Proxy Server for the Collaborative Virtual Environment

Jayanthan Elumalai

Department of Computer Science

New Mexico State University

Advisor: Dr. Clint Jeffery

Abstract

The goal of this project is to develop a proxy to be deployed at remote sites where multiple users will be accessing the Unicron Collaborative Virtual Environment. The goal of the proxy server is to minimize of the amount of traffic generated over the wide area network because of the connections that exist between the clients in the remote place and the central server. In a distributed system, the deployment of proxy servers will help increase the scalability of the overall system, and reduce the load of the Central Server.

Reformatted with minor corrections and possible errors by Jeffery from a mixture of the last .doc received and the final version PDF.

1.0 Unicron – A Collaborative Virtual Environment

This section introduces the Unicron Project. Unicron is a Collaborative Virtual Environment(CVE) being developed at the NMSU CS department. In the Unicron CVE, clients are represented by avatars. Clients can move around, see each other, communicate with each other, transfer files between them and perform all sorts of group activities [1].

The Unicron server supports each client through TCP connections. The communication between the clients is done in a centralized manner. There are different types of communication messages,

(1) Client sends message to the all other clients
(2) Client sends a message to other clients in a particular group
(3) Server sends message to a single client in response to an event.
(4) Server broadcasts a message to all clients in response to an event
(5) Server broadcasts a message to a subset of clients. For example clients in a particular group or room etc.

Since the clients do not have direct connections to each other, communication between ' two clients passes through the server. So the message types 1 and 2 also involve the server. We analyze this more thoroughly in the coming sections. Section 2 gives the literature review. Section 3 motivates the proxy and talks about the network protocol between proxy and server, and presents various modules in the proxy server code. Section 4 is a performance analysis of the proxy and server. Section 5 is the conclusion.

2.0 Related Work

Proxy servers are not new in a distributed environment like CVEs. Proxy servers are used in Internet, to provide faster retrieval of web pages[5]. One of the CVEs which uses a proxy is the DIVE CVE [2]. The proxy server in DIVE is used to multicast messages to all the clients in the multicast group. Proxy servers are also used in streaming multimedia applications, especially video applications[3]. Proxies are related to the idea of file objects inside internetworks [4]. Judiciously placed file caches reduce the volume of FTP traffic and NSFNET backbone traffic. Danzig et al assert that the performance of all distributed systems depends on caching, and if they don't cache they don't perform well. A hierachical arrangement of caches mirroring the topology of wide area network can be used to distribute the load away from the server[6]. These papers share a common theme which is "Caching". The basic principle of all proxy servers is caching, although they can differ to a great extent on what they cache and how they use caching. In the Unicron CVE, the proxy stores the room information of clients so that it can serve future client requests (for example the \move command). Caching is just a part of the unicron proxy server. It also does other functions such as multicasting and forwarding messages.

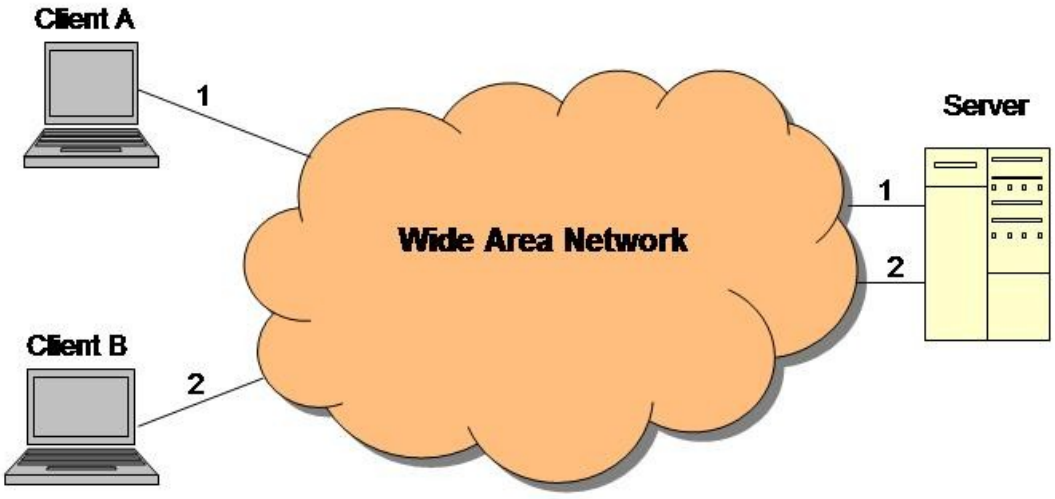

Figure 1 – Unicron Server - Client communication through Wide Area Network

3.0 Proxy Server

Consider two clients A and B which are located at a remote campus. They are connected to the unicron server at the NMSU campus. If client A sends a message to client B, the message goes to the server and the server sends the message to client B. Though the clients are located in the same network, the message traverses an external wide area network, reaches the server, and returns to the destination traversing the same wide area network (Figure 1). If the network distances between clients A and B and the central server is not very big or if the size of the communication messages are very small, then there is not much of a gain from optimizing the overall bandwidth utilized and delay. But on the other hand if there are frequent communications between the two clients and the size of each message is high, a considerable performance improvement can result from optimizing the communication overhead. Since there is a high probability that the latter case occurs often, the existence of a proxy server at the remote campus aimed at reducing the communication overhead is justified.

