ENHANCEMENT OF TSUNAMI UDP PROTOCOL

Akash Panda (11UCS004)
Anand Solanki (11UCS008)
Ankur Pratap (11UCS010)
Shubham Upadhyaya (11UCS057)
Supriya Sarkar (11UCS063)

COMPUTER SCIENCE & ENGINEERING DEPARTMENT
NATIONAL INSTITUTE OF TECHNOLOGY,AGARTALA
INDIA-799046
March 2015

ENHANCEMENT OF TSUNAMI UDP PROTOCOL

Report submitted to
National Institute of Technology, Agartala
for the award of the degree
of
Bachelor of Technology

by

Akash Panda (11UCS004)
Anand Solanki (11UCS008)
Ankur Pratap (11UCS010)
Shubham Upadhyaya (11UCS057)
Supriya Sarkar (11UCS063)

Under the Guidance of

Mr. Anupam Jamatia
Assistant Professor, CSE Department, NIT Agartala, India

COMPUTER SCIENCE & ENGINEERING DEPARTMENT
NATIONAL INSTITUTE OF TECHNOLOGY AGARTALA
January, 2022

Dedicated To

To our Loving Families for their kind love & support.
To our project supervisor Mr. Anupam Jamatia for sharing his valuable knowledge, encouragement & showing confidence on us all the time.

"The rise of Google, the rise of Facebook, the rise of Apple, I think are proof that there is a place for computer science as something that solves problems that people face every day."

-Eric Schmidt (Executive Chairman, Google)

REPORT APPROVAL FOR B.TECH

This report entitled "*Enhancement of TsunamiUDP Protocol*", by Akash Panda (11UCS004), Anand Solanki (11UCS008), Ankur Pratap (11UCS010), Shubham Upadhyaya (11UCS057), Supriya Sarkar (11UCS063), is approved for the award of ***Bachelor of Technology*** in ***Computer Science & Engineering***.

Mr. Anupam Jamatia
(Project Supervisor)
Assistant Professor
Computer Science and Engineering Department
NIT Agartala

Mr. Rajib Chowdhuri
(Project Coordinator)
Assistant Professor
Computer Science and Engineering Department
NIT Agartala

Mr. Mrinal Kanti Deb Barma
(Head of the Department)
Assistant Professor
Computer Science and Engineering Department
NIT Agartala

Date:_____
Place:NIT Agartala

DECLARATION

We declare that the work presented in this report titled "*Enhancement of Tsunami UDP Protocol*", submitted to the Computer Science and Engineering Department, National Institute of Technology, Agartala, for the award of the ***Bachelor of Technology*** degree in ***Computer Science & Engineering***, is an extension of existing work. We have not submitted the same work for the award of any other degree. In case this undertaking is found incorrect, I accept that my degree may be unconditionally withdrawn.

January 2022
Agartala

Akash Panda

Anand Solanki

Ankur Pratap

Shubham Upadhyaya

Supriya Sarkar

CERTIFICATE

It is certified that the work contained in the report titled "*Enhancement of Tsunami UDP Protocol*", by Akash Panda (11UCS004), Anand Solanki (11UCS008), Ankur Pratap (11UCS010), Shubham Upadhyaya (11UCS057), Supriya Sarkar (11UCS063), has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

Mr. Anupam Jamatia	Mr. Mrinal Kanti Deb Barma
(Project Supervisor)	(Head of the Department)
Assistant Professor	Assistant Professor
Computer Science and Engineering Department	Computer Science and Engineering Department
NIT Agartala	NIT Agartala

Acknowledgement

We would like to take this opportunity to express our deep sense of gratitude to all who helped us directly or indirectly during this project work.

Firstly, we would like to thank our supervisor, **Mr. Anupam Jamatia**, Assistant Professor for being a great mentor and the best adviser we could ever have. His advise, encouragement and critics are source of innovative ideas, inspiration and causes behind the advancement of the project. The confidence shown on us by him was the biggest source of inspiration for us. It has been a privilege working with him.

We are highly obliged to all the faculty members of Computer Science and Engineering Department for their support and encouragement. We also thank our Director **Dr. Gopal Mugeraya** and H.O.D, CSED **Mr. Mrinal Kanti Deb Barma** for providing excellent computing and other facilities without which this work could not achieve its quality goal.

Finally, we are grateful to our **parents** for their support. It was impossible for us to advance in work without their love, blessing and encouragement.

- Akash Panda

- Anand Solanki

- Ankur Pratap

- Shubham Upadhyaya

- Supriya Sarkar

List of Figures

List of Tables

Abstract

One of the biggest challenges facing companies/firms/research units that want to leverage the scale and elasticity of data for analytics or research is how to move their data into the cloud. Its increasingly common to have data-sets that are multiple petabytes. Moving data of this magnitude can take considerable time. To accomplish this, Tsunami-UDP protocol comes into action.

