Design and Implementation a Server Receiving Data in Both Forms TCP and UDP Through the Same Port and its Impact on the Network Performance

Husam Ali Abdulmohsin
Science College, University of Baghdad /Baghdad.
husamex@yahoo.com

Received on:25/12/2014 & Accepted on:20/1/2016

ABSTRACT

Internet is the largest network that transfers a huge amount of information through the web and that requires data transfer between many network bottlenecks, devices and different hardware technologies. This data movement requires data transfer between many application, software's and operating systems. Many theses and researches were published in the topic of solving the issue of extreme data transfer rate; this issue causes time consuming problems. There are many technologies of transferring data across the internet; two of the major data transfer technologies are the User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). Those different data transfer technologies through the internet cause most of the servers much of their time translating the data from one technology to another because most of the servers deal with one data transfer technology and therefore it has to translate all other technologies to the technology that it deals with. This research establishes a Server named TCP-UDP Server (TUS), to receive data from both UDP and TCP nodes through the same path without the need of changing the network entities or protocols connected to the server and to avoid the need for transferring the data from the (UDP) form to the (TCP) form and vise versa. All the operations performed by the server are accomplished without any hardware intrusion to avoid time consuming. The TUS server as many servers support the multithreading technology to serve a large amount of nodes at the same time. Each node has its own thread to deal with. This thread has its own life time determined by many facts, and that in turn decides to terminate the thread or not.

Keywords*:* Multithreading technology, TLT (Thread Life Time), NC (Node Connectivity), LAN (Local Area Network), TUS (TCP-UDP Server), UDP, TCP.

INTRODUCTION

The web is a massive storage of information, and accessing this information through the internet has been a major problem discussed in many researches and theses. One of those problems is the different types of data transfer technologies used by the clients. If you are behind a router you must forward the port's UDP/TCP traffic from your router to the machine or server [1]. This can give the opportunity to others to retrieve any data through the internet regardless of the data transfer technology they're using. Information can be retrieved easily with the right tools and methods. Such problems appear in many situations such as service providing servers and game servers that deal with many kinds of data transfer technologies and sometimes need to transfer the data received from any technology to the other.

The TUS based Multithreading technology proposed in this research operates in the application layer as a standalone server application. So the type of connection used in this work is called the

https://doi.org/10.30684/etj.34.2B.15
2412-0758/University of Technology-Iraq, Baghdad, Iraq

open layer interconnection, were each OSI layer can interact directly with the equivalent layer in the other node [2].
The TUS solves the problem of receiving data from different technologies through the same port and path.

Methodology

JAVA has been used to reduce the complexity of setup, through creating a low- level code to deal with any kind of operating system and hardware, to enhance the portability, flexibility and scalability of TUS.

The performance and interaction between different types of operating systems and different versions of the same type, like WindowsXP and windows7, will be examined through this research. The TUS server uses the same port to recieve all requests from both types of connections, UDP or TCP, regardless of the node's operating system type or version. To be mentioned, multiple machines connected to the same router will need to choose different ports for each machine to avoid conflicts, but the TUS server solved that problem, both machines requests are recieved on the same port, using the same port will reduce the port monitoring time.
The general idea of the project is illustrated in figure 1, the TUS can deal with different nodes that either use TCP or UDP data transfer, and the server deals with both using the same port, so the listener of the server can monitor both the UDP and the TCP on the same port.

A runtime error handler is added to the server to avoid any jam in the system. The server is testbeded on a Local Area Network (LAN), with different versions of the windows and open source operating systems, on different hard specification. Tools designed in this research such as Thread Life Time (TLT) and tools used such as Jperf to measure the bandwidth of the server connection, and wireshark (node connectivity) will be used to verify and measure the connectivity and performance of the TUS server. Relationship and interoperability between each peer and server will be discussed.

At the same time privacy for the user in the LAN can be introduced with a minimum specification. Local Area Network (LAN) which consists of a specific subnet will be designed to support client to server model for TUS and this will also introduce a trusted domain and double layer privacy[3].
Skills in network and operating systems will help to reduce the troubleshooting occurring when the TUS and Client connection is established. The establishment of the testbed will involve the installation of the TUS Server, routing and the security firewall.