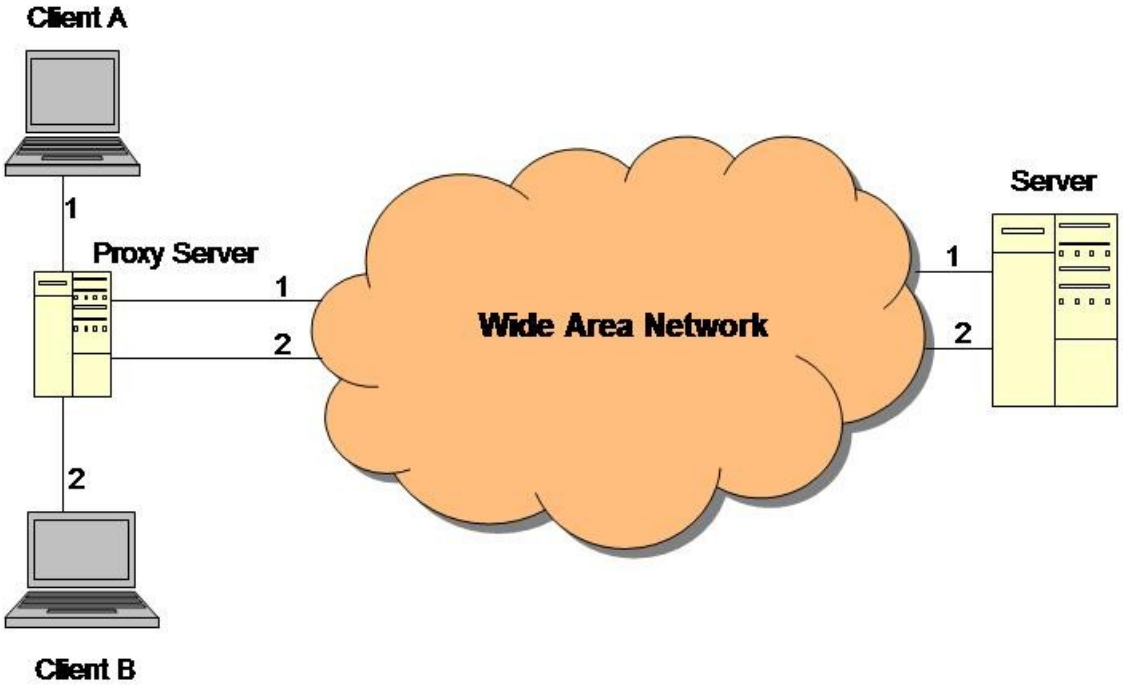

Figure 2 – Unicron Server - Proxy - Client communication through Wide Area Network

Having established the case for the development of a proxy server, consider the details. Figure 2 shows how the proxy server sits in between the Server and the Clients. In order to achieve its goals, the proxy server has to make intelligent routing decisions by reading the messages it receives from the client. The rest of this paper discusses the design of the Unicron proxy server.

3.1 Proxy Server Architecture

This section shows the organization of the source code of the proxy server. The proxy maintains a set of sockets and uses the select() system call to monitor those sockets. The select() call reports the sockets that have changed status. The proxy then does a read() from those sockets. Figure 3 shows the basic architecture. There exists a one-to-one mapping between the client sockets and server sockets.