Tsunami UDP Protocol: A fast user-space file transfer protocol that uses TCP (Transmission Control Protocol) control and UDP (User Datagram Protocol) data for transfer over very high speed long distance networks, designed to provide more throughput than possible with TCP over the same networks. This project report provides an attempt to enhance current standards of Tsunami-UDP protocol. We have taken it as a challenge to integrate Multi-threading, a TUI/GUI (Terminal User Interface/Graphical User Interface), CLI (Command Line Interface) and prediction of data delivery to the current standard of Tsunami-UDP protocol.

Contents

CHAPTER 1

Introduction

Professionals working in various fields such as astronomy, biology, research, industry etc. need to have access to vast volumes of data and information (also referred to as big data) for analysis, evaluation and research work. So this much data may need to transfer over long distances to allow collaboration among professionals around the world and work efficiently and effectively. Hence it is important for them to have appropriate and suitable transfer protocols to optimize data transfer speed. For improving the performance of data transfer, providing only higher link capacity is not going to help much because the data transfer protocols have certain limitations. It is essential to select an efficient data transfer protocol that can efficiently use the available capacity, and accelerate the overall data transfer process. For the very same purpose Tsunami-UDP protocol comes into action, but before we study that, we need to have a little insight about TCP and UDP protocols and their limitations and how Tsunami-UDP protocol came into existence.

1.1 Transmission Control Protocol

TCP is a well defined standard that tells how network conversation can be established and well maintained through which application programs on either side can communicate and exchange data. TCP works along with the Internet Protocol (IP), which defines how computers at sender and receiver send packets of data to each other. Together, TCP and IP are the basic rules defining the Internet. TCP is defined by the Internet Engineering Task Force (IETF) in the Request for Comment (RFC) standards document number 793. TCP is a connection-oriented protocol, which means a connection is established and maintained until the communication is intact i.e. until the application programs at each end have finished exchanging messages. It determines how to break application data into packets that networks can deliver, sends packets to and accepts packets from the network layer, manages flow control, and because it is meant to provide error-free data transmission handles retransmission of dropped or garbled packets as well as acknowledgement of all packets that arrive. In the Open Systems Interconnection (OSI) communication model, TCP covers parts of Layer 4, the Transport Layer, and parts of Layer 5, the Session Layer.

1.2 User Datagram Protocol

Unlike TCP, the User Datagram Protocol (UDP) does not present data as a stream of bytes, also there is no requirement of establishing a connection with another program in order to exchange information. Data is exchanged in discrete units called datagrams, which are similar to IP datagrams. In fact, the only features that UDP offers over raw IP datagrams are port numbers and an optional checksum to maintain data integrity. Transmission, using this protocol is susceptible to threats lying in the transmitting medium UDP is sometimes referred to as an unreliable protocol because when a program sends a UDP datagram over the network, there is no way for it to know that it actually arrived at its destination. This means that the sender and receiver must typically have their own application protocol on top of UDP. Much of the work that TCP does transparently (such as generating checksums, acknowledging the receipt of packets, retransmitting lost packets and so on) must be performed by the application itself.

1.3 Need For Tsunami-UDP Protocol

With the limitations of UDP, you might wonder why it's used at all. UDP has the advantage over TCP in two critical areas: speed and packet overhead. Since TCP is a reliable protocol, it goes through great lengths to ensure that data arrives at it's destination intact, and as a result it exchanges a fairly high number of packets over the network i.e. retransmission of packets take place. UDP doesn't have this overhead, and is considerably faster than TCP. In those situations where speed is paramount like in video conferencing, or the number of packets sent over the network must be kept to a minimum, UDP is the solution. Simply relying on TCPs congestion control may result in inefficient utilization of long fat pipes due to its conservative behavior when handling packet loss events. Understanding behavior of TCP and UDP based high speed data transfer protocols will allow researchers to choose a protocol that best suits their network environment, which may dier from one research site to another.

1.4 Tsunami-UDP Protocol

Because of limitations of TCP and UDP protocols, need for Tsunami UDP protocol was felt. The authors were working for the launch of the Global Terabit Research Network [11]. A launch demonstration was there at a meeting in Brussels and they wanted to do something tacky and worth memorable. They had demonstrated wire-rate gigabit Ethernet transfers in their lab using normal Ethernet MTUs and were confident they could easily achieve more than 500Mbps. One PC was shipped to Belgium and one to Seattle. Once they were set up, the testing began. Because of a 3 percent packet loss, the rates varied from a few tens of Mbs to a very few hundreds of Mbs. Less than one week before the demo, the Lab decided they were going to have to design their own protocol. Less than 3 days after a white board diagram, there was a working prototype and a few days later the demo managed to average over 800Mbps for 17 hours and 40 minutes. From the general comments and the amount of time involved, this was probably a UDP blast with a minimum of other features.