The routing has to be established previously in the server and the TCP and UDP port is established at the client side. Because the command to open a specific port is temporary, routing and firewall configuration will need to be set after each booting to avoid a disconnection error. The cause of any error cannot be predicted but this has been handled by an error handler that is supported by the JAVA language but one has to determine the error expected at that part of the program code, for example, if you are writing a code to create and connect a port, you have to tell the error handler to look for a port creation or connection error as shown in appendix A (line 6,16,45,52 and 76). A start-up script has to be developed to capture both types of connection the TCP and UDP as shown in Appendix A (line 5 and 17). Different operating systems have different styles. In window operating system, the TUS doesn't have to activate in system administrator mode, because all the changes will be permitted in the configuration file. The TUS didn't have to take care of stacking TCP/UDP processes; because the operating system takes care of that at the application level, mentioning that the multithreading technology added to the server solve the stack overflow problem and most of the time there are few tasks in the stack and if an overflow occurred it can be solved by adding stack overflow Exception to the error handler. A lot of problems appeared because of lack of understanding routing and firewall. If the IP is using DHCP, which can also cause a problem because the remote Client address is

changing. The changing of IP will make the connection fail because it cannot find the remote client to validate and authenticate the Identity of the client. It is wise to change the IP to static IP for the client and server. Wireshark and ping will be used to debug the connection [4]. If one wants to use the TCPDUMP version 4.5.1, he must install PSSDK protocol driver and must use Windows 7, Windows 8 and Windows Server 2012 [5]. The measurement of bandwidth will be captured through Jperf 2.0.0 version [6]. This bandwidth will provide the performance and relationship to various operating systems.

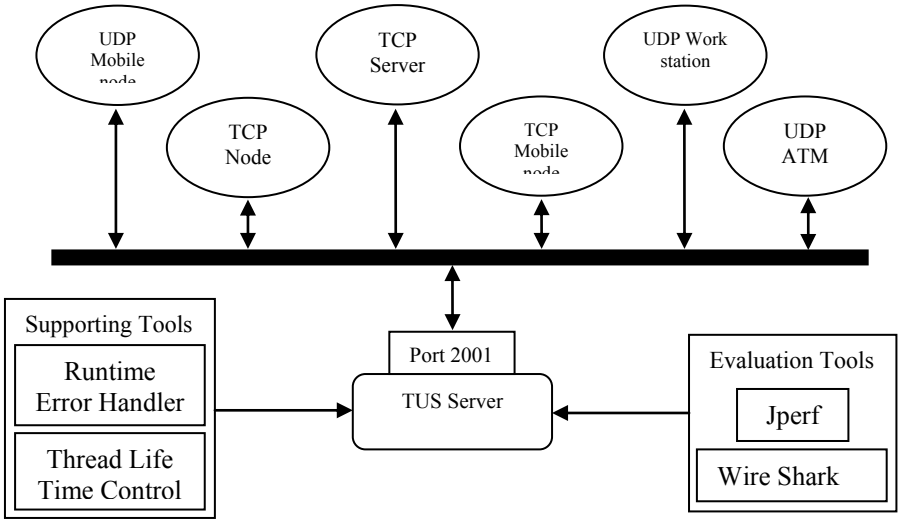


Figure (1). Module Flowchart that Illustrate a TUS server acting with different types of clients through the same connection.

Related Work

NETCAT, THE "SWISS ARMY KNIFE" Netcat is a facility that enables the server to receive and send data through TCP and UDP Connection network technologies. When you are responsible of monitoring a network, managing a network or securing a network, it is very important that you understand the facilities the Netcat provides. There are many uses for the Netcat, like a port redirector, listener or scanner and many other uses in the network world. A simple example of scanning a port using a Netcat is "**c:\tools>nc -v -w 2 -z target 20-30**", this instruction will scan all the ports in the interval (20 -30) at the targeted node, and in each port test, the Netcat will inform about the status of the Telnet, FTP and mailer server as shown in Figure 2. The Netcat can be used to see what ports the target is listening to, by the –z instruction, this instruction will refuse sending any data through a TCP connection and very limited probe through a UDP connection. If the listener wants to slow down the scanning speed he can use the –i instruction [7].