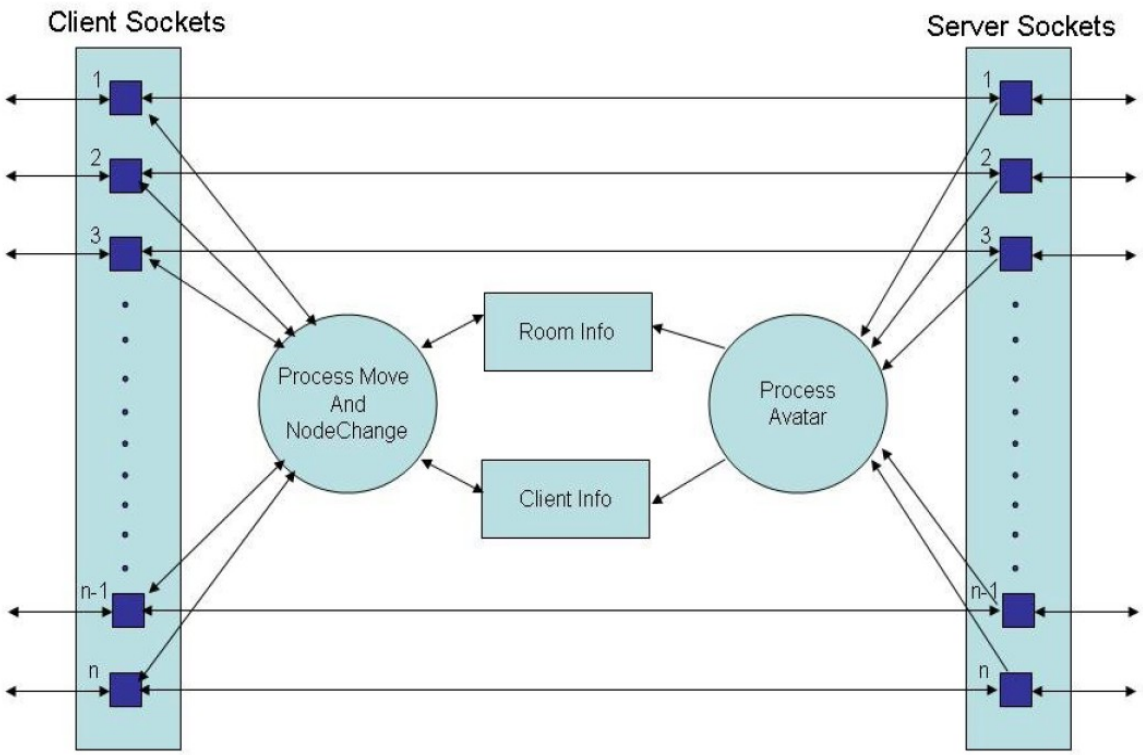

Figure 3: Proxy Server Architecture

3.2 Network Protocol

The protocol between proxy and server is very simple. The messages the proxy receives from its clients are divided into 2 types.
Type 1: These are messages the proxy code processes. (i.e. it knows what to do with that message). For example the \move command. Currently, this is the only command in this category out of the total 42 commands. When we consider the amount of bandwidth utilized, the \move commands constitute more than 60% of the overall network bandwidth used by the CVE. See section 4 for measurements.
Type 2: These are messages the proxy does not process.

For Type 1 messages the Proxy-Server protocol is as follows.
The Proxy forwards the message to the server through the corresponding socket. The server receives the message. Assume that this message triggers a response message M from the server. Now if M has to be sent to a subset of clients, the server will not send M to the clients which are connecting through the proxy. This is because the server knows that the proxy has the code to handle type 1 messages.

For Type 2 messages, the normal Client-Server protocol is followed. The proxy forwards Type 2 messages transparently. It reads some Type 2 messages to maintain awareness of avatar locations.

3.3 Initial Setup

When a client logs into the proxy server, the proxy makes the connection for that client with the central server. The proxy server sends a command to the central server informing it that this client is indeed connecting through the proxy. This information is helpful for the server in future. Also when a client logs out, the proxy server will issue a command to the server. The following are commands which are supported by server for synchronizing with the proxy.

(1) \proxynumber "number"
    Proxy will send this command when it starts up. The server will maintain a list of proxies which are currently running.
(2) \delproxy "number"
    Proxy will send this command just before it closes. The server will remove the proxy from its list.
(3) \proxylogin "uid"
    Proxy will send this command when a Client has logged in.
(4) \proxylogout "uid"
    Proxy will send this command when a Client logs out.

All the messages that are being communicated are in the form of commands. As a first step we will figure out some ways to optimize the commands. These are the list of some commands,

(1) \say hello – chats to the current room
(2) \say joe hello – chats to a specific person
(3) \move 0, 0, 0.5 – move to (x,y,z)

We specifically concentrate on the \move command for now.

3.4 The \move command

Whenever a client moves, a "\move x,y,z" command is generated by that client and sent to the server. So normally when a client moves, it'll generate \move commands continuously. (One \move command is generated for every 0.15 units of displacement). The server maintains a dynamic list of clients' room information. When a client moves from one room to another, it updates that dynamic list. Also whenever the server receives the \move command from a client, it figures out which other clients need to know about this move, and then it sends the move to those clients.

The server figures out which clients need to be informed of a move using some visibility rules. For example, the clients who are in the same room as the client that moved and also the clients in adjacent rooms. Basically the \move has to be sent to all the clients who are

in close vicinity and the clients which can actually see that move. Currently the visibility rule is this : If a client A moves, then send that move to all the clients in that room, the adjacent rooms and the adjacent rooms of the adjacent rooms.

The idea is this: If the proxy server has the knowledge of the visibility rules and the dynamic room information of each client that has logged into it, then it can make the decision of what other clients are to be informed of the move without having to contact the central server. Now that the central server knows the proxy will take care of the \move command for the clients which are logged into it, it will not have to send the \move if the recipients that use the proxy. So some redundant information and some computation at the proxy enable performance optimizations. In order to optimize, the proxy has to have the room information and visibility rules.