CHAPTER 2

Related Works

With fiber optic interconnections, processor speeds in gigahertz, and network interface cards of 10 gigabits per second, hardware did not seem to be the limiting factor in data transfer. It had become obvious that the protocols that were the basis of networking are not suited to the technology of the current generation networking [4]. The most widely used protocol for data transfer over IP networks is TCP because of the fact that TCP guarantees delivery in order, no loss of data, and fairness. Unfortunately, the implementations of these features had made TCP ill-suited to get optimal bandwidth from high-speed, high latency network links [4]. In industry, Cloud computing and Big Data had been catching everybodys attention lately because of their resourcefulness. But Cloud Computing involves transmission of large amount of datasets (Big Data) from one geographical location to other, so they had to be complimented with some protocol that makes this data transmission fast. In terms of performance, Tsunami-UDP protocol did provide better transmission than any TCP protocol because of being a UDP protocol from core but using transmission controls on top of it [5]. This is why AWS (Amazon Web Services) had been using Tsunami-UDP protocol to transfer Big Data from Amazon EC2 to Amazon S3 [6].

But current version of Tsunami-UDP protocol did lack over other traditional TCP and UDP protocols in terms of functionalities. Current version of Tsunami-UDP protocol did lack in

multithreading, TUI/GUI, CLI(Command Line Interface), predictability of data delivery, support to Jumbo packets etc [7]. So we got motivated to enhance the current version of this protocol by integrating multithreading, TUI/GUI, CLI and predictability of data delivery.

CHAPTER 3

Transmission Control Protocol VS User Datagram Protocol

Now let us discuss about the TCP and UDP protocols and their comparison. TCP is connection oriented once a connection is established, data can be sent in either direction. UDP is a simpler, connectionless Internet protocol. Multiple messages (called datagrams) are sent as packets in chunks using UDP.

3.1 Comparison

	Transmission Control Protocol	User Datagram Protocol
Connection	TCP is a connection-oriented protocol.	UDP is a connectionless protocol.
Function	As a message makes its way across the internet from one computer to another. Since it is connection based, proper connection is required.	UDP is also a protocol used for message transport or transfer. It is not connection based which means that one program can send a load of packets to another and that would be the end of the relationship. No overhead for connection is required.
Usage	TCP is generally used for those applications that require high reliability, and transmission time is relatively less critical such as in e-mailing.	UDP is suitable for applications that require fast, efficient transmission, such as games and video conferencing. UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
Ordering of data packets	TCP rearranges data packets in the order specified.	UDP has no inherent order as all packets are independent of each other. UDP is not responsible for ordering of packets.If ordering has to be done, it has to be managed by the application layer.

Speed of transfer	The speed for TCP is slower than that of UDP.	UDP is faster than TCP because there is no error-checking for packets at Data link layer.
Reliability	TCP ensures that the data transferred remains intact and arrives in the same order or fashion in which it was sent.	UDP does not even guarantee that the messages or packets sent would reach.
Header Size	TCP header size is 20 bytes	UDP Header size is 8 bytes.
Common Header Fields	Destination port, Source port, Check Sum	Destination port, Source port, Check Sum
Streaming of data	Data is read as a byte stream or contigious stream of bytes, no clear distinguishing indications are transmitted to signal message (segment) boundaries.	Packets (Datagrams) are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.
Weight	TCP is heavy-weight. Before any user data could be sent, there is a requirement to set up a socket connection for which TCP uses three packets. TCP handles both reliability and congestion control.	Unlike TCP, UDP is lightweight. There is no particular ordering of messages, no tracking connections, etc. It is simply a small transport layer designed or implemented on top of IP.
Data Flow Control	TCP does Flow Control i.e manage the rate of data transmission. TCP handles both reliability and congestion control.	UDP does not have any option for flow control which means low reliability and low congestion control.

Error Checking	TCP does error checking and do recovery by either retransmission or error correction.	UDP does error checking, but no recovery options.
Fields	1. Sequence Number, 2. AcK number, 3. Data offset, 4. Reserved, 5. Control bit, 6. Window, 7. Urgent Pointer 8. Options, 9. Padding, 10. Check Sum, 11. Source port, 12. Destination port	1. Length, 2. Source port, 3. Destination port, 4. Checksum
Acknowledgement	Acknowledgement segments	No Acknowledgment
Handshake	SYN, SYN-ACK, ACK	No handshake (connectionless protocol)

Table 1: Differences between TCP and UDP [10]

3.2 Other Comparisons

Differences in Data Transfer Features

TCP: A reliable and ordered delivery of a stream of bytes from user to server or vice versa is ensured by TCP.

UDP is not dedicated to end to end connections and communication does not check readiness of receiver. So it is not reliable.

Connection

TCP is a heavy weight connection. It requires three packets for a socket connection and handles both congestion control and reliability.

UDP is a lightweight transport layer designed on top of IP. There are no tracking connections or ordering of messages.

Method of transfer

TCP: Data is read as a byte stream and message is transmitted to segment boundaries.

UDP: Here messages are in form of packets which are sent individually and on arrival are checked for their integrity using checksum. Packets have defined boundaries while data stream has none.

