resigns or a client quits, messages are sent to the server to store only certain information.

Offloading is accomplished as the short-span communication from the countless clients to the one server has now been distributed to certain clients which act as servers. But, a central repository is required. Some tasks handled by the server are to communicate to the clients as to which clients are online, what are their statistics with respect to a particular game, email messaging, authentication mechanisms and all the tasks that do not required over loading it with simultaneous hoards of requests.

Multithreading is used to deal with the multiple communications among the clients and between the clients and the server. It should be noted that in this architecture clients also act as servers and we refer to them as server-clients. As the server is started, it waits for clients to send requests to it and to perform authentication and then establish connection. The total number of connections with the server cannot be greater than the number of threads spawned by the server. But as soon as a client quits, the thread is ready to be utilized by another client waiting in line to play a game or just establish a connection. During over loaded activities now handled by the server-clients, even if the server is shut down intentionally, certain message passing can still take place between the clients and the server-clients via the other connection.

As soon as the clients are authenticated by the server and are involved in a game, communication takes place via different sockets on their respective ports. Thus, communication still persists even if one turns down the server on purpose. The gameplay would persist without any involvement of the server. The purpose of the server with respect to gameplay would be to hold records regarding a particular game.

**5. Over-all features implemented in the Game**

1. who # list all users online
2. stats [name] # display information about [name]
3. match <name> <b/w> [t] # try to play with [name] as b/w in a t-sec game
4. resign # resign from a game
5. game # list all current games
6. observe <num> # observe game <num>
7. unobserve # unobserve a game
8. refresh # redisplay an in-play or observed game
9. kibitz <msg> # comment on an observed game
10. ' <msg> # another form of kibitz
11. tell <name> <msg> # tell user <name> message
12. shout <msg> # shout <msg> to every one online
13. mail <id> <title> # send id a mail
14. listmail # list the header of the mails
15. readmail <msg\_num> # read the particular mail
16. deletemail <msg\_num> # delete the particular mail
17. info <msg> # change your information to <msg>
18. passwd <new> # change password
19. exit or quit # quit the system
20. help # print this message
21. register <name> <pwd> # register a new user

**Features off-loaded from the server side to client**

1. match <name> <b/w> [t] # try to play with [name] as b/w in a t-sec game
2. resign # resign from a game
3. game # list all current games
4. observe <num> # observe game <num>
5. unobserve # unobserve a game
6. refresh # redisplay an in-play or observed game
7. kibitz <msg> # comment on an observed game
8. ' <msg> # another form of kibitz
9. help # print this message

**6. Approach Followed**

Initially for this project, we had the advantage that we had already developed the client-server version of the tic-tac-toe game server, which was fully functional and worked fine with telnet client. So we first thought of developing the client version code which should work similar to the telnet. Once, we were able to do this then we thought of establishing a new connection between two clients, but this became too tedious and we got stuck here as the code became too complex and messy on both sides. So we thought of starting with the basics, the approach that we used for the project 4, that is, to first start with the peer-to-peer echo-client server model and then build a game server out of it.

Through this approach, finally we were able to first establish a p2p echo-server, in which the client will first send connection request to the server, once the connection gets accepted, the client will the create two threads, one to send messages to the server, that is, handle the input terminal and the second thread to print the data received from the server. Meanwhile the main process binds itself to a specific port and gets blocked on the accept connection state loop. Also the client sends the port and address information to the server, which the client stores in a data structure along with the name of the client.

Now to explain the connection establishment between the clients more clearly, we consider a scenario of two clients A and B. On the server side we specified a condition in which if the server receives the “Connect B” from a client A then the server sends the port number of client B to the client A. Then the client A will send connect request to the client B using the address and the port number received from the server, B which is already blocked on the accept state, accepts the connection and creates new two new threads to interact with the client A and client A will also create two new threads to interact with the B. Through this A got connected to both the server and with B and now to will get the echo message from both the server and client B. By following this approach we were successfully able to connect two clients and if we closed the server, the clients were still able to communicate with each other and were able to send echo messages.

Taking this as the basis for the peer to peer connection, we started building the game server, around the sample p2p echo server.

**7. Conclusion**

We were successfully able to off-load the gameplay portion from the server to client. This was a very significant portion removed from the server side as, in the client-server version of the game the server code was of 1700 lines (approximately) with nothing on the client side (used telnet instead), while now the server is of mere 1000 lines and the game play chunk is now in the client side which is approximately 1300 lines of code.

Another thing implemented was that in some situation if the server goes down and there are games going on amongst the clients, then game doesn’t get interrupted due to the connection lost with the server, this is because the game is being played on the new connection.

The game server has become more scalable as majority load from the server has been lifted off. In future we intend to shift more features from the server to client side.

**8. Future Enhancements**

In particular, the number of threads spawned by the main thread in the server represents the number of clients in the peer-to-peer gaming network architecture. No more clients could thus join when all the threads are being used at a point in time.

Also we intend to include our own features, like a “friend request”. In this feature, a client can send a friend request to the client currently online. Once the friend request is accepted by the other client then that clients name is permanently stored in the friend list and whenever that client will come online the all its other online friends will be notified that he is online and available to play game. But this may increase the work load on the server and may not the be a good option if the motive is to offload the server.

Another feature that we intend to add in the game is “Rematch”. In this feature, after the game is over between two clients, the client who lost the game will be prompted with a message “Challenge the player for a rematch”, this gives him an option to say “yes” or “no”. If he says “yes”, then the winner client gets prompted about the rematch request and if that client says “yes” to the request the match gets started and the server gets notified about the started game. But if either of the clients says “no”, then the connection gets closed. And if both clients want to play game with each other then they will have to use the “Match” command.

Also we intend to a GUI for the game, as now that will be much easier because we have already developed a client code and the GUI portion will go in the client side and the will not cause any overhead on the server.