3.4.1 Room Information of the Clients:

Unicron uses a graph structure in which nodes are rooms and edges are doors and openings between rooms. This is fairly static and is stored in a file called model.dat. The proxy reads this file and stores the room information in a hash table data structure so that searching for a room is fast.

To make intelligent decisions on forwarding the \move command the proxy server uses the location information of each client. The proxy reads the clients' messages, enabling it to get the location information of its clients without having to get them from the Server. These are the 2 types of messages the proxy reads to keep track of its clients' locations.

(1) \avatar uid, x, y, z, angle

When a client logs in, the central server sends an \avatar command to that client, that has the x, y, z coordinates of its location. The proxy then calculates which room contains that x, y, z in its area and places the avatar in that room.

(2) \nodechange new_room, old_room

When a client moves from one room to another it sends a \nodechange command to the server. This command help the proxy update the room information of the client as it moves from one room to another.

3.4.2 Visibility rules for the \move command

Visibility rules are those which tell the server, what clients in the CVE need to be informed when a client makes a move. The visibility rules for the proxy and central server are as follows,

(i) Send the \move to all other clients which are in the same room.
(ii) Send the \move to all clients which are in adjacent rooms.
(iii) Send the \move to all clients which are in adjacent rooms of the rooms in (ii)
(Adjacent rooms are the rooms which share an opening or a door)

3.5 Data Structures

3.5.1 Client

The proxy maintains information about each client using a struct data type. The elements in the structure are

```
struct client {
        int client_socket;
        int server_socket;
        char *client_name;
        char *current_room;
};
```

3.5.2 Room

The proxy maintains information about each room in a struct data type. The elements are
(1) x, y, z coordinates - This is the reference point for that room.
(2) width - width of that room.
(3) Length - Length of that room.
(4) Client sockets - Clients which are present in that room.
(5) Links - These are rooms which are adjacent to it.

```
struct room {
        double x;
        double y;
        double z;
        double width;
        double length;
        char *name;
        int *socks;
        char **links;
};
```

5.1.3 Hash Function

The room data structure is accessed constantly as the clients are moving. The proxy uses hashing techniques to make the search for rooms faster. It uses a simple hash function (because the number of rooms is not going to be very large) and uses linear probing technique for placing the keys (the room names) in an array. The average case searching time for a room is O(1).

The proxy server code is organized into 5 modules. They are

(1) Main module – Performs synchronous I/O multiplexing. Reads and writes to the sockets. (Figure 4)

(2) Process Message – Determines the type of message. (Figure 5)
(3) Process node change command – Parses the \nodechange command and updates the room information of the client. (Figure 6)
(4) Process avatar command – Parses the \avatar command and determines the client's room information. (Figure 7)
(5) Process move command - Determines who are the clients that need to be informed of this move. (Figure 8)