Working Of TCP and UDP

A TCP connection is established via a three way handshake, which is a process of initiating and acknowledging a connection. Once the connection has been established, data transfer can begin. After transmission is done, the connection between sender and receiver is terminated by closing of all established virtual circuits.

UDP uses a simple transmission model without any inbuilt hand-shaking dialogues which can guarantee reliability, ordering, or data integrity. Thus, UDP provides a service which is unreliable and datagrams may or may not arrive out of order, appear duplicated, or go missing without any notice. UDP assumes that error checking and correction is either not necessary or performed in the application, and hence avoids the overhead of such processing at the network interface level which results in increased speed. Unlike TCP, UDP is compatible with packet broadcasts (sending to all on local network) and multicasting (send to all subscribers).

Different Applications of TCP and UDP

Web browsing, email and file transfer are some common applications that make use of TCP. TCP is used for controlling segment size, rate of data exchanged, flow control and network congestion. TCP is preferred over UDP where error correction facilities are required at network interface level.

UDP is largely used by those applications which are time sensitive as well as by servers that answer small queries from huge number of clients. UDP is compatible with packet broadcast - sending to all on a network and multicasting sending to all subscribers. UDP is commonly used in Domain Name System, Voice over IP, Trivial File Transfer Protocol and online games.

CHAPTER 4

Performance Comparison of Tsunami-UDP protocol with other UDP-based Protocols over Fast Long Distance Network

4.1 Efficiency and Stability

The protocol efficiency can be measured by Utilization of network resources which are currently available. The bandwidth utilization is important in case of FLDNet (Fast Long Distance Network) because the link in this case is costly. High speed with stable throughput is very important in case of most e-Science applications. Several experiments are made in typical FLDnet configuration (RTT>300 m sec, loss rate = 0.1 percent). TCP has low throughput (about 1.05 Mbps) [1]. Figure 4.1 shows the throughput of the UDP-based protocols while transferring payloads of different size.

TCP has much smaller throughput than all three UDP-based protocols. RBUDP (Reliable Blast UDP) and Tsunami has throughput up to 660 Mbps which is very high as compared to others

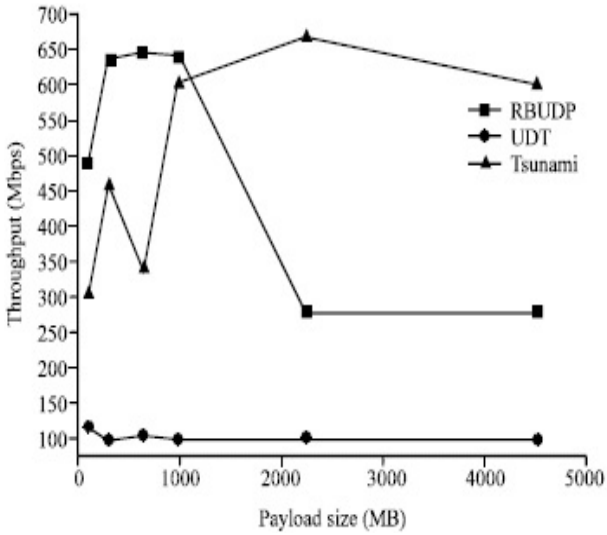

Figure 4.1: Comparison of throughput of UDP based protocols [1]

[1]. But in case of large file transfer (>2 GB), we can see drop in throughput of RBUDP which is obvious, because for transferring large file RBUDP use stream. In case of Tsunami, there is no limit on file size while throughput remains high. But block size and buffer size are the parameters which should be at optimal value. The throughput in case of UDT (UDP based application level Data Transfer Protocol) is relatively low. And there is no considerable effect on UDT throughput because of payload size.

As for stability, there is no dramatical change in the throughputs of RBUDP and Tsunami with time. But UDT is unstable. Figure 4.2 represents the changes in throughput of UDT throughput with time.

There are similarities in throughput of UDT and TCP, its because congestion control in UDT is made by using DAIMD (Delay-based Additive Increase Multiplicative Decrease) algorithm. There is dramatical reduction in the sending rate whenever there is a packet loss. Thus UDT has difficulty in keeping throughput stable in case of noisy link.

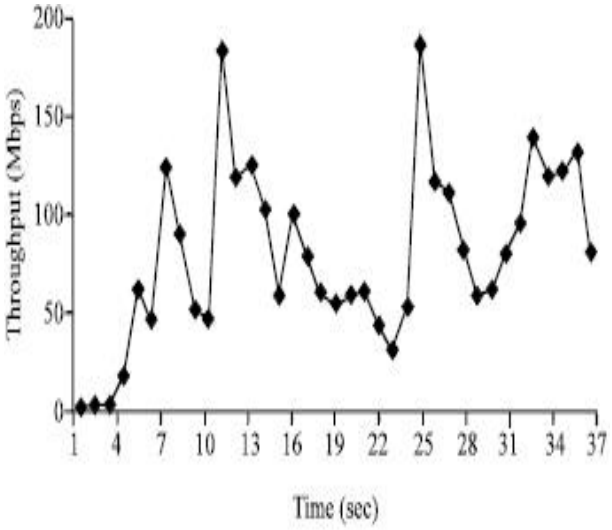

Figure 4.2: UDT Throughput vs time [1]