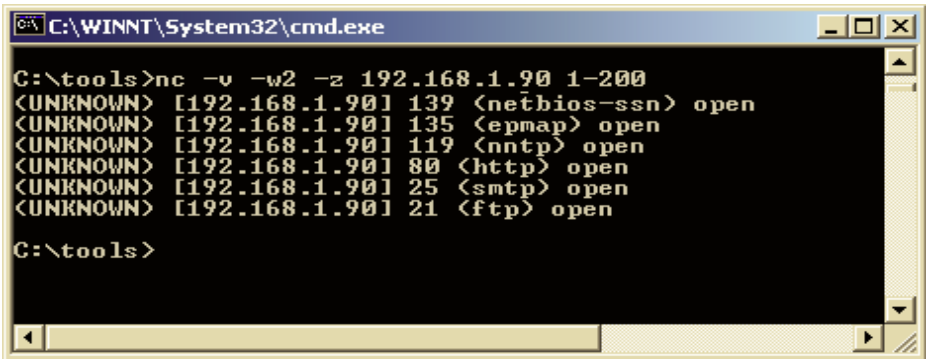


Figure (2). Netcat scanning example.

NETWORK ATTACHED STORAGE (NAS),

During the NAS1 project, it was decided to configure it without the need to configure the network configuration and setting, but it was noticed that they need to use the DHCP as a perfect solution. Therefore we looked for a network that didn't use the DHCP in is structure. Here we think about using the UDP communication technology to gain the ability to send a message to all nodes in the network without the need to know the IP address of all nodes. The main idea of this project was to have the ability to use administrative interfaces via SSH, Telnet, HTTP and HTTPS. At this point the necessity for an iproxy was needed and the need for the tunnelling TCP over UDP was essential. All iproxy infrastructures are constructed from two proxy servers, the iproxy-client and the iproxy server. Such a proxy infrastructure converts the TCP data stream to UDP datagram and vice versa. The iproxy-client at the user's node listens for the incoming TCP streams that might be received from a web server or any node sending a TCP stream. At this point the iproxy-client duty is to encapsulate the TCP stream into a UDP datagram. The iproxy-server at this point is running at the NAS and listening for the UDP datagrams sent from the iproxy-client. But now it is different, there is no need for the iproxy-client to encapsulate the TCP in to UDP, the ability to receive TCP and UDP through two different paths has been added to the iproxy-server and can deal with both at the same time.

EXPERIMENTAL

The experiment test-bed LAN is based on client-server model which consist of two java server, one supports window 8 and the other supports an open source operating system and the clients support two versions of windows operating system, the Vista and Windows 2007 version. The test bed LAN which consists of 20 clients, is using the same specification as shown in table 1, and of the LAN is connected through a Hub.

Table (1). Hardware and operating system specification

Specification	Server	Server	Client	Client1
Operating System	Windows 8	Open source	Vista	Windows 2007
Memory	1GB	1GB	2GB	4GB
Processing	64-bit	64-bit	32-bit	32-bit
Hard disk	500GB	500GB	1T	250GB
Software	TUS server	TUS server	TCP socket	UDP socket
	Multithread creator	Multithread creator	Open socket and port	Open socket and port

Notice that the memory is different from one machine to another and all of them functioned perfectly without any system jam. Even the server, we didn't need very high memory storage because of the thread life termination determined by the TUS as shown in Appendix A (line 38 and 67). The TUS will be installed on the server, and the last in turn will open a TCP socket

using the SERVERSOCKET command, see Appendix A (line 5), and will open a datagram socket using the command DATAGRAMSOCKET [9]. During the authorization process a certificate authority name and key will be generated. This key is only needed by the server for authentication and signing-in if needed.

Once the installation of the TUS is completed a thread will be started by using this command as shown in Appendix A (line 10 and 15).