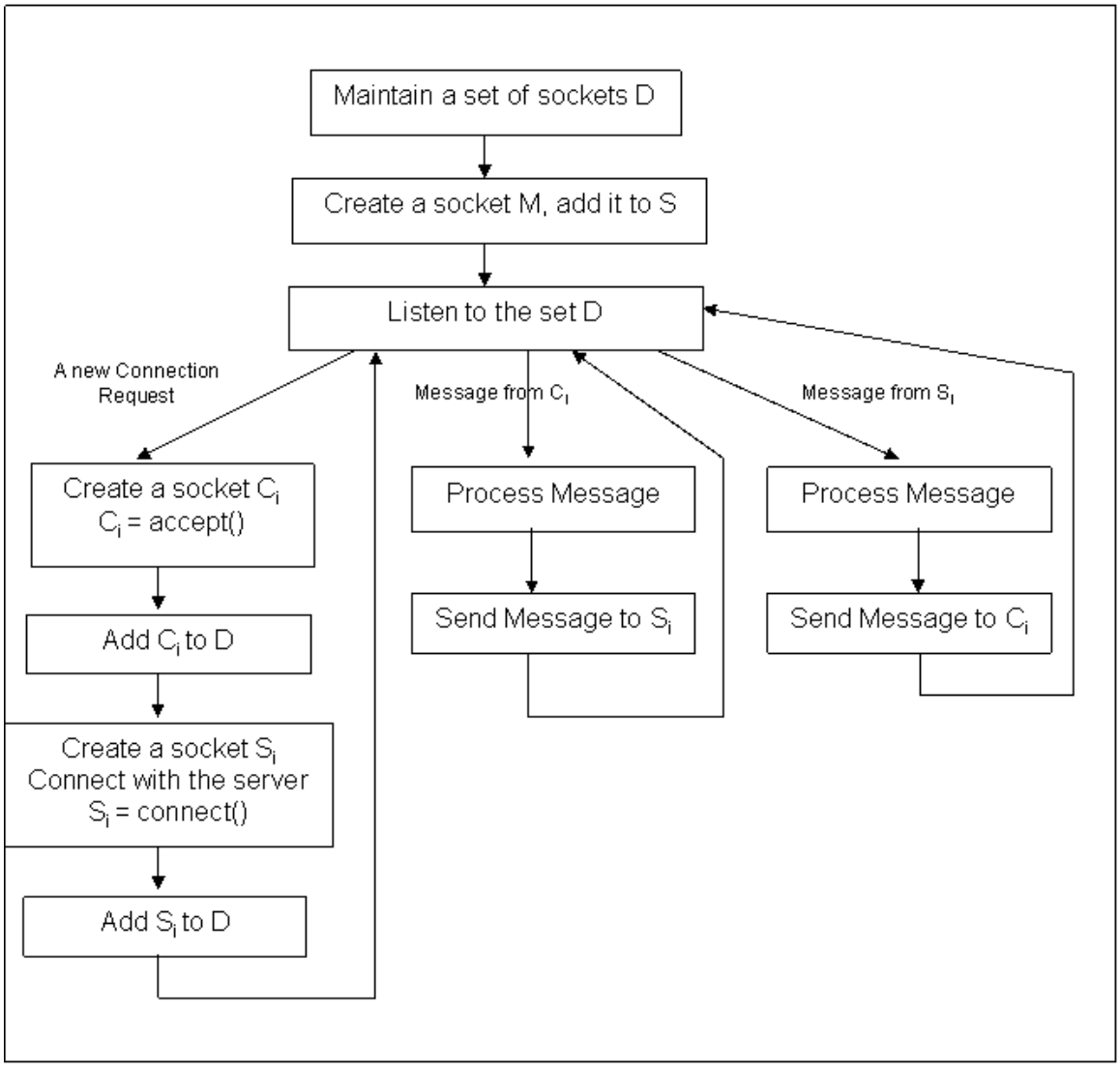

Figure 4: Main Module

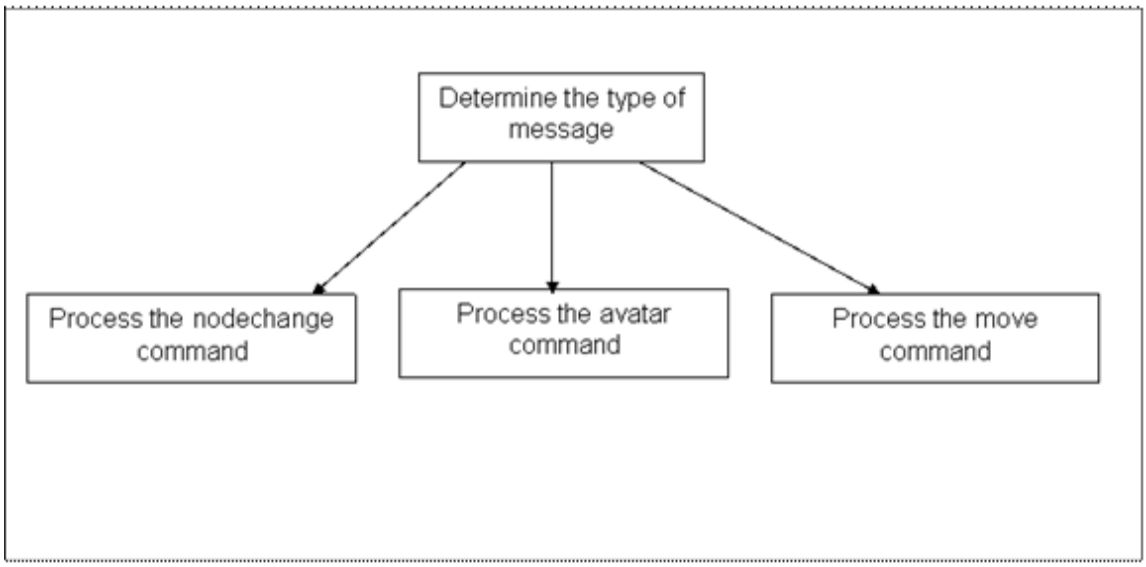

Figure 5: Process Message Module

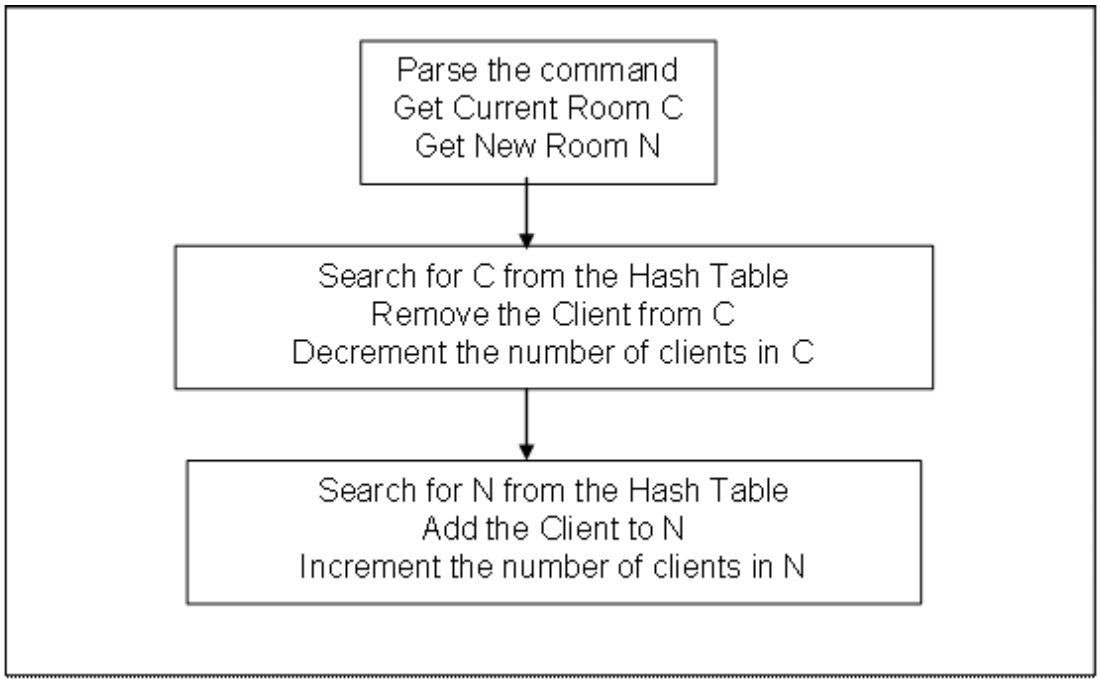

Figure 6: Process Node Change

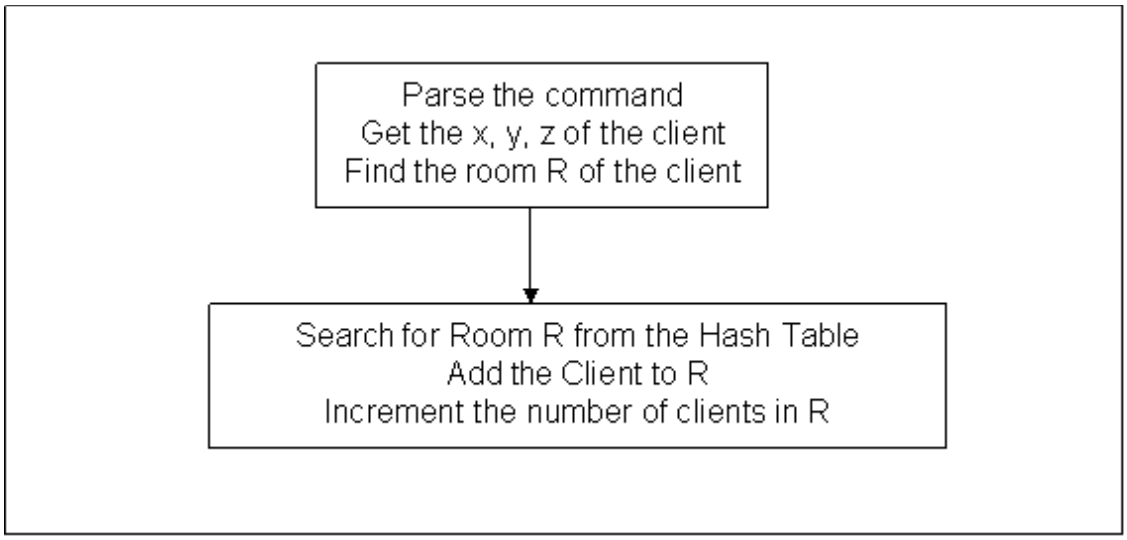

Figure 7: Process Avatar

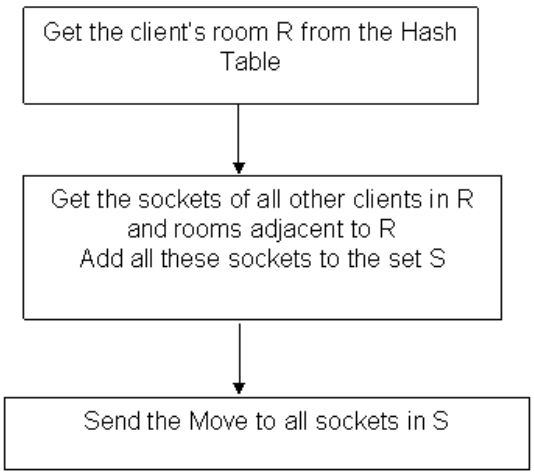

Figure 8: Process Move module

4.0 Performance Analysis of the proxy and central server

This section presents a set of measurements to evaluate whether the proxy server helps the central server increase the maximum number of clients it can handle. The first measurement is the maximum number of clients the Central Server is able to handle. Then the proxy servers are introduced into the system and the tests are rerun to find out

the maximum number of clients the whole system (Proxies + Central Server) is able to service. Testing with enough standard (human-operated) clients was not feasible for this project. In order to produce a test load, an artificial Test Client program was developed.