4.2 Throughput vs Loss

On FLDnet, in case of end-to-end lightpath link, the rate of packet loss is very low because this type of dedicated link has no congestion and the error rate in case of optical link is much low. But, sometimes, packet losses occur due to several reasons like network equipment problems or performance problems in terminal system. So, the several loss rate conditions are applied to investigate throughputs. [1]

It can be represented that, in complete absence of loss rate, high throughput can be achieved by TCP (361.28 Mbps) too (Buffer size of TCP is adjusted). However, in the presence of even a much small loss rate, there is a dramatical reduction in throughput. Comparatively, protocols based on UDP have throughput much higher than that of TCP. In case of RBUDP and Tsunami, the throughput doesn't get affected much because of the packet loss rate. But for UDT, the impact is effective. It's just because the mechanism for congestion control of UDT has great sensitivity for the packet loss. In case of higher loss rate, performance of UDT is also poor like TCP.

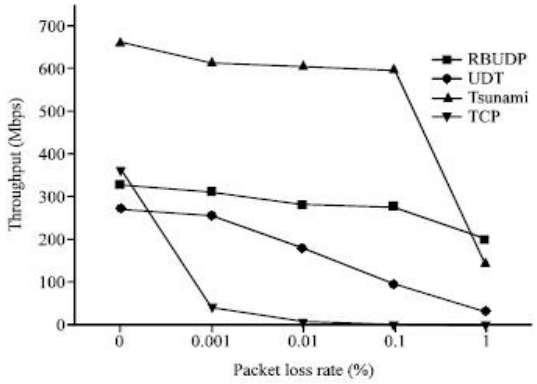

Figure 4.3: Throughputs under different loss [1]

4.3 Throughput vs RTT (Round Trip Time)

On FLDnet, there is transfer of bulk data between different terminals throughout the world. So, different propagation delays occur if the distance between terminals is different. Therefore, different RTT conditions are applied to test throughputs of these protocols. It can be observed

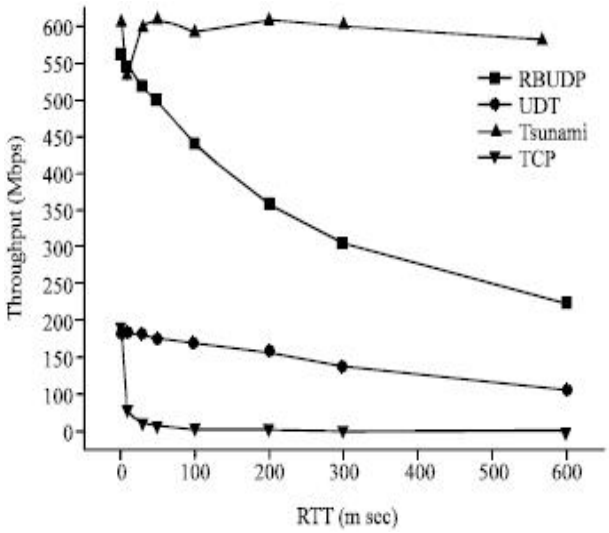

Figure 4.4: Throughputs under different RTT conditions [1]

that in the case where RTT is very small, performance of TCP is better (<2 m sec in LAN), but performance becomes poor in case of big RTT (in case of WAN and FLDnet). Still, protocols based on UDP have much higher throughput than that of TCP. As the increasing of RTT occurs, there is a decrease in the throughputs of UDT and RBUDP. The throughput in case of Tsunami protocol is higher than others and changes much less as the increase in RTT.

	UDT	**Tsunami**	**UFTP**	**GridFTP**
Multithreaded	no	no	no	yes
Protocol Overhead	10%	20%	10%	6-8%
Encryption	no	no	yes	yes
C++ Sourcecode	yes	yes	yes	yes
Command line	no	no	yes	yes
Binaries	no (source code only)	no (source code only)	no (CLI only)	no (CLI only)
UDP based point-to-point	yes	yes	yes	no
Firewall friendly	Partial (no auto-detection)	no	Partial (no auto-detection)	no
GUI client	no	no	no	no
Server with secure user accounts	no	no	no	yes
Congestion control	yes (UDP blast-mode preferred)	yes (limited)	yes (congestion control file has to be specified before the transfer starts)	yes-using TCP
Automatic retry and resume	no	no	no (manual resume yes)	yes
Jumbo Packets	yes	no	yes (upto-8800 bytes)	yes

IPv6 Support	yes	no	no	no
Support for any packet loss	no	no	no	yes
Support for low bandwidth	no	no	no	no
High packet loss i.e. satellite	no	no	no	no

Table 2: Comparison between for products using UDP based data transfer [8]

CHAPTER 5

Report on Present Investigation

5.1 Problem Statement

The present version of Tsunami-UDP requires to be built from source (no binaries have been provided). This is only a source code implementation of the sender and receiver. All the functionality involving authentication of user, reporting, monitoring and file management have to be implemented by the programmer. This open source version of implementation of the source code lacks in some aspects compared to similar paid products - no multithreading, no GUI/TUI support, no CLI and no prediction of data delivery, etc. Our focus is to integrate some of these services into the existing Tsunami-UDP open source project.