In the server a command as shown in Appendix A (line 5 and 59) will be executed to setup a port (2001) in the server that will be used to receive any data from any client [10][11]. Both types of data transfer, the TCP and UDP, will have a direct connection with the server on the same port, by creating sockets for both types of data transfer technologies (TCP and UDP) that will create a tunnel with the server through port (2001).

Once the server have fully completed, then both clients such as TCP and UDP will be activated. At this point the clients can start sending their requests, for example, a client sending a Standard Query Language (SQL) command through the browser as shown in figure 3, but at this case the server has to set on port (8080) to receive all web requests. As soon as the server receives the request a thread will be created for that certain client.

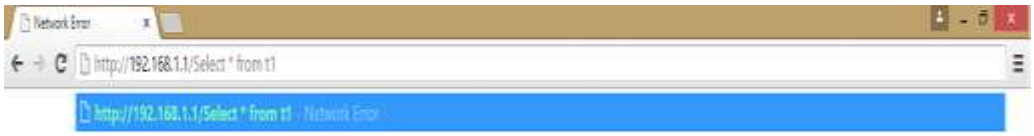


Figure (3). Client request

EVALUATION

Debugging tool called jperf will be used to monitor the connection performance and analyse the results. The results received will help to evaluate the relationship of TUS with operating system and with the TCP and UDP clients [12]. This experiment uses different specifications with a minimum memory is 1GB whereas the maximum memory is 4GB. The bandwidth result will help to determine that if the memory has low impact to the bandwidth compared to the type of operating system. The type of the operating system is the main criteria which influence the performance of the bandwidth in the LAN.

This evaluation is based on comparing the data transfer rate and bandwidth between the TUS server designed in this work and an ordinary system that receives TCP and UDP requests each on a separate server then directed to the right position.

The connection between client and server will be through the port 2001 regardless of the communication protocol the client is using, the connection is successful once it communicates as shown in Table 2. As you can see in table 2, that no reject nor ignore status occurred during the connection and that's because we are implementing in a local area network (LAN), but if such a case occurred, the client will go back to the waiting queue until the connection is achieved, because the connection request is in an infinite loop and in case of some reason that caused the request to go out of the queue, the request will go back into the queue and wait for a connection. A failure status will appear only if the server was not functioning or the client IP has changed as shown in Table 2. A reject status will appear only if the client was not authorized to connect to the server.

Table (2). Connection Status

Client	Time	Source	Destination	Protocol	Port	Status	Description
C1	20:18:01	http://192.168.1.2	http://192.168.1.1	TCP	2001	Succesful	
C1	20:18:22	http://192.168.1.7	http://192.168.1.1	TCP	2001	Failure	IP client changed
C1	20:19:47	http://192.168.1.4	http://192.168.1.1	TCP	2001	Failure	Server Turned off
C1	20:20:14	http://192.168.1.4	http://192.168.1.1	TCP	2001	Succesful	

C2	20:19:02	http://192.168.1.71	http://192.168.1.1	UDP	2001	Reject	IP Not Authorised
C2	20:19:57	http://192.168.1.3	http://192.168.1.1	UDP	2001	Succeful	

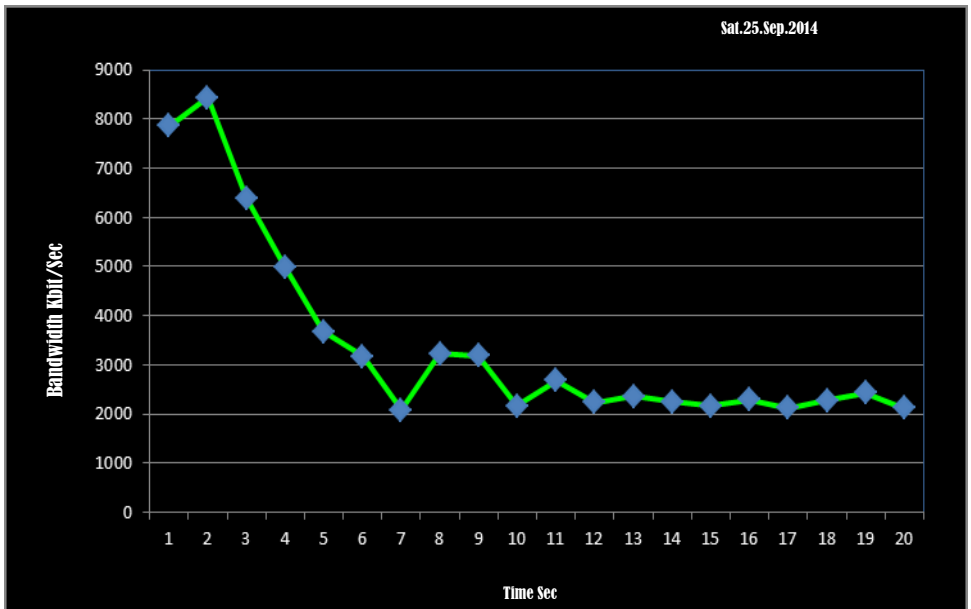
If the connection happens in one way, then the routing or firewall has not been configured and opened yet, here the error handler is involved to avoid any jam in the system, so it will terminate the connection and send the request back to the waiting queue [10][11].