4.1.0 Test Clients

Test clients are artificial clients, which are used for testing and simulation purposes. The test clients log in and send move commands and nodechange commands, such that the avatar moves in a circular fashion continuously. Delays in between the moves simulate the real world client.

A realistic test client for this CVE would exhibit normal user behavior and incorporate an understanding of the layout of the science hall (the model which is used by the Unicron CVE). The test client avatars move continuously between the rooms. Consider these two lanes in science hall.

Lane1 – cooridoor-nw, cooridoor-167, cooridoor-ne (Figure 9)
Lane2 – cooridoor-se, cooridoor-125, cooridoor-main

The format of the move command is \move uid, body, x, y, z. To make the avatar move in a to-and-fro fashion in the Lane 1 the test clients vary (increment and decrement) the x values between 5 and 86.

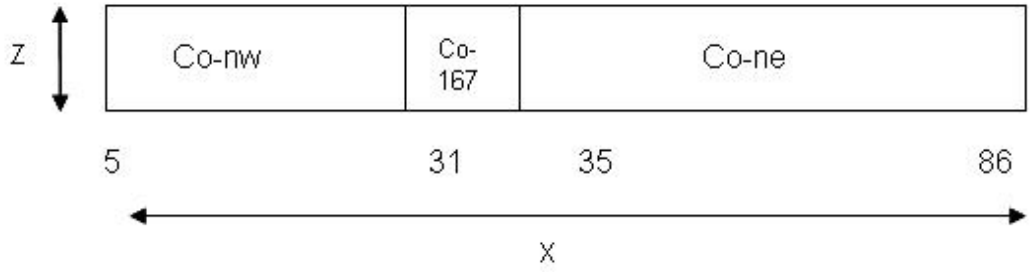

Figure 9: The layout of Lane 1

The algorithm for the test client program is simple.

(1) Establish connection with the server.
(2) Use fork() system call to create a new child process.
(3) Do the read() from the server in the child process.
(4) Do the write() in the parent process.
(5) Get the user name and password from console and send it to the server in the form of a \login command. The server will respond with positive acknowledgment.
(6) Let x = 5.2, y = 1.5, z = 5.0 and angle = 1.65.
(7) Send the move command to the server.

(8) Increment x by 0.2 and execute step 6. If x value goes greater than 86 go to step 8. Send nodechange commands when x values crosses 31 and 35. (The real server increments the x value by 0.15)
(9) Let x = 85.8, y = 1.5, z = 5.0 and angle = 1.65.
(10) Send the move command to the server.
(11) Decrement x by 0.2 and execute step 9. If x value becomes less than 5 go to step 7. Send nodechange commands if x value crosses 31 and 35.

4.1.1 Bandwidth Meter

Measurement of the bandwidth used by the Central Server and the Proxy, was collected using an open source software tool from sourceforge.net called Bandwidth Monitor NG, a simple, console-based bandwidth monitor for Linux and Solaris machines. This tool monitors the ethernet port and gives number of bytes received in Kb/s and number of bytes transmitted in Kb/s for every half a second.

4.2.0 Client Capacity of the CVE

The central server was intially able to handle only 4 clients (2 normal + 2 test clients). The same results were found even when the proxy was used. Both the two normal clients were freezing when a 4th client was added into the room. This was consistent with manual test observations (lockup at 4-5 users). The average CPU load on the machine running central server is 3 %, where as on the machine running a client the average CPU load was found to be 70%. The proxy was using less than 1 % of the CPU and it did not show up when top command was used.

4.2.1 Bandwidth measurements

Without a proxy, the average bandwidth used by the central server for 8 clients is 41 Kb/s (bandwidth used is the sum of number of bytes received per second and number of bytes transmitted per second). Although the clients froze up when the load was just 4, the central server worked fine. But when all the 8 clients are connecting through the proxy, the average bandwidth used by the Central Server was 32 kb/s. Since the proxy processes all the moves, the load at the central server was reduced.

If a number of moving clients in adjacent rooms share a proxy, it helps the central server by reducing the bandwidth utilized. This means the traffic over the wide area network will be reduced if the proxy and the clients are on a remote LAN.

4.2.2 Inferences

From the test results and during related testing, some major drawbacks in the client and central server were found and fixed. When CVE load increased, the clients were the

limiting bottleneck. The central server and proxy server were using only 1 to 3 percent of the total CPU, where as the client was using 70% (X + iconx) of the CPU on the average. Here, the X server renders the OpenGL graphics on the screen. Iconx virtual machine is an imaginary computer that executes instructions for Unicon programs[7]. The client's graphics code was failing to handle the simultaneous motion of a number of avatars when they were in the same room. This led to the discovery of major inefficiencies in the client code. The server handled the multicast of the \move messages inefficiently which further worsened the speed of the client. A moving client aggregates 3 move commands in a single TCP packet and writes to the server to be multicast. This is a good strategy. But the central server was sending out each \move command in its own TCP packet.