5.2 Our work

Implementation of multi-threading : The server was designed in such a way that it runs as a single process to service the requests. Whenever it receives a request it forks a separate child

process to service that request. In fact, the process creation method is time consuming and resource intensive, however if the new process will perform the same task as the existing processes, why incur all that overhead?
We can design the server in such a way that on receiving a clients request it would select a new thread from available threadpool to service the request and resume listening for additional client requests. Hence, a thread will be chosen everytime from threadpool to service the request.
Before our implementation, if two clients are connected to the server, then two separate instances of the server process would run to service the requests by the two clients simultaneously. But after our implementation of multithreading, now multiple clients are serviced by a single server process. We can check the processes running in the system with the ps command.
Presently, for the first step a simple multi-threaded model is followed here. A thread pool is taken.

```
thpool_t* threadpool; /* make a new thread pool structure */
threadpool=thpool_init(MAX_THREADS); /* initialise it to MAXTHREADS
number of threads */
thpool_add_work(threadpool, (void*)client_handler,
(void*)(&thread_arguments)); /*Function to Run the thread */
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; /* Create
mutex variable */
thpool_t* thpool_init(int threadsN) /* Initialise thread pool */
void thpool_thread_do(thpool_t* tp_p) /* What does the thread do */
void thpool_destroy(thpool_t* tp_p) /* Destroy the threadpool */
```

Figure 5.2 shows the snapshot of ps command while running current version of Tsunami UDP and Figure 5.3 shows the snapshot of ps command while running our enhanced version of it. We can see that while it was connected with two different clients, two different process of tsunamid are being forked to handle the different clients along with a server process. But in our enhanced version, a single process handles all the two clients in a multithreaded approach, allocating a thread to each client to service its request.

Implementation of Terminal User Interface:
The TsunamiUDP presently requires to be compiled and then run the binaries created directly and had no user friendly interface. We have developed a TUI interface for this protocol using

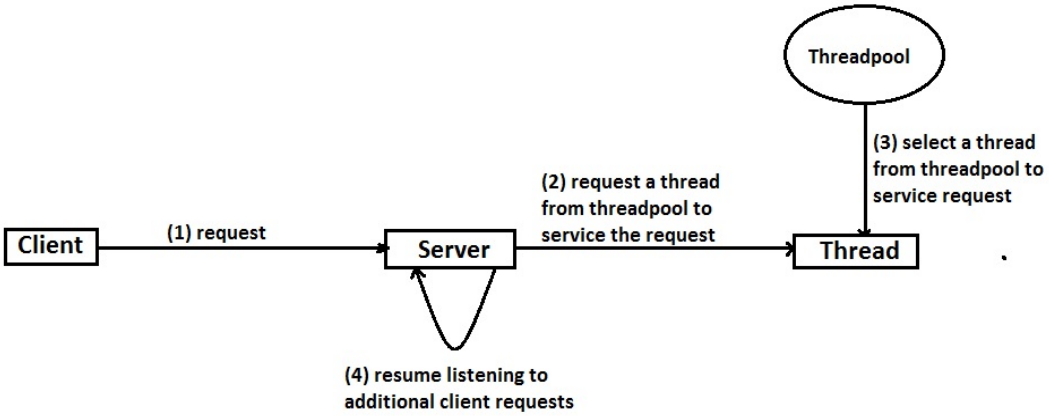

Figure 5.1: Multithreaded Server Architecture

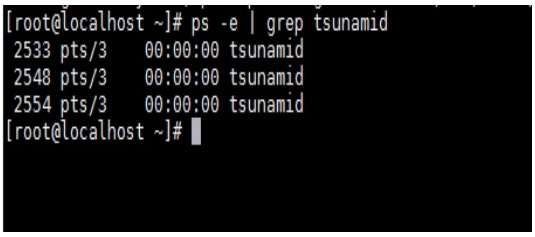

Figure 5.2: Snapshot of ps commad while running current version

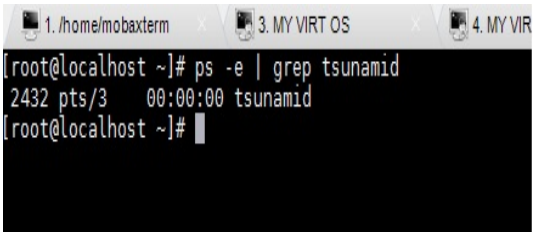

Figure 5.3: Snapshot of ps command while running our improved version

Python and Shell Scripting which automates the whole process in a user friendly way.
We can now use the whole complex process of usage (described in Appendix A) of the protocol in a very simple and user friendly manner.