The jperf was set to capture the bandwidth and the data transfer rate in 21 Seconds limit of time. Table 3 show the bandwidth of the standard Server decrease and the data transfer show that the standard server bandwidth reduces based on the time taken. The maximum data transfer is 8.23Mbytes with a maximum bandwidth 7.86Mbyte/sec but the overall data transfer is 66.53Mbytes and the average of the bandwidth is 3.39Mbyte/sec.

Table 3. Standard Client Server Data Transfer using windows 8 operating system

Item	Interval	Transfer	Bandwidth
1	0.0- 1.0 sec	7.68 Mbytes	7.86 Mbyte/sec
2	1.0- 2.0 sec	8.23 Mbytes	7.85 Mbyte/sec
3	2.0- 3.0 sec	6.24 Mbytes	6.91 Mbyte/sec
4	3.0- 4.0 sec	4.88 Mbytes	5.86 Mbyte/sec
5	4.0- 5.0 sec	3.59 Mbytes	3.91 Mbyte/sec
6	5.0- 6.0 sec	3.10 Mbytes	3.05 Mbyte/sec
7	6.0- 7.0 sec	2.04 Mbytes	2.21 Mbyte/sec
8	7.0- 8.0 sec	3.16 Mbytes	2.99 Mbyte/sec
9	8.0- 9.0 sec	3.11 Mbytes	3.18 Mbyte/sec
10	9.0- 10.0 sec	2.12 Mbytes	2.18 Mbyte/sec
11	10.0- 11.0 sec	2.62 Mbytes	2.27 Mbyte/sec
12	11.0- 12.0 sec	2.18 Mbytes	2.17 Mbyte/sec
13	12.0- 13.0 sec	2.31 Mbytes	2.12 Mbyte/sec
14	13.0- 14.0 sec	2.20 Mbytes	2.11 Mbyte/sec
15	14.0- 15.0 sec	2.12 Mbytes	1.98 Mbyte/sec
16	15.0- 16.0 sec	2.23 Mbytes	2.30 Mbyte/sec
17	16.0- 17.0 sec	2.08 Mbytes	2.23 Mbyte/sec
18	17.0- 18.0 sec	2.22 Mbytes	2.34 Mbyte/sec
19	18.0- 19.0 sec	2.37 Mbytes	2.15 Mbyte/sec
20	19.0- 20.0 sec	2.07 Mbytes	2.12 Mbyte/sec
21	0.0- 20.0 sec	66.53 Mbytes	3.39 Mbyte/sec

The data transfer will drop when time passes by because of the increment of clients as shown in figure 4. One can also notice that the data transferred is not stable in the time interval 5-20 sec, and that explains the time taken to transfer requests from UDP to TCP and vice versa and the time required to receive each type of data transfer on a separate port. The graph shows that the bandwidth is fluctuating.



Figure(4). Bandwidth of the iproxy server using windows 8 operating system

TUS server performs with higher bandwidth when using the direct connection to the UDP or TCP clients, mentioning that the operating system used can change the results but still the difference is not very big, for example, the Data transfer in an open source operating system is higher than using windows 8 [12]. The maximum data transfer the TUS server is using is 11.9Mbytes as shown in table4. From figure 5, we can notice that the bandwidth of the TUS is stable and all the nodes are served in parallel.