4.3 Test Results after the faults were corrected

The test results above helped to identify the reasons for the poor scalability of the CVE. As an immediate step, the client and server codes were altered to correct the above mentioned shortcomings. As expected, the modified CVE showed improved performance. The test results enabled a major enhancement in the Unicron CVE, which is evident from the fact that the CVE, which was handling 4 clients before is now able to handle 26 simultaneously moving clients. More detailed explanation of the test results for the new CVE is given in the following sections.

4.3.1 Number of clients handled by the Central Server and Proxy

This test is to see how much load the central server is able to handle with and without proxy. In both the circumstances central server is able to handle 26 clients. This test involved

(1) 24 test clients distributed over 5 different machines
(2) 2 normal clients distributed over 2 machines
(3) One proxy and a Central Server

Distribution of the clients in the CVE is 13 per lane. The CVE handles nearly 20 clients in a single lane (It varies between 18 to 24).

When the load is 26, the average CPU used by the central server is 3%, proxy server is less than 1%. The clients are using 60% to 70% of the CPU at peak load. So this again suggests that the clients are the bottlleneck during peak loads. Because of this reason the tests do not show exactly how well the proxy helps in improving the scalability. A maximum of 40 simultaneous clients were put to test, both the proxy and the central server handled them comfortably. To produce a load on the central server to make it fail, huge numbers of test clients would be needed (to serve 26 clients the central server used just 3% of the CPU). But the test results explained in the next section proves that the proxy will improve scalability.

The graphics rendering speed of the test machines were 300 to 500 frames/second. This is on the lower side and this also suggests the use of hardware graphics accelerators to boost the performance of the CVE in terms of speed and load capability.

4.3.2 Bandwidth used by Central Server

This section shows how much bandwidth the central server is using with and without proxy. The measurements were obtained when the CVE is handling its peak load. All 26 clients used for this testing are in same room. This kind of scenario is common in CVE, for example during lecture sessions we can expect more than 20 avatars assembling in a single room.

Without a proxy, the average bandwidth used by the Central Server at peak load (26 clients) was 148 Kb/s. The average was taken over 1020 readings for every half a second. But when the proxy is used and all 26 clients are using the proxy, average bandwidth used by the central server at peak load (26 clients) was found to be 82 kb/s. The average was taken over 1020 readings for every half a second. The average bandwidth used by the proxy was 94 Kb/s.

These results indicate that the proxy considerably reduces the load at the central server. It saves about 45% of the bandwidth used over the wide area network. This is due to the fact that all 26 clients are connecting through the proxy and proxy handles all of the move traffic which is about 70% of the overall traffic. The amount of bandwidth saved increases as we group more number of clients.

5.0 Conclusion and future work

The proxy server for the Unicron CVE is successfully developed and tested. Modifications to the central server were done to make it proxy aware. Test clients proved to be a helpful tool in assessing the load handling capability of the servers. The test results helped to identify the shortcomings of the CVE and improve its performance. The proxy reduces the bandwidth used by the central server up to 45%, this in turn will help the central server to handle more load provided the client bottleneck is resolved. The tests were not conducted by deploying the proxy in remote campus. This might provide more accurate figures. Nevertheless it will not affect the current results and claims. The proxy server can be extended to multicast video and audio messages in future. Future work includes making minor changes to the \avatar command format to reduce the computations at the proxy. The testing results doesn't show the upper bound on the server load handling capability. This requires deployment of a large number of test clients and is a part of future work.

References

[1] Jeffery, C. (2005). *Unicron – A Virtual Computer Science Community*. Retrieved May 02, 2005, from New Mexico State University, Department of Computer Science Web Site: http://www.cs.nmsu.edu/~jeffery/vcsc/

[2] E. Frécon and M. Stenius, *"DIVE: A Scaleable network architecture for distributed virtual environments", Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments), Vol. 5, No. 3, Sept. 1998, pp. 91-100.*

[3] H. Fahmi, M. Latif, S. Sedigh-Ali, A. Ghafoor, P. Liu, L. Hsu, *"Proxy Servers for Scalable Interactive Video Support", IEEE journal, September 2001, pp. 54-60.*

[4] P. Danzig, R. Hall, and M. Schwartz. A case for caching file objects inside internetworks. In Proceedings of the ACM SIGCOMM '93 Conference, Sept. 1993. http://citeseer.ist.psu.edu/danzig93case.html .

[5] A. Luotonen and K. Altis, "*World-Wide Web Proxies*", Computer Networks and ISDN systems, Vol 27, 1994.

[6] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrell. *A Hierarchical Internet Object Cache*. In Proceedings of the 1996 USENIX Technical Conference, Jan. 1996.

[7] Griswold, R. Jeffery, C. Townsend, G. *"Graphics Programming in Icon", Peer-to-Peer Communications, 1998, p. 479.*