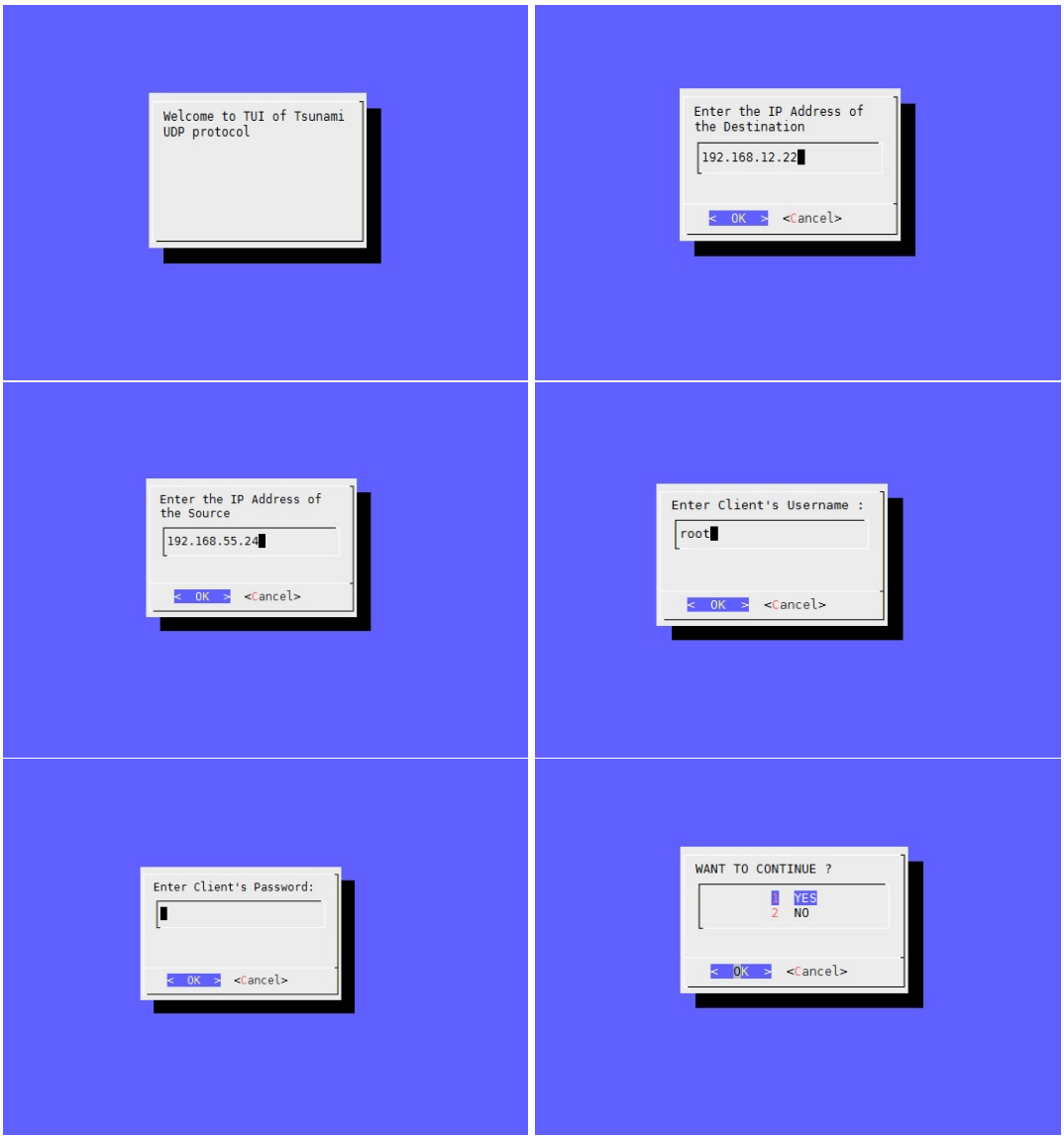

Figure 5.4: Screenshots of TUI Application

Implementation of Command Line Interface (CLI) :
A CLI is an user interface to an operating system or an application where responses to visual prompt are made by user by typing in a command on a specified line, responded back from the system, and then enters another command, and so forth. But the present version of Tsunami-UDP doesn't have the interactive features of a CLI. We have tried to make the CLI more user

interactive and implemented many ideas in the project, as present in many Command Line Interfaces (like in Bash shell, ftp, etc). Examples of some of the features are running through the history list with the UP and DOWN arrow keys, auto-completion of the commands and filename using TAB key and getting a list of commands and their help while using the application. We have implemented CLI using the GNU Readline Library, which provides a set of functions to be used by applications that allow users to edit command lines as they are typed in [12]. We have successfully implemented the CLI in the Tsunami-UDP application and has in turn made it more user interactive.