Table (4). Client TUS Server Data Transfer using windows 8 operating system

Item	Interval	Transfer	Bandwidth
1	0.0- 1.0 sec	11.9MBytes	12.45Mbyte/sec
2	1.0- 2.0 sec	11.8 MBytes	11.81Mbyte/sec
3	2.0- 3.0 sec	11.6 MBytes	11.80Mbyte/sec
4	3.0- 4.0 sec	11.5 MBytes	11.79Mbyte/sec
5	4.0- 5.0 sec	11.3 MBytes	11.67Mbyte/sec
6	5.0- 6.0 sec	11.2 MBytes	11.69Mbyte/sec
7	6.0- 7.0 sec	11.1 Mbytes	11.70Mbyte/sec
8	7.0- 8.0 sec	11.1 Mbytes	11.81Mbyte/sec
9	8.0- 9.0 sec	11.2Mbytes	11.81Mbyte/sec
10	9.0- 10.0 sec	11.2 Mbytes	11.80Mbyte/sec
11	10.0- 11.0 sec	11.2 MBytes	11.78Mbyte/sec
12	11.0- 12.0 sec	11.2 MBytes	11.79Mbyte/sec
13	12.0- 13.0 sec	11.2 MBytes	11.79Mbyte/sec
14	13.0- 14.0 sec	11.1 MBytes	11.84Mbyte/sec
15	14.0- 15.0 sec	11.2 MBytes	11.79Mbyte/sec
16	15.0- 16.0 sec	11.1 Mbytes	11.82Mbyte/sec
17	16.0- 17.0 sec	11.1 Mbytes	11.76Mbyte/sec
18	17.0- 18.0 sec	11.2Mbytes	11.81Mbyte/sec
19	18.0- 19.0 sec	11.2 Mbytes	11.81Mbyte/sec
20	19.0- 20.0 sec	11.2 Mbytes	11.80Mbyte/sec
21	0.0- 10.0 sec	113 Mbytes	11.8Mbyte/sec

In some cases, the server receives the data encrypted for security reasons, and the server has to decrypt the data to manipulate it as required and in some cases the server has to check the authorization of the user, like checking the user name and password, and sometimes the password is generated from two or more functions and all of that needs more time to check [13], or recognizing a fingerprint using gabor filter [14]. So encryption and decryption or authentication process will involve the CPU, that's why it depends on the hardware specification to process the data. The performance of encryption and decryption will not be evaluated in this experiment.

The higher bandwidth shows the perfect performance of the TUS because the TUS does not interfere with the session layer and many communication transfer protocols [2].