CHAPTER 6

Conclusion and Future Direction of Work

Conclusion

Multithreading has been implemented in Tsunami UDP and now the process selects a thread of execution for serving a new request everytime instead of forking a new child process. The Terminal User Interface has given altogether a whole new look and feel to the Tsunami UDP protocol as it is now much easier to use without the hassle of configuration and other formalities that had to be carried out. We have successfully implemented the CLI in the Tsunami-UDP application and has in turn made it more user interactive in the command line. The project has been uploaded to `http://github.com/akashpanda123/tsunami-udp-enhanced` and been made open source to motivate further enhancement in this.

Future Direction of work

Tsunami-UDP still lacks in various fields, like it doesn't support predictability of data delivery, is not firewall friendly, doesn't have server with secure user accounts and doesn't have the facility of automatic retry and resume in case of failure. This is the main reason why tsunami is not up to the mark when compared to other paid transfer protocols.

References

[1] Li J., Qian H., Ren Y., Tang H. (2009), "Performance Comparison of UDP-based Protocols Over Fast Long Distance Network". Information Technology Journal 8, Beijing, China, pp. 600-604.

[2] Brownlee N., Mahanti A., Se-young Yu. (2013). "Comparative performance analysis of high-speed transfer protocols for big data", 2013 IEEE 38th Conference on Local Computer Networks (LCN), Sydney, NSW, pp. 292-295.

[3] Li J., Ren Y., Yue Z (2011), "Performance Evaluation of UDP-based High-speed Transport Protocols", 2011 IEEE 2nd International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, pp. 69-73.

[4] Steven M. Carter, Tom Dunigan, Florence Fowler (2004), "An Evaluation of UDP Transport Protocols".

[5] https://skillupjapan.co.jp/news/rd_product/pdf/UDPFiletransfer_Final.pdf

[6] http://blogs.aws.amazon.com/bigdata/post/Tx33R88KHCWEOHT/Moving-Big-Data-into-the-Cloud-with-Tsunami-UDP

[7] http://www.atomrain.com/it/it/open-source-udt-tsunami-uftp-gridftp-paid-data-transfer-protocols

[8] http://www.filecatalyst.com/open-source-fast-file-transfers

[9] https://github.com/res0nat0r/tsunami-udp

[10] http://www.diffen.com/difference/TCP_vs_UDP

[11] http://www.csm.ornl.gov/dunigan/net100/udp/UDP_Tsunami.html

[12] http://cnswww.cns.cwru.edu/php/chet/readline/rltop.html

APPENDIX I

Installation and Usage of Tsunami-UDP

I.1 Installation

I. Dependencies

1. C and C++ Compiler :

```
yum install gcc-c++
```

2. Automake :

Automake package is required for the making binaries using make
If automake is not currently installed, then we can install it using

```
yum install automake
```

II. Get the code:

If the source code has not yet being downloaded, then we can do so with an anonymous CVS checkout:

```
cvs -z3 -d:pserver:anonymous@tsunami-udp.cvs.sf.net:/cvsroot/
tsunami-udp co -P tsunami-udp
```

Later we can use the usual

```
cvs update
```

command to get and merge updates from the CVS into our local copy of the code.

III. Compilation

If we want to build Tsunami for a big endian i.e. non-Intel platform, we have to first edit ./common/Makefile.am and have to see instructions there.

For compiling tsunami, the normal process is:

```
$ ./configure
$ make
```

From the configure output errors, we have to find all those missing packages that we had never even heard of and install them, then try ./configure again.

If there are ./configure Makefile.in complaints or similar, try

```
$ ./recompile.sh
```

That should build the entire tsunami, including everything in the subdirectories (in /server, /client, etc).

An optional

```
$ make install
```

will install the tsunami binaries onto the system. The binaries that will be installed are:

App	Name	From
Tsunami server	tsunamid	./server/tsunamid
Tsunami client	tsunami	./client/tsunami
Realtime server	rttsunamid	./rtserver/rttsunamid
Realtime client	rttsunami	./rtclient/rttsunami

The binaries will be placed into /usr/local/bin or similar.

Contents of individual subdirectories can be recompiled with for example

```
$ cd client
$ make clean
$ make
```

(Note that you first have to compile the entire source tree via ’./configure’ and ’make’. Only after that you can recompile individual subdirectories without problems)

I.2 Usage

–Server–

On the server PC, we have to the directory where we have the files that should be served, and then start the Tsunami server with e.g.:

```
$ cd /where/my/files/are
$ ./tsunamid
```

The server serves files from its current working directory.

To allow clients to use the ”GET *” command, start the Tsunami server with:

```
$ tsunamid *
or
$ ./tsunamid fileToServe1 fileToServe2 ...
```

–Client–

On the client PC, start the command line client:

```
$ cd /where/to/receive/files
$ ./tsunami
```

The client will store downloaded files into the current working directory.

Usual commands in the client are like crude FTP:

```
tsunami> connect [IP ADDRESS]
tsunami> get filename
tsunami> get *
```