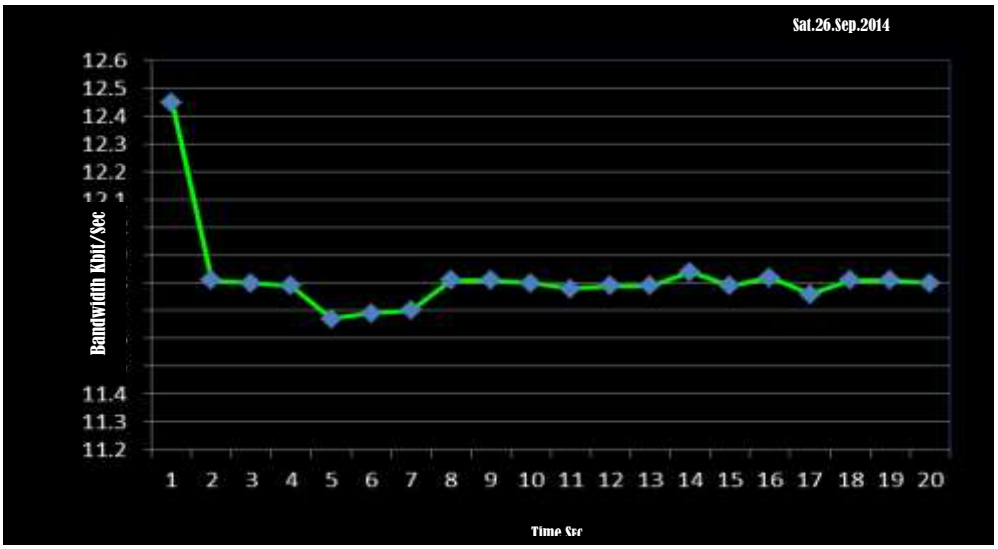


Figure (5). Bandwidth of the TUS server using windows 8 operating system

When time increases the value of bandwidth is reduced between 0-2 seconds and the bandwidth becomes fluctuate until the interval time between 8-20 second becomes stable because of the multithreading technology added to the TUS and receiving requests on the same port, as shown in Figure (5).

In order to do a comparison study between the designed server and the iproxy server usually used, we must calculate the average bandwidth of both servers, as shown in Table (5). The results show that bandwidth is always higher when the TUS server is used as a server receiving the clients request and to transfer the data. These results prove two things, first the performance of the TUS is faster than the standard server, and second, whether or not the specification of the hardware is higher such as CPU, memory or hard disk, it is not a key indicator of the performance of bandwidth and data transfer.

Table(5). Overall data at server with windows been captured first

Item	Server type	Interval	Data Transfer	Bandwidth
1	Standard server	0.0- 20.0.sec	66.53 MBytes	3.39 Mbyte/sec
2	TUS server	0.0-20.0 sec	113 MBytes	11.8 Mbyte/sec

CONCLUSION

The time needed to transfer the data received from one form to another at the server area has been indispensable because the TUS server receives both types of data (TCP and UDP) both through the same path. The TUS operates at the application level of the network to avoid time

consuming. The server is applicable to many kinds of operating system and hardware, and that was gained by using the JAVA programing language that supplies the low level language that provides the server with a compatible and interoperability with all kinds of software and hard testbed. Each node in the network can obtain its own connection with the TUS server without any delay or queue because of the multithreading technology added to the TUS server.

The time required to transfer the data from one form to another at the node is avoided, no application needed for the data form transfer, now the node doesn't need any time to do the data transformation because the TUS server will receive the data regardless of its form. One port can be used to do all the data talking between all the nodes and the server, and that will minimize the port monitoring time required. Although the memory of the windows operating system is high but the TUS server took the memory utility under consideration by terminating some the threads needed to achieve server and client relationship are under some conditions, such as, long time ideal, send a quit message, the connection caused an input/output problem if not handled by the run time catch error at the server which is very rare.

A security level can be added to the Server according to the organization need and security level. Such as encrypting the data at both sides of the connection, including a key needed for the security or adding a user name and password as an authentication user level.

A redirecting technology can be added to the server, to transfer the request of a client to another server or another client, to solve the request, at this point transferring data from TCP to UDP and vice versa is needed.

A supporting server can be added to the system, that can support the TUS server in case of some over load at the peak time, by creating threads that direct the request of the client to the supporting server, and sending the request results from the supporting server to the client directly.

With different specifications of hardware in the test-bed of this experiment, we distinguished that different hardware doesn't influence the bandwidth that was captured in the results.

REFERENCES

[1] Calvert K. L., Donahoo M. J., TCP/IP Sockets In Java, Morgan Kaufmann Publishers, 2008.

[2] Noergaard T., Embedded Systems Architecture, Elsevier Inc., 2013.

[3] Hjorth T. S., Thorbensen R., *Trusted Domain: A security Platform for home Automation 2010, Compute, 2010.*

[4] Chappell L., Troubleshooting With Wireshark: Locate The Source of The Performance Problems, James Aragon, 2014.

[5] Acton Q. A., Issues in Applied Computing, Scholarly Editions, 2013.

[6] Charlie H., Binu J., Java Performance, prentice hall, 2011.

[7] Kauclirz Jr. J., Brian B., Dan C., Michael S. J., Eric S. S., Wihelm T., NetCat Power Tools, Syngress Publishing Inc., 2008.

[8] Darve G., Disaster Recovery, Course Technology, 2011.

[9] Rusty H. E., Java Network Programming, O'Reilly Media Inc., 2013.

[10] Jan G., An Introduction To Network Programming With Java, Springer Science and Business Media, 2013.

[11] Doug L., Java All-In-One For Dummies, John Wiley and Sons Inc., 2011.

[12] Lee R., Barry W., Networked: The New Social Operating System, Massachusetts InstiTUSe of Tecgnology, 2011.

[13] Hadi H., Two Factor Authentication Based Generated One Time Password, Eng. & Tech. Journal. Vol.33. Part (B), No.3.2015.

[14] Ekbal H. A., Hussam A. A., Hanady A. J., Fingerprint Recognition Using Gabor Filter with Neural Network, Eng. & Tech. Journal. Vol.32. Part (A), No.2.2015.

Appendix A

```
1 public class EchoServer {
2 ServerSocket m_ServerSocket;
3 public EchoServer() {
4   try
5       { m_ServerSocket = new ServerSocket(2001);
6       } catch(IOException ioe) {}
7       System.out.println("Listening for clients on 2001...");
8       int id = 0;
9       ClientServiceThread1 cliThread1 = new ClientServiceThread1();
10        cliThread1.start();
11   while(true)
12        { try
13          {   Socket clientSocket = m_ServerSocket.accept();
14             ClientServiceThread cliThread = new          ClientServiceThread(clientSocket, id++);
15              cliThread.start();
16          }catch(IOException ioe) {}
17       } //while
18  }
19  public static void main(String[] args) {new EchoServer();}
20 class ClientServiceThread extends Thread
21    {   Socket m_clientSocket;
22       int m_clientID = -1;
23       boolean m_bRunThread = true;
24       ClientServiceThread(Socket s, int clientID)
25       {   m_clientSocket = s;
26          m_clientID = clientID;}
27       public void run()
28       {   DataInputStream in = null;
29          DataOutputStream out = null;
30          System.out.println("Accepted Client : ID - " + m_clientID + " : Address - " +
31                        m_clientSocket.getInetAddress().getHostName());
32          try
33          {   in = new DataInputStream(m_clientSocket.getInputStream());
34             out = new DataOutputStream(m_clientSocket.getOutputStream());
35             while(m_bRunThread)
36             { String clientCommand = in.readUTF();
37                if(clientCommand.equalsIgnoreCase("quit"))
38                {m_bRunThread = false;
39                 System.out.print("Stopping client thread for client : " + m_clientID);}
40                else
41                {System.out.println("Client Says :" + clientCommand);
42                 out.writeUTF(clientCommand);
43                 out.flush();}} // while
44          } //try
45          catch(IOException e){e.printStackTrace();}
```

```
46          finally
47          {try
48             {  in.close();
49                out.close();
50                m_clientSocket.close();
51                System.out.println("...Stopped");}
52             catch(IOException ioe){ioe.printStackTrace();}
53 }}}
54   class ClientServiceThread1 extends Thread
55   {   boolean m_bRunThread = true;
56       ClientServiceThread1(){}

57       public void run()
58       {   try
59         {   DatagramSocket serverSocket = new DatagramSocket(2001);
60         while(true)
61            { byte[] receiveData = new byte[1024];
62              byte[] sendData = new byte[1024];
63              DatagramPacket receivePacket = new
DatagramPacket(receiveData, receiveData.length);
64              serverSocket.receive(receivePacket);
65              String sentence = new String( receivePacket.getData()).trim();
66                 if(sentence.equalsIgnoreCase("quit"))
67                 {m_bRunThread = false;
68                   System.out.print("Stopping client thread for client : " +
m_clientID);}
69              System.out.println("RECEIVED: " + sentence);
70              InetAddress IPAddress = receivePacket.getAddress();
71              int port = receivePacket.getPort();
72              String capitalizedSentence = sentence.toUpperCase();
73              sendData = capitalizedSentence.getBytes();
74              DatagramPacket sendPacket =
75              new DatagramPacket(sendData, sendData.length, IPAddress,
port);
76              serverSocket.send(sendPacket);